INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS, VOL. 25, 183–203 (1997)

A NEW OPTIMAL SHAPE DESIGN PROCEDURE FOR INVISCID AND VISCOUS TURBULENT FLOWS

BIJAN MOHAMMADI*

INRIA, BP 105, F-78153 Le Chesnay, France

SUMMARY

A new approach for optimal shape design is introduced. The main ingredients are an unstructured CAD-free framework for geometry deformation and automatic differentiation (AD) in reverse mode. Transonic inviscid and viscous turbulent flows are investigated. Both two- and three-dimensional configurations are considered. These cases involve up to several thousand control parameters. © 1997 by John Wiley & Sons, Ltd.

Int. J. Numer. Meth. Fluids, **25**: 183–203 (1997).

No. of Figures: 23. No. of Tables: 0. No. of References: 22.

KEY WORDS: shape design; turbulent flow; automatic differentiation

1. MOTIVATIONS

Two key points in optimal shape design when using a gradient method are a good evaluation of the Jacobian of the cost function with respect to control parameters and a suitable shape and mesh deformation algorithm.

Concerning the Jacobian, it is clear that if the number of control parameters is large, an adjoint method is necessary. Adjoint methods are based on a linearization of the relation describing the dependence between the cost function and the control parameters. This is usually realized at the continuous level and not after discretization.[1,2] Therefore the inconsistencies (such as the numerical viscosity) of the numerical schemes cannot be exactly taken into account. The first motivation of this paper is to present our experience of automatic differentiation in reverse mode for optimal shape design applications. As in this technique the Jacobian of the discrete cost function is evaluated, the non-consistencies of the schemes are naturally taken into account. Moreover, the cost of the evaluation is independent of the number of control parameters as in a classical adjoint method.

For shape deformations the difficulty comes from the fact that CAD-based tools for shape description are quite complicated and difficult to take into account when evaluating the Jacobian. Our second motivation is to show that it is possible to consider all the mesh points on the body as control points in so far as we preserve the initial regularity of the shape during the design process. This makes it possible and easy to consider all kinds of shapes. In our optimization process the only geometrical entity available is a two- or three-dimensional unstructured mesh.

Our third motivation is to see the effect of using different gradients, based on different numerical

* Correspondence to: B. Mohammadi, INRIA, BP 105, F-78153 Le Chesnay, France

CCC 0271–2091/97/020183–21 $17.50 *Received 13 June 1996*

fluxes used for the solution of the Euler equations, on the optimization procedure. We applied automatic differentiation to Roe and Osher fluxes with the same second-order MUSCL construction. In Reference 3 we showed that having a precise gradient taking into account in particular the MUSCL construction affects not only the convergence but also the optimized shape. Here we want to see whether having a smoother numerical flux will have a similar effect. We also showed[3] through an inverse problem that our approach does not suffer from a change in the nature of the equations when passing from the transonic to the supersonic regime. In this paper we consider not only inviscid flows but also viscous turbulent configurations.

Our last motivation is to show that using these techniques it is possible to perform 3D optimization for compressible flows on a workstation (configured at 10 Mflops, 100 MB) through the night.

2. MODELS

We denote by $J(x, U(x))$ the cost function we want to minimize under geometric and aerodynamic constraints. Here x indicates the geometrical description of a configuration and the flow pattern $U(x)$ around this shape is the solution of the steady Navier–Stokes and k–ε[4] system of fluid dynamics in conservation form,

$$\nabla \cdot F(U) = S(U), \tag{1}$$

where U is the vector of conservative variables, i.e. $U = (\rho, \rho\vec{u}, \rho(C_v T + \frac{1}{2}|u|^2), \rho k, \rho\varepsilon)^T$, F represents the advective and diffusive operators and S contains the source terms of the k–ε model. This system has six equations in 2D (seven equations in 3D) for seven variables (eight variables in 3D) and is closed using the equation of state $p = p(\rho, T)$. We do not take into account turbulent contributions to the pressure and total energy.[5]

Turbulence modelling is done through a two-equation eddy viscosity model. Special wall laws including pressure and convection effects and valid up to the wall have been used. These wall laws are particularly suitable for separated and unsteady flows. For instance, we managed to capture the secondary eddy for a backward step and 'Strouhal'-type unsteadiness behind a cylinder.[6]

In an optimization process involving mesh deformation, wall laws are more suitable than low-Reynolds-number modelling (i.e. up to the wall). Indeed, this former approach requires much finer meshes and conformal mesh deformation will be harder to achieve.

3. NUMERICS

The numerical technique is based on a finite volume–Galerkin approach on unstructured meshes.[7–12] The convective part of the equations has been solved using a Roe[9] or Osher[12] approximate Riemann solver together with MUSCL reconstruction and Van Albada limiters.[10] The viscous term is discretized using a classical piecewise linear finite element method. The time-dependent equation

$$\frac{\partial U}{\partial t} + \nabla \cdot F(U) = S(U)$$

is marched in time to a steady state. Our time discretization is based on a four-stage Runge–Kutta scheme. Inflow and outflow boundary conditions are of characteristic type and for outflow boundaries care has been taken to correctly treat subsonic configurations. Details of these techniques can be found in References 7–12. A brief description of the governing equations and numerics is given in Appendix I.

4. OPTIMIZATION PROBLEM

We consider the following minimization problem:

$$\min_{x_c} J(x_c, U(x_c)),$$
$$E(x_c, U(x_c)) = 0, \quad g_1(x_c) \leqslant 0, \quad g_2(U(x_c)) \leqslant 0,$$

where $x_c \in \mathbb{R}^{nc}$ describes the control parameters, $U \in \mathbb{R}^N$ the flow, $E \in \mathbb{R}^N$ the state equation and $g_{1,2}$ direct ('geometrical', on x_c) and indirect ('physical', on $U(x_c)$, such as a given lift) constraints.

Indirect constraints have been taken into account in the cost function by penalty. Geometrical constraints are of three types. The first one is imposed by defining two limiting surfaces (curves in 2D) between which shape variations are allowed. As all shapes in this paper are of wing type, the second constraint is that the original plan-form should remain unchanged. This means for instance in 2D that the leading and trailing edges are frozen. The last constraint is that the volume of the wing has to be conserved during optimization. The first and second constraints have been taken into account by projection and the last constraint by penalty in the cost function.

4.1. The gradient

To find $\mathrm{d}J/\mathrm{d}x_c$, we can do one of the following.

1. Use finite differences:

$$\frac{\mathrm{d}J}{\mathrm{d}x_c}(i) = \frac{J(\vec{x}_c + \epsilon\vec{e}_i) - J(\vec{x}_c - \epsilon\vec{e}_i)}{2\epsilon}.$$

The difficulties are the choice of ϵ and a cost proportional to nc.

2. Avoid choosing ϵ by using

$$\frac{\partial E}{\partial U}\frac{\partial U}{\partial x_c} = -\frac{\partial E}{\partial x_c}, \tag{2}$$

which means using an implicit flow solver after changing the right-hand side. After substitution we have

$$\frac{\mathrm{d}J}{\mathrm{d}x_c} = \frac{\partial J}{\partial x_c} + \frac{\partial J}{\partial U}\frac{\partial U}{\partial x_c}.$$

However, the cost problem remains because $\partial E/\partial x_c \in (N, nc)$ and we need to solve (2) nc times.

3. Introduce the Lagrangian $L = J + pE$ and by the optimality conditions get

$$\frac{\partial L}{\partial U} = \frac{\partial J}{\partial U} + p\frac{\partial E}{\partial U} = 0, \tag{3}$$

where to get p we again use the solver

$$\frac{\partial E}{\partial U}X = f.$$

Hence after substitution we have

$$\frac{\mathrm{d}J}{\mathrm{d}x_c} = \frac{\partial L}{\partial x_c} = \frac{\partial J}{\partial x_c} + p\frac{\partial E}{\partial x_c}.$$

Here the cost is independent of ne but we still need to compute $\partial E/\partial x_c$. This can be easily done using AD in direct mode for instance.[13]

4. Use AD in reverse mode to compute dJ/dx_c to avoid solving (3) and assembling $\partial E/\partial x_c$. In this approach the lines of the programmes describing the relations between the variation in the design variables and the cost function including the grid and the flow equations are considered as constraints. We associate with each of them a Lagrange multiplier p and we construct an augmented Lagrangian L. The values of the Lagrange multipliers are obtained from the condition that the first variations in L with respect to intermediate (or dependent) variables vanish. The solution can always be obtained simply by back substitution (hence the notion of reverse mode). Once the Lagrange multipliers are evaluated, the gradient of L can be easily calculated (a brief description of AD is given in Appendix II).

We use the automatic differentiator Odyssée developed at INRIA.[14–16] Both direct differentiation, producing a Jacobian matrix or a gradient vector, and reverse mode, computing the cotangent linear application, are implemented in Odyssée. In References 17–19 the gradients obtained by Odyssée have been compared with those obtained using finite differences for similar problems.

This gradient is then used in a gradient method to solve the optimization problem. Our minimization tool is quite simple. It is based on a gradient method with a fixed descent step. As we said, we use projection to take into account local geometrical constraints, while global constraints are taken into account in the cost function using Lagrange multipliers.

More precisely, the algorithm is as follows (we denote $J(x_c, U(x_c))$ by $J(x)$):

$$\begin{gathered} x_0 \quad \text{given,} \\ \text{for} \quad n = 1, 2, \ldots \quad \text{do} \\ x^n = P[x^{n-1} - \lambda \nabla_{x^*} J(x^{n-1})], \end{gathered}$$

where P is the projection operator and

$$\nabla_{x^*} J = \frac{\partial J}{\partial x} + \left(\frac{\partial J}{\partial U}\right)^{\mathrm{T}} \frac{\partial U}{\partial x}.$$

5. GEOMETRICAL TOOLS

We describe the set of tools needed for mesh deformation from a variation in control parameters x_c:

$$\delta x_c \to \delta x_w \to \delta x_m,$$

where x_w are the discretization points on the geometry and x_m are the internal mesh nodes. To get the Jacobian of the cost function with respect to control parameters, all these tools have to be differentiated. However, the systems involved in this section are solved using iterative schemes. This is a real problem when using the reverse mode of AD. Indeed, the intermediate states have to be stored for the adjoint computation (see next section).

5.1. Shape deformation

Usually, for 2D applications, control points are fitted by splines. Splines have two features.

1. They permit that $nc \ll nw$, with nw the size of x_{w}.
2. They smooth the variations in control points when propagating to the other body points.

However, general 3D surface splines are quite complicated to handle, especially on unstructured meshes. Indeed, this involves CAD concepts and deriving these objects is more difficult than the fluid

solver for instance. We introduce an unstructured framework for geometry modifications based on the following points.

1. All the nodes on the shape are control points (i.e. $nc = nw$).
2. To avoid oscillations, a smoothing operator is defined over the shape. This can be, for instance, a few 'local' Jacobi iterations to solve the system

$$(I - \varepsilon\Delta)\delta\tilde{x}_w = \delta x_w, \tag{4}$$

where $\delta\tilde{x}_w$ is the smoothed shape variation for the shape nodes and δx_w is the variation given by the optimization tool (see below). By 'local' we mean that if the predicted shape is locally smooth, it remains unchanged during this step.

The importance of this step can be understood by the following argument.

We want the variation $\delta x_w \in C^1(\Gamma)$ if Γ designs a variety of dimension $n-1$ in a domain $\Omega \in \mathbb{R}^n$. From Sobolev inclusions we know that $H^n(\Gamma) \subset \mathrm{C}^1(\Gamma)$. It is easy to understand that the gradient method we use does not necessarily produce $\mathrm{C}^1(\Gamma)$ variations δx_w and therefore we need to project them into $H^n(\Gamma)$ (an example of this is given in Figure 1). This means that the projected variations $\delta\tilde{x}_w$ are the solution of an elliptic system of degree n. However, as we are using a P^1 discretization, a second-order elliptic system is sufficient, because the edges of the geometry are considered as constraints for the design. Therefore we project the variations only into $H^2(\Gamma)$ even in 3D.

Once the variations x_w are known, we have to expand them over all the mesh. This is done by solving a volumic elasticity system which is also of the form of (4).

6. LIMITATIONS WHEN USING AD

The limitation of the reverse mode of AD comes from the required memory for Lagrange multipliers (see Appendix II) and intermediate variables.

The need for intermediate variable storage can be understood by the following argument.

If nc is the number of controls, let f be a function f: $\mathbb{R}^{nc} \to \mathbb{R}$ such that

$$f(x) = f_3 \circ f_2 \circ f_1 \circ f_0(x).$$

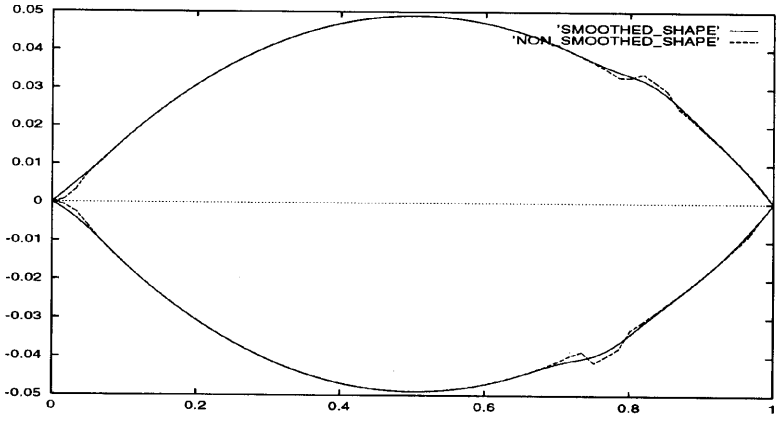

Figure 1. Smoothed and unsmoothed shapes. We can see that the gradient jumps through shocks and also produces a non-smooth shape in leading edge regions. This is the result of the first iteration of the optimization. If we continue, the shape becomes more and more unsmooth

For instance, in our case, f_0 smooths the control point variations, f_1 propagates the control point variations over all the mesh, f_2 computes the flow and f_3 evaluates the cost. The reverse mode produces the Jacobian $D^{\mathrm{T}}f$: $\mathbb{R} \to \mathbb{R}^{nc}$ as

$$D_x^{\mathrm{T}} f = D_x^{\mathrm{T}} f_0 \circ D_{f_0(x)}^{\mathrm{T}} f_1 \circ D_{f_1 \circ f_0(x)}^{\mathrm{T}} f_2 \circ D_{f_2 \circ f_1 \circ f_0(x)}^{\mathrm{T}} f_3.$$

We can see that we have to store (or recompute) the intermediate states (i.e. $f_0(x)$, $f_1 \circ f_0(x)$ and $f_2 \circ f_1 \circ f_0(x)$) before making the product.

This can be a real problem when using an iterative method (for time integration for instance). Indeed, in this case the intermediate states cannot be recomputed as they will generate a cost of the order

$$T = (\text{number of iterations})^2 \quad T_{\text{one iteration}}.$$

Therefore they have to be stored.

As we said, our flow solver is explicit, with one external loop for time integration of size *KTMAX* as follows:

```
External Loop in time 1,...,KT
  Loop over triangles (tetrahedra) 1,...,NT
    NAT affectations
  End loop NT
  Loop over edges 1,...,NE
    NAE affectations
  End loop NE
  Loop over nodes 1,...,NN
    NAN affectations
  End loop NN
End external loop KT
  cost evaluation
```

Inside each time step we have loops on nodes, segments and tetrahedra (triangles in 2D) of sizes *NN*, *NS* and *NT*. Inside each internal loop we have some affectations of the type

```
new_var=expression(old_var)
```

describing the spatial discretization. The numbers of these affectations are *NAN*, *NAS* and *NAT*. The required memory to store all the intermediate variables is therefore given by

$$M = KTMAX \times [(NN \times NAN) + (NS \times NAS) + (NT \times NAT)].$$

This is out of reach even for quite coarse meshes. To give an idea, for the 3D configuration presented here we have

$$KT \sim 10^4, \quad (NN, NS, NT) \sim 10^5, \quad (NAN, NAS, NAT) \sim 10^2,$$

which makes $M \sim 10^{11}$ words ~ 100 GO, while the available memory is of the order of 100 MO.

6.1. Key points when using AD

To use AD in reverse mode in our optimization problem, we make use of the following remarks.

Steady flows. The first key remark is that the target applications are steady flows. Therefore, when computing the gradient by the reverse mode, it is sufficient to store only one state if we start with an initial state corresponding to the steady state for a given shape. This reduces the size of our problem by the number of time step (*KTMAX*):

$$M = [(NN \times NAN) + (NS \times NAS) + (NT \times NAT)].$$

Inter-procedural differentiation. The second important point is to use inter-procedural derivation. This means replacing what is inside an internal loop by a subroutine and deriving this subroutine. This will reduce the required memory to

$$M = NAN + NAS + NAT,$$

but will imply extra calls to subroutines.[16] In fact we need less than that, as this memory is allowed in a dynamical way.

Adjoint accuracy. We saw that, starting from the steady state for a given shape, we only need one iteration of the forward procedure before starting the reverse integration. Therefore we need to know what convergence is sufficient for the adjoint for the optimization to converge. Of course, we can integrate for a very long time, but this will not be optimal.

We denote by J^∞ the cost at convergence of the optimization and by J^n the cost at step n. We notice that at convergence we would like to have $\nabla J_h^\infty = 0$. Therefore, for the optimization to converge, it is sufficient for ∇J_h^n to be strictly decreasing. This gives the criterion we use in the reverse computation to stop the reverse time step loop. In other words, we have two different numbers of time step, one for the forward system (taken equal to one) and another for the adjoint system (very large), and we leave the reverse time integration loop once the above criterion is satisfied. We therefore need to evaluate the cost function inside the time integration loop, but this is cheap (see programme above).

7. RESULTS

In this section we present 2D and 3D results for inverse as well as optimization problems. We consider both inviscid and viscous flows in the transonic regime.

As we said, one of our goals is to achieve three-dimensional optimization for compressible flows on workstations. All these computations have been performed on a workstation making about 10 Mflops with about 100 MB of memory. 2D configurations require a few hours (always less than 3 h, even for the viscous turbulent case) and 3D cases have been computed through the night, showing that the cost of this approach is quite reasonable. Concerning memory requirements, owing to the optimization described in the previous section, the reverse mode requires approximately 10 times more memory than the direct solver. This is reasonable, as the direct solver is explicit and therefore does not require as much memory. All these computations have been performed in simple precision. An estimation of the memory required by the direct solver is given by $50 \times NN$ words (1 word = 32 or 64 bits depending on architectures), where *NN* is the number of nodes. This is more than is necessary in a structured solver, because the data structures involved are much more complicated in an unstructured approach.

In the following examples, when global constraints are present, the different penalty coefficients in the cost function are chosen for the different quantities involved to have variations of the same order of magnitude.

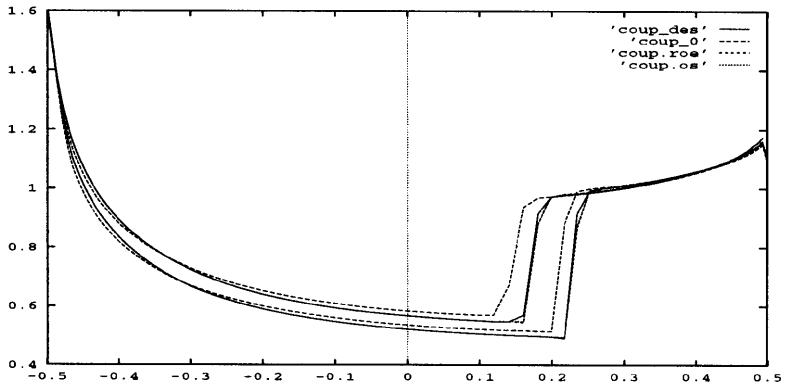

Figure 2. 2D inverse problem: pressure distribution. Roe and Osher flux-based gradients give the same results

7.1. An inverse problem in 2D (Figures 2–4)

This is a pressure recovery problem over the NACA 0014 aerofoil at inflow Mach number 0·85 and 1° of incidence. The inverse problem has been solved using the gradients based on the Roe and Osher fluxes with the same second-order construction. We can see that both approaches give the same results. The cost function is defined by

$$J(x) = \tfrac{1}{2}\int_x |p_x - p_{\text{target}}|^2 \mathrm{d}x, \tag{5}$$

where p_{target} is a given target pressure and p_x is the actual flow pressure. In this case the cost function has been reduced by two orders of magnitude in four iterations.

7.2. Drag reduction for an inviscid 2D flow (Figures 5– 9)

This is a drag reduction problem with constraint on the lift coefficient. The initial aerofoil is the NACA 0014 at Mach number 0·85 and 1° of incidence. As in the previous example, the gradients based on the Roe and Osher fluxes lead to the same results. The cost function in this case is

$$J(x) = \tfrac{1}{2}\int_x |p - p_0|^2 + 10C_{\text{d}} + |C_{\text{l}} - C_{\text{i}}^0|,$$

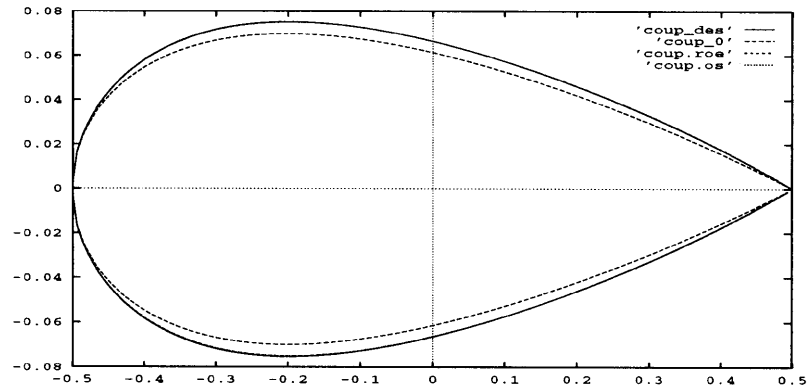

Figure 3. 2D inverse problem: initial, target and computed shapes

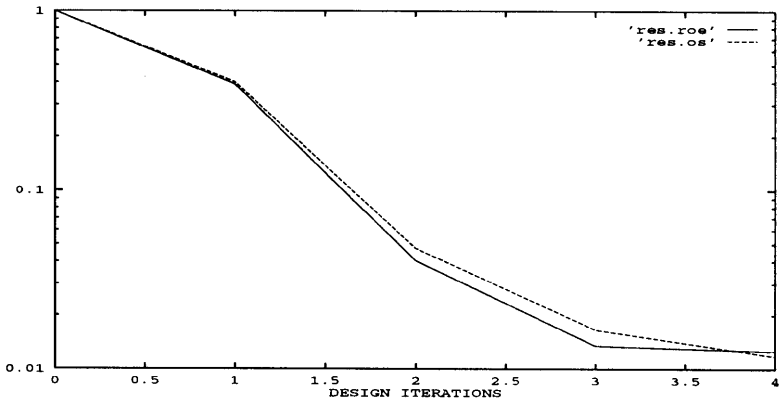

Figure 4. 2D inverse problem: convergence history for optimization procedure

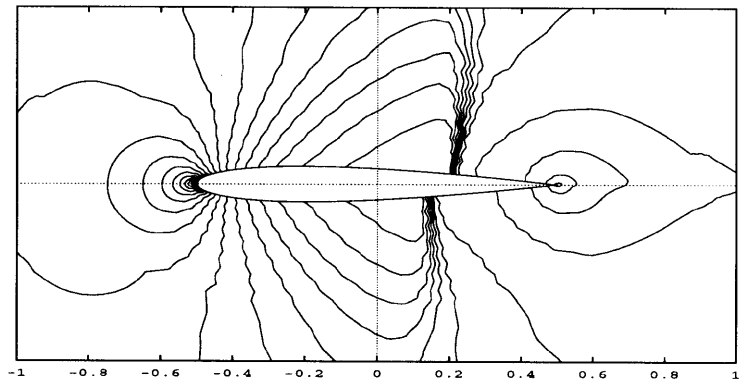

Figure 5. Inviscid 2D drag reduction: iso-Mach contours over initial shape

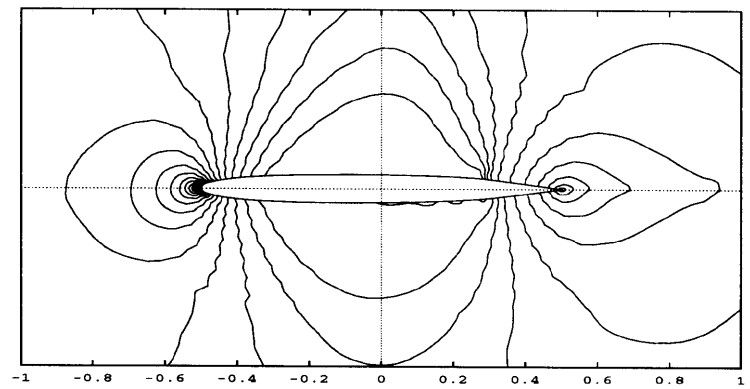

Figure 6. Inviscid 2D drag reduction: iso-Mach contours over optimized shape using Roe flux-based gradient

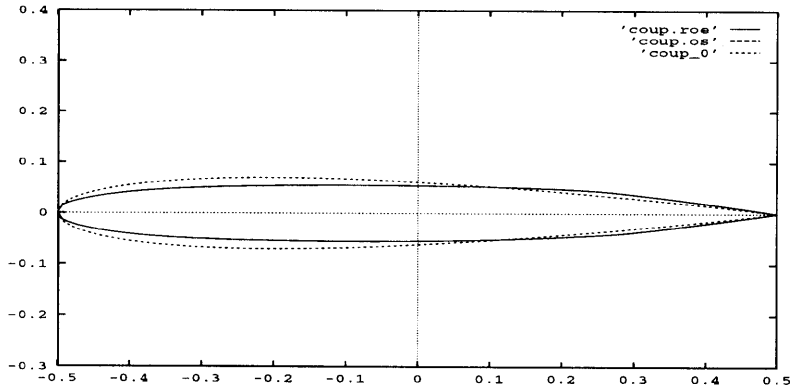

Figure 7. Inviscid 2D drag reduction: initial and optimized shapes obtained using Roe and Osher flux-based gradients

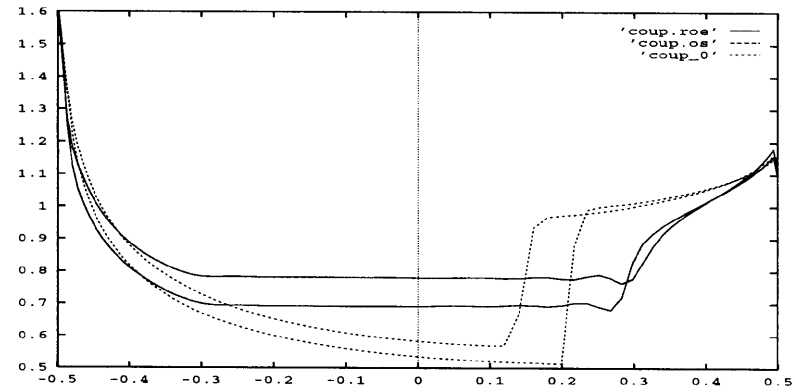

Figure 8. Inviscid 2D drag reduction: pressure distribution over initial and optimized shapes. The shapes obtained with Roe and Osher flux-based gradients are almost the same

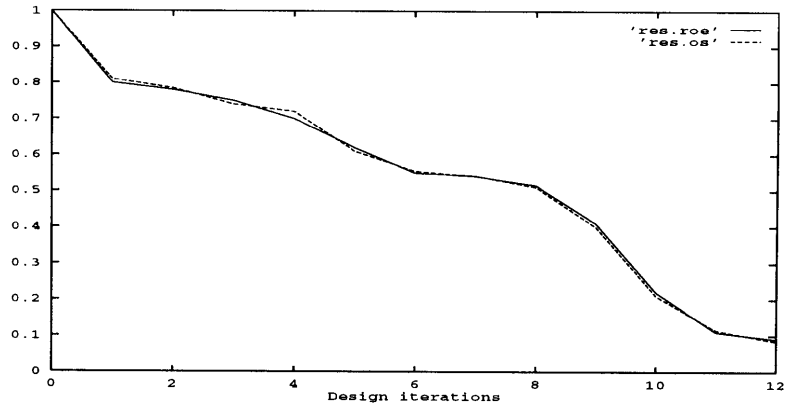

Figure 9. Inviscid 2D drag reduction: convergence history for optimization procedure with Roe and Osher flux-based gradients

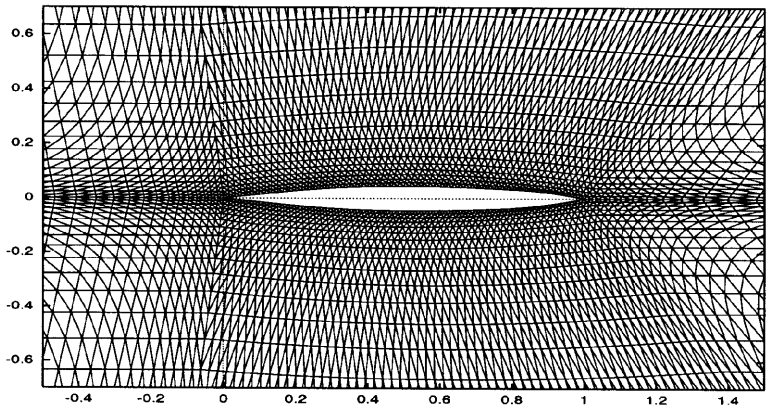

Figure 10. Turbulent versus inviscid 2D drag reduction: partial view of mesh (about 5000 nodes)

where C_1 and C_1^0 are the actual and initial lift coefficients. In this case we obtain an almost shock-free aerofoil. The cost function has been reduced by one order of magnitude in 12 iterations and the drag by more than one order. The lift has increased slightly.

7.3. Drag reduction for a 2D transonic turbulent flow (Figures 10–17)

This is a drag reduction problem with constraints on the volume and lift coefficient. The optimization has been done for an inflow Mach number of 0·85 at 1° of incidence. The initial aerofoil is a 5 per cent arc aerofoil. Both inviscid and viscous turbulent (at a Reynolds number of 10^7) configurations have been considered to show the impact of turbulence on the optimized shape. The cost function is given by

$$J(x) = \tfrac{1}{2}\int_x |p - p_0|^2 + 10C_d + |C_1 - C_1^0| + |V - V_0|,$$

where C_1 and C_1^0 (resp. V and V_0) are the actual and initial lift coefficients (resp. aerofoil volumes). Of course, for the viscous case the drag coefficient includes the viscous effect contributions.

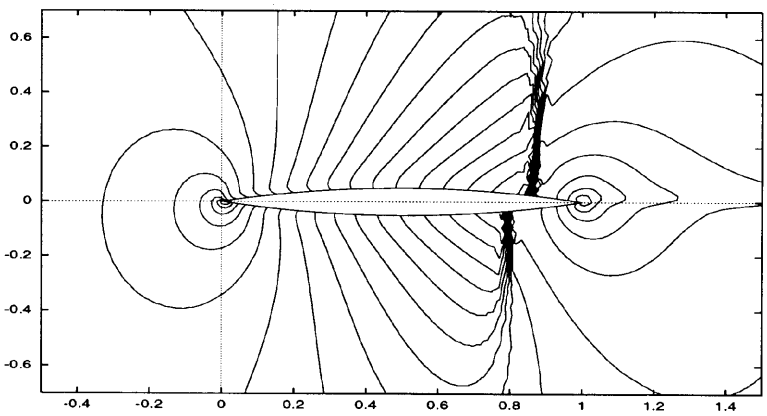

Figure 11. Turbulent versus inviscid 2D drag reduction: initial iso-Mach contours for Euler computation

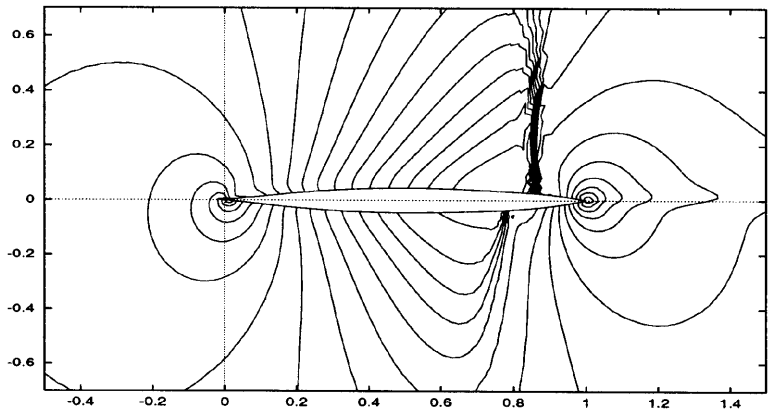

Figure 12. Turbulent versus inviscid 2D drag reduction: iso-Mach contours for Euler computation over optimized shape

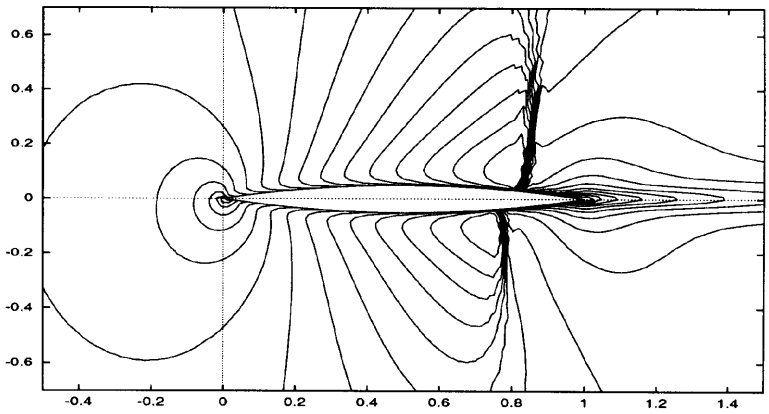

Figure 13. Turbulent versus inviscid 2D drag reduction: iso-Mach contours for turbulent computation

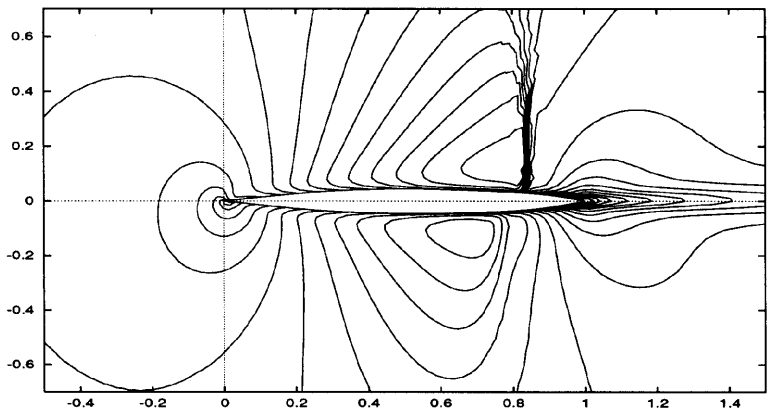

Figure 14. Turbulent versus inviscid 2D drag reduction: iso-Mach contours for turbulent computation over optimized shape

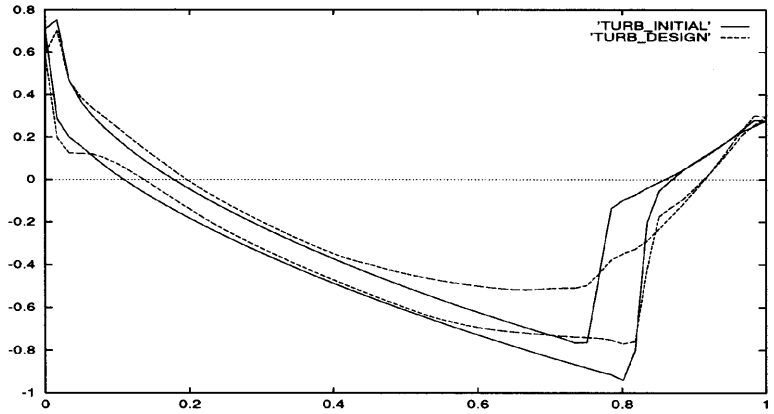

Figure 15. Turbulent versus inviscid 2D drag reduction: pressure distribution over initial and optimized shapes for viscous turbulent computation

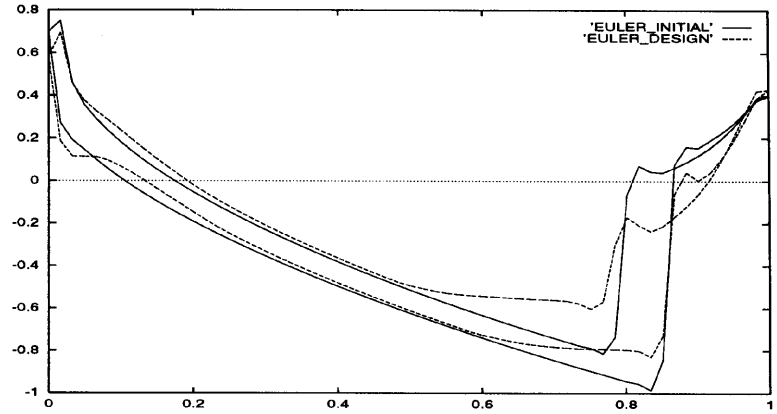

Figure 16. Turbulent versus inviscid 2D drag reduction: pressure distribution over initial and optimized shapes for inviscid computation

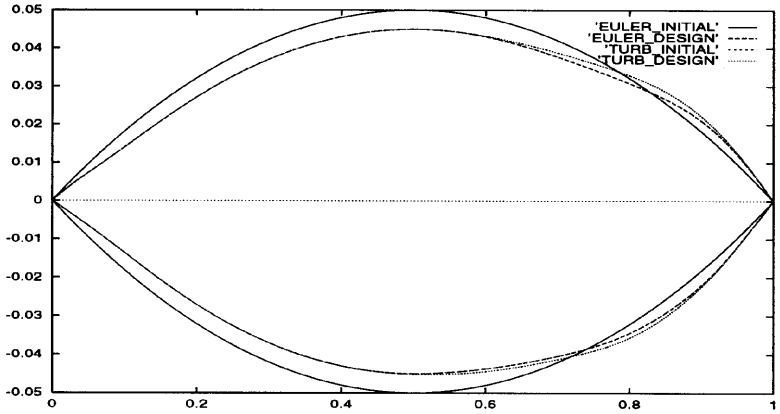

Figure 17. Turbulent versus inviscid 2D drag reduction: initial and optimized shapes for inviscid and viscous computations

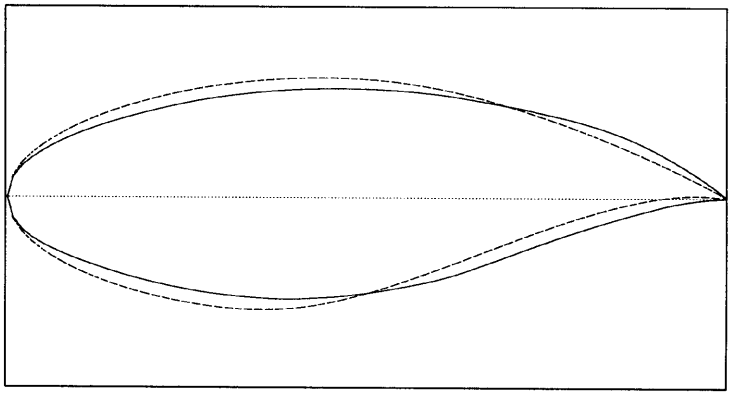

Figure 18. 3D drag reduction: initial and final shape cross-sections at span station $z = 0$

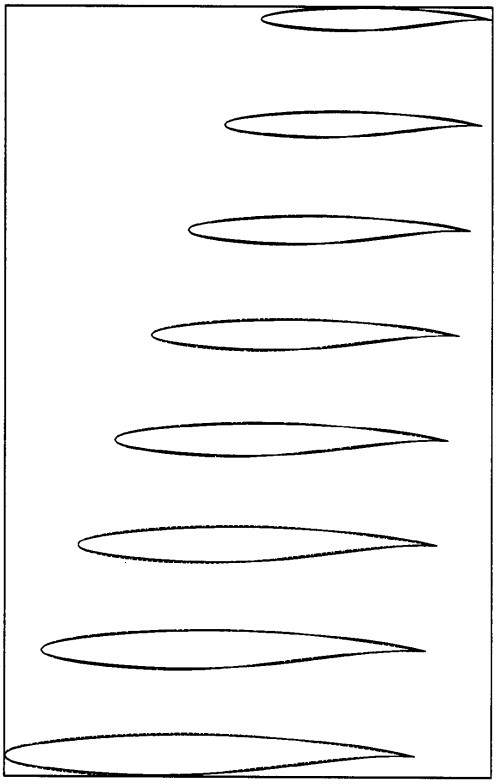

Figure 19. 3D drag reduction: initial and final shapes (cross-section). No by-section definition of the wing has been used for the optimization. There are about 3000 control points on the wing

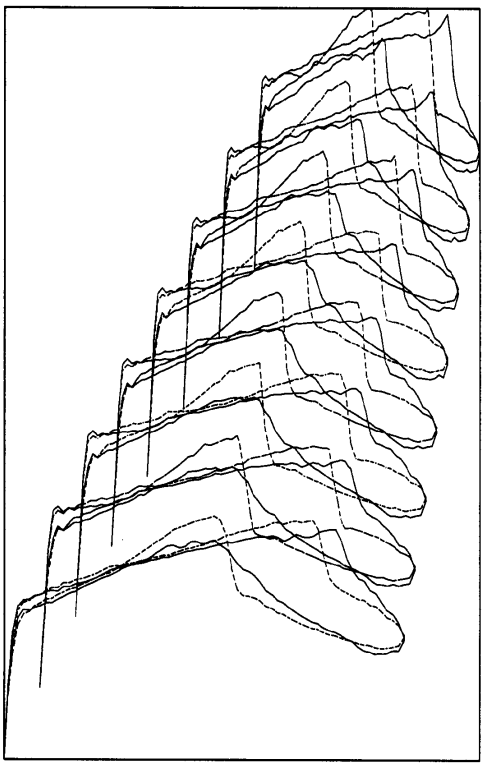

Figure 20. 3D drag reduction: initial and final pressure distributions (cross-section)

It is interesting to notice that owing to the smoothing effect of the viscosity, it is easier to obtain a shock-free profile for the turbulent flow configuration than for the corresponding inviscid case. Therefore, once the difficulties related to the introduction of new non-linear operators are removed, treating viscous cases becomes easier than treating inviscid ones. Of course, the CPU issue remains. In this case we used the same mesh for both inviscid and viscous computations. The mesh is therefore unnecessarily too fine for the inviscid computation. This might also explain why the optimization was easier for the viscous flow. In all cases the major difficulty comes from the evaluation of the Jacobian for the Navier–Stokes and k–ε systems, something which is quite painful to achieve without automatic differentiation.

7.4. 3D inviscid drag reduction (Figures 18–22)

This is a shock-induced drag reduction case in 3D on a swept wing with constraint on the wing volume. The RAE 2822 has been used to obtain a 3D wing after translation, zoom and rotation. The design takes place at an inflow Mach number of 0·85 and 1° of incidence. The mesh has about 3×10^5 tetrahedra and is unstructured. There are about 3000 control points on the wing. The by-section definition of the wing is not available and the presented cross-section results have been obtained by interpolation. The cost function is given by

$$J(x) = C_d + |V - V_0|.$$

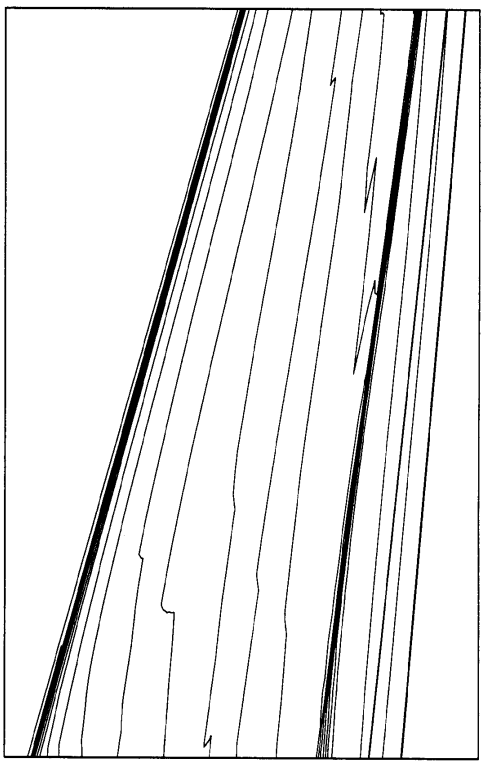

Figure 21. 3D drag reduction: iso-Mach contours over upper surface of initial wing

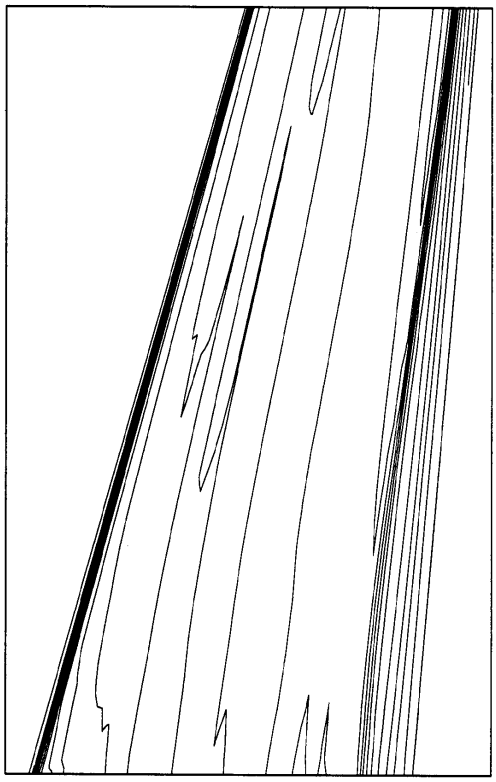

Figure 22. 3D drag reduction: iso-Mach contours over upper surface of optimized wing

We can see that two sharp shocks are present on the upper and lower surfaces. In the optimized shape the lower surface is almost shock-free and the shock on the upper surface has been quite smoothed. The drag has been reduced by about 50 per cent (from $9{\cdot}8 \times 10^{-2}$ to 5×10^{-2}), while the volume remained almost unchanged (from 0·28 to 0·275) and the lift increased (from 0·48 to 0·55).

8. CONCLUDING REMARKS

Results of optimal shape design for two- and three-dimensional configurations of inviscid and turbulent flows have been presented to feature a new shape optimization approach. The main ingredients of this technique are the reverse mode of automatic differentiation and a CAD-free framework making it possible to consider all the wall nodes in an unstructured or structured 2D or 3D mesh as control points. Preliminary examples show the ability of the method to treat inverse and control problems involving up to several thousand control parameters. The reverse mode is therefore a powerful tool for providing the gradient of the discrete operators involved.

It is important to notice that all the iterative schemes used are explicit. A parallel implementation of this approach is therefore quite simple, except for the gradient computation by reverse mode. However, this is shown to be cheap. Future work will include more extensive validation of these techniques in 3D configurations. Another aspect of our current research concerns the introduction of mesh adaption in the optimization procedure. To this end we have to take into account connectivity changes in the mesh.

ACKNOWLEDGEMENTS

The author would like to thank Professors M. Hafez, P. Le Tallec and O. Pironneau for their support and helful discussions. The author would also like to thank N. Rostaing-Schmidt and C. Faure of INRIA Sophia Antipolis for their assistance in using Odyssée.

APPENDIX I: FLOW SOLVER

We describe a few ingredients of our flow solver. Consider the following form of the Navier–Stokes and k–ε equations:

$$\frac{\partial U}{\partial t} + \nabla \cdot [F(U) - N(U)] = S(U), \tag{6}$$

where $U = (\rho, \rho u, \rho v, \rho E, \rho k, \rho\varepsilon)^{\mathrm{T}}$ is the vector of conservation variables, F and N are the convective and diffusive operators and $S(U) = (0, 0, 0, 0, S_k, S_\varepsilon)^{\mathrm{T}}$.

The k–ε model[4] we use is rather classical and is an extension to compressible flows of its incompressible version.[1,20] The right-hand sides S_k and S_ε contain the production and destruction terms for ρk and $\rho\varepsilon$:

$$S_k = c_\mu \rho \frac{k^2}{\varepsilon} P - \tfrac{2}{3}\rho k \nabla \cdot u - \rho\varepsilon, \tag{7}$$

$$S_\varepsilon = c_1 \rho k P - \frac{2c_1}{3c_\mu}\rho\varepsilon \nabla \cdot u - c_2 \rho \frac{\varepsilon^2}{k}. \tag{8}$$

The constants c_μ, c_1, c_2, c_ε of this model are respectively 0·09, 0·1296, 11/6, 1/1·4245 and $P = S : \nabla u$. The constants c_2 and c_ε are different from their original values of 1·92 and 1/1·3. A justification for these values can be found in References 5, 6 and 21.

Discretization

Let $\Omega_h = \cup_j T_j$ be a discretization by triangles of the computational domain Ω and let $\Omega_h = \cup_i C_i$ be its partition into cells (Figure 23). Thus we can associate with each $w_h \in V_h$, where V_h is the set of continuous affine functions on our triangulation, a w_h' piecewise constant function on cells by

$$w_h'|C_i = \frac{1}{|C_i|}\int_{C_i} w_h.$$

Conversely, knowing w_h' piecewise constant, w_h is obtained as $w_h(S_i) = w_h'|C_i$.

The weak formulation of (6) is: find $U_h \in (V_h)^4$ such that, $\forall\, \phi_h \in V_h$,

$$\int_\Omega \frac{\partial U_h}{\partial t}\Phi_h - \int_\Omega (F_h - N_h)(U_h)\nabla(\phi_h) + \int_{\partial\Omega}(F_h - N_h)\cdot n\phi_h = 0. \tag{9}$$

This is equivalent to the following weak formulation obtained by taking in the convective part of (9) for ϕ_h the characteristic function of C_i and by using an explicit time integration:

$$|C_i|\frac{U_i^{n+1} - U_i^n}{\Delta t} + \int_{\partial C_i} F_d(U^n)\cdot n = \text{RHS}. \tag{10}$$

We use a centred scheme to compute the right-hand side:

$$\text{RHS} = -\int_{\Omega_h} N(U^n)\nabla(\phi_h) + \int_{\partial\Omega} N(U^n)\cdot n\phi_h.$$

Moreover, $F_d(U_h^n) = F(U_{\partial\Omega})$ on $\partial C_i \cap \partial\Omega$ and elsewhere F_d is a piecewise constant upwinded approximation of $F(U)$ satisfying

$$\int_{\partial C_i} F_d \cdot n = \sum_{j\neq i} \Phi(U'|_{C_i}, U'|_{C_j})\int_{\partial C_i \cap C_j} n. \tag{11}$$

After writing $\tilde{B}$ for the Jacobian of F at Roe's mean values, we take for Φ the Roe flux[9]

$$\Phi_{\text{Roe}}(u, v) = \tfrac{1}{2}[F(u) + F(v)] - |\tilde{B}|\frac{v-u}{2}.$$

Spatial second-order accuracy is obtained by using a MUSCL-like extension involving a combination of upwind and centred gradients. More precisely, let ∇U_i be an approximation of the gradient of U at node i. We define the following quantities on the segment $[i, j]$:

$$U_{ij} = U_i + 0{\cdot}5\text{Lim}(\beta(\nabla U)_i \overrightarrow{ij}, (1-\beta)(U_i - U_j)),$$
$$U_{ji} = U_j - 0{\cdot}5\text{Lim}(\beta(\nabla U)_j \overrightarrow{ij}, (1-\beta)(U_j - U_i)),$$

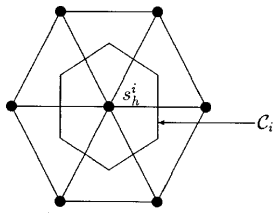

Figure 23. Discretization geometry

where Lim is a Van Albada-type limiter[10] given by

$$\mathrm{Lim}(a, b) = 0{\cdot}5[1 + \mathrm{sgn}(ab)]\frac{(a^2 + \alpha)b + (b^2 + \alpha)a}{a^2 + b^2 + 2\alpha},$$

with $0 < \alpha \ll 1$, and β is a positive constant containing the amount of upwinding, $\beta \in [0, 1]$ (here $\beta = \frac{2}{3}$. Now the second-order accuracy in space is obtained by replacing U_i' and U_j' in (11) by U_{ij} and U_{ji}. These techniques have been successfully extended to unstructured meshes in the past.[7]

Applying this approach to the k–ε equations will not guarantee the positivity of ρk and $\rho\varepsilon$. Therefore the convective fluxes for the turbulent equations are computed using the PSI fluctuation-splitting scheme[22] which is positive and linearity-preserving.

The boundary and initial conditions are classical. In particular, a Steger–Warming[11] flux-splitting scheme is used for inflow and outflow boundaries.

Implementation of wall laws

In weak form (finite element or finite volume approach) the following boundary integrals appear in the momentum and energy equations in the case of adiabatic walls ($\vec{t}$, $\vec{n}$) denotes the local orthogonal basis for a wall node):

$$\int_{\Gamma_w} (\mathbf{S} \cdot \vec{n})\mathrm{d}\sigma, \qquad \int_{\Gamma_w} (\vec{u}\mathbf{S})\vec{n}\mathrm{d}\sigma,$$

where $\mathbf{S} = (\mu + \mu_t)(\nabla u + \nabla u^{\mathrm{T}} - \frac{2}{3}\nabla \cdot uI)$ is the Newtonian strain tensor. We decompose $\mathbf{S} \cdot \vec{n}$ over ($\vec{t}$, $\vec{n}$):

$$\mathbf{S} \cdot \vec{n} = (\mathbf{S} \cdot \vec{n} \cdot \vec{n})\vec{n} + (\mathbf{S} \cdot \vec{n} \cdot \vec{t}) \cdot \vec{t}. \tag{12}$$

In our implementation the first term (S_{nn}) on the right-hand side of (12) is computed explicitly and the following wall laws are used:

$$\vec{u} \cdot \vec{n} = 0, \qquad (\mathbf{S}\vec{n} \cdot \vec{t})\vec{t} = -\rho u_\tau^2 \vec{t}, \qquad \vec{u}\mathbf{S}\vec{n} = -\rho u_\tau^2 \vec{u} \cdot \vec{t},$$

where u_τ is the friction velocity, the solution of $\vec{u} \cdot \vec{t} = u_\tau f(u_\tau)$. We decompose $f(u_\tau)$ into two parts:

$$f(u_\tau) = f_r(u_\tau) + f_c(u_\tau),$$

with $f_r(u_\tau)$ the non-linear Reichardt equation

$$f_r(y^+) = 2{\cdot}5\log(1 + \kappa y^+) + 7{\cdot}8\left(1 - \mathrm{e}^{-y^+/11} - \frac{y^+}{11}\mathrm{e}^{-0{\cdot}33y^+}\right),$$

where $y^+ = \rho u_\tau y/\mu$. The f_c contribution exists with pressure and convection effects:

$$f_c(y^+) = \frac{35C\mu}{\kappa\rho^2 u_\tau^3}\log\left(1 + \kappa\frac{(y^+)^2}{70}\right) \quad \text{if } y^+ \leqslant 5{\cdot}26, \tag{13}$$

$$f_c(y^+) = \frac{C\delta}{\kappa\rho u_\tau^2} \quad \text{if } y^+ \geq 5{\cdot}26, \tag{14}$$

where C is explicitly computed as

$$C = \frac{\partial p}{\partial x} + \frac{\partial \rho u^2}{\partial x} + \frac{\partial \rho uv}{\partial y}.$$

Once, u_τ is computed, k and ε are set to

$$k = \frac{u_\tau^2}{\sqrt{c_\mu}}\alpha, \qquad \varepsilon = \frac{u_\tau^3}{\kappa\delta}\min\left(1, \alpha + \frac{0{\cdot}2\kappa(1-\alpha)^2}{\sqrt{c_\mu}}\right),$$

where δ is the distance of the fictitious computational domain from the solid wall and $\alpha = \min(1, y^+/10)$ reproduces the behaviour of k when δ tends to zero. The distance δ is given *a priori* and is kept constant during the computation.

APPENDIX II: AD, A SIMPLE EXAMPLE

We give a simple example of automatic differentiation in reverse mode. Details of these techniques can be found in References 14–16. Consider the following Fortran 77 programme:

```
u1 = x
u2 = x**2 + 2*u1**2
f = u1 + u2
```

The aim is to compute $\mathrm{d}f/\mathrm{d}x$. In automatic differentiation in reverse mode we consider the lines of the programme as constraints and associate with each of them a Lagrange multiplier and define an augmented Lagrangian as follows:

$$L = u_1 + u_2 + p_1(u_2 - x^2 - 2u_1^2) + p_2(u_1 - x).$$

We know that at the solution we have

$$\frac{\partial L}{\partial u_1} = 1 - 2p_1u_1 + p_2 = 0, \qquad \frac{\partial L}{\partial u_2} = 1 + p_1 = 0.$$

We notice that to find p_i, we have to solve the previous set of equations in 'reverse' order. Once p_i are known, we have

$$\frac{\partial L}{\partial x} = \frac{\mathrm{d}f}{\mathrm{d}x} = -2p_1x - p_2,$$

which is the Jacobian of f.

DO-IF

Consider the evolution of $|u(t)|$ (non-differentiable) with respect to the initial condition u_0. u is the solution of

$$\frac{\mathrm{d}u}{\mathrm{d}t} = -au, \qquad u(0) = u_0.$$

We use an explicit discretization

$$\frac{u^{i+1} - u^i}{\Delta t} = -au^i$$

which can be programmed as

$$\begin{aligned} & u = u_0, \\ \text{do} \quad & i = 1, \ldots, N, \\ & v = -au, \\ & u = u + \Delta t v, \\ & \text{end do} \\ & f = |u|. \end{aligned}$$

After expansion via

$$\begin{aligned} & u_1 = u_0, \qquad v_1 = -au_1, \qquad u_2 = u_1 + \Delta t v_1, \\ & v_2 = -au_2, \ldots, v_N = -au_{N-1}, \qquad u_{N+1} = u_N + \Delta t v_N, \end{aligned}$$

we introduce the Lagrangian of the programme as before:

$$L = |u_{N+1}| + p_0(u_1 - u_0) + \sum_{i=1}^{N} [p_i(v_i + au_i) + p_i'(u_{i+1} - u_i + \Delta t v_i)].$$

Optimality conditions give

$$\frac{\partial L}{\partial u_0} = \frac{\partial f}{\partial u_0} = -p_0, \qquad \frac{\partial L}{\partial u_1} = p_0 + p_1 a - p_1',$$

$$\frac{\partial L}{\partial v_i} = p_i + p_i' \Delta t, \quad i = 1, \ldots, N, \qquad \frac{\partial L}{\partial u_i} = p_i a - p_i', \quad i = 1, \ldots, N,$$

$$\text{if} \quad (u < 0) \quad \frac{\partial L}{\partial u_{N+1}} = -1 + p_N', \qquad \text{if} \quad (u \geq 0) \quad \frac{\partial L}{\partial u_{N+1}} = 1 + p_N'.$$

The limit of the method is the memory required to store p_i and p_i', especially if internal loops are present. We can see that the branches of conditional statements are treated separately and that the results are assembled after derivation.

Limitations

There are a few limitations when using the reverse mode of Odyssée. The most important is that GOTO instructions should not be used (neither RETURN nor ENTRY). This is logical, as in the reverse mode we have to follow the graph of the programme in reverse order and using a GOTO makes the graph much more complicated.

REFERENCES

1. O. Pironneau, *Optimal Shape Design for Elliptic Systems*, Springer, New York, 1984.
2. A. Jameson, 'Optimum aerodynamic design via boundary control', *AGARD Rep. 803*, Von Karman Institute Courses, 1994.
3. M. Hafez, B. Mohammadi and O. Pironneau, 'Optimum shape design using automatic differentiation in reverse mode', *Proc. ICNMFD*, Monterey, CA, 1996.
4. B. E. Launder and D. B. Spalding, *Mathematical Models of Turbulence*, Academic, New York, 1972.
5. B. Mohammadi and O. Pironneau, *Analysis of the K–Epsilon Turbulence Model*, Wiley, New York, 1994.
6. B. Mohammadi and O. Pironneau, 'Unsteady separated turbulent flows computations with wall-laws and k–ε model', *Comput. Methods Appl. Mech. Eng.*, submitted.
7. A. Dervieux, 'Steady Euler simulations using unstructured meshes', *VKI Lecture Ser. 1884-04*, 1985.
8. B. Mohammadi, 'CFD with NSC2KE: a user guide', *INRIA Tech. Rep. 164*, 1994.

9. P. L. Roe, 'Approximate Riemann solvers, parameters vectors and difference schemes', *J. Comput. Phys.*, **43**, (1981).
10. G. D. Van Albada and B. Van Leer, 'Flux vector splitting and Runge– Kutta methods for the Euler equations', *ICASE 84-27*, 1984.
11. J. Steger and R. F. Warming, 'Flux vector splitting for the inviscid gas dynamic with applications to finite-difference methods', *J. Comput. Phys.*, **40**, 263–293 (1983).
12. S. Osher and S. Chakravarthy, 'Upwind difference schemes for the hyperbolic systems of conservation laws', *Math. Comput.*, (1982).
13. J. M. Malé, B. Mohammadi and N. Rostaing-Schmidt, 'Direct and reverse modes of automatic differentiation of programs for inverse problems: application to optimum shapes design', *Proc. 2nd Int. Workshop on Computational Differentiation*, Santa Fé, NM, 1996, SIAM, Philadelphia, PA, 199x.
14. J. C. Gilbert, G. Le Vey and J. Masse, 'La différentiation automatique de fonctions représentées par des programmes', *INRIA Res. Rep. 1557*, 1991.
15. N. Rostaing-Schmidt, 'Différentiation automatique: application à un probléme d'optimisation en météorologie', *Ph.D. Thesis*, University of Nice, 1993.
16. C. Faure, 'Splitting of algebraic expressions for automatic differentiation', *Proc. 2nd Int. Workshop on Computational Differentiation*, Santa Fé, NM, 1996, SIAM, Philadelphia, PA, 199x.
17. B. Mohammadi and O. Pironneau, 'New progress in optimum shapes design', in *CFD Review 95*, Wiley, New York, 1996.
18. B. Mohammadi, *Automatic Differentiation and Nonlinear PDE*, ECCOMAS, Paris, 1996.
19. B. Mohammadi, 'Différentiation automatique par programme et optimisation de formes aérodynamiques', *MATAPLI*, 1996.
20. D. Vandromme, 'Contribution à la modélisation et la prédiction d'écoulements turbulents à masse volumique variable', *Ph.D. Thesis*, University of Lille, 1983.
21. F. Hecht and B. Mohammadi, 'Mesh adaption by metric control for multi-scale phenomena and turbulence', *Proc. 35th AIAA Conf.*, Reno, NV, 1997, AIAA, New York, 199x.
22. R. Struijs, H. Deconinck, P. de Palma, P. Roe and G. G. Powel, 'Progress on multidimensional upwind Euler solvers for unstructured grids', *AIAA Paper 91-1550*, 1991.

