Chapter 5
Optimization Platform and Implementation Issues

Contents

1 Introduction

In this chapter we briefly describe other ingredients needed to build a design and control platform. These are:

- The definition of a control space (shape parameterization).
- Operators acting on elements of the control space (shape deformation tool).
- Operators to link the control space and the geometrical entities (mesh deformation tool).
- Minimization algorithms based on pseudo-unsteady systems formulation,
- Mesh adaptation.

For each point, we will only describe the approach we actually use. Therefore the description does not intend to be exhaustive. The idea is to show the kind of results that can be obtained with a platform build with these tools. We are interested in particular by gradient based minimization algorithms which have a pseudo-unsteady interpretation.

2 Shape parameterization and shape and mesh deformation tools

We describe the set of tools needed to define the shape and mesh deformation from a variation of control parameters ($x(1..n_c)$):

$$\delta x \rightarrow \delta x_w \rightarrow \delta x_m,$$

where $x_w(1..n_w)$ denotes the set of discretization points on the geometry and $x_m(1..N)$ the internal mesh nodes.

For a given parameterized curve $\gamma(t)$ in $\mathbf{R}^2$ of parameter t, a natural way to specify deformation, is to relate these deformations to the local normal $n(t)$ of $\gamma(t)$. Hence, the deformed curve $\tilde{\gamma}$

$$\tilde{\gamma}(\alpha, t) = \gamma(t) + f(\alpha, t).n(\gamma(t)).$$

$\alpha \in P \subset \mathbf{R}^{n_c}$ describes the control space P which can be small or large depending to the kind of parameterization chosen. In general three level of parameterizations are possible: CAD-based, in a convex hull of a set of shapes, CAD-free.

2.1 CAD-based

In this parameterization the shape and its deformations can be defined through a few control points compared to the number of nodes needed for the discretization of the shape for a simulation, with finite elements for instance:

$$n_c << n_w.$$

Of course, shape deformations should be admissible for the parameterization. For instance, a cubic spline will not allow a singularity in the shape. Another feature of this parameterization is to smooth the variations of control points when propagating to body discretization points.

2.2 Basis of shape functions

We can express the deformations as a linear combination of admissible deformations available in a data base. This is probably the most convenient way but it is specific of each application and requires a rich data base.

$$S = \Sigma \lambda_i S_i, \quad \Sigma \lambda_i = 1, \quad S \in \{S_i, i = 1, .., n_c\}.$$

2.3 CAD-Free or 'h'

In this parameterization, we choose to use the same parameterization for the simulation and optimization. This is the easiest way to start:

$$n_c = n_w.$$

Hence, the geometrical modeler is removed from the optimization loop. The geometrical modeler is usually quite complicated to differentiate and the source code is not available to the user when a commercial tool is used. Moreover, the CAD parameterization is not necessarily the best for optimization. The idea therefore is to perform the optimization at the 'h' (discretization) level, i.e. on the discrete problem directly. Of course, we have correspondence between the surface mesh and the CAD parameterization. Hence, in this parameterization the only entity known during optimization is the mesh.

2.3.1 A local second order smoother

This is an attempt to recover the smoothing feature of the CAD-based parameterization when using a CAD-Free parameterization. The need for a smoothing operator can find a theoretical justification from the consistent approximation theory of Polak and Pironneau given in Chapter 6.

The importance of a smoothing step can also be understood by the following argument:

If Γ denotes a manifold of dimension $(n-1)$ in a domain $\Omega \in R^n$, we want the variation $\delta x_w \in C^1(\Gamma)$. From Sobolev inclusions, we know that $H^{(2n-1)/2}(\Gamma) \subset C^1(\Gamma)$. It is easy to understand that the gradient method we use do not produce necessarily $C^1(\Gamma)$ variations δx_w, but only $L^2(\Gamma)$ and therefore we need to project them into $H^{(2n-1)/2}(\Gamma)$ for instance (an example of this is given in picture (1)).

The fact that the gradient has necessarily less regularity than the parameterization is also easy to understand. Suppose that the cost function is a quadratic function of the parameterization: $J(x) = (Ax - b)^2$ with $x \in H^1(\Gamma)$, $A \in H^{-1}(\Gamma)$ and $b \in L^2(\Gamma)$. The gradient $J'_x = (2(Ax - b)A) \in V$ with $H^{-1}(\Gamma) \subset V \subset L^2(\Gamma)$. Again, any parameterization variation using J'_x as descent direction will have less regularity than x: $\delta x = -\rho J'_x = -\rho(2(Ax - b)A) \in V$, where $H^{-1}(\Omega) \subset V \subset L^2(\Omega)$. We need therefore to project (engineer would say smooth) into $H^1(\Omega)$.

One way is to choose the projected variations $(\delta \tilde{x}_w)$ to be the solution of an elliptic system of degree $(2n - 1)$. However, as we are using a P^1 discretization, a second order elliptic system is sufficient even in 3D if we suppose the edges of the geometry (where the geometry is not initially C^1, for instance a trailing edge) as being constrained for the design. This means that they remain unchanged. Therefore we project the variations (δx_w) only into $H^2(\Gamma)$ even in 3D.

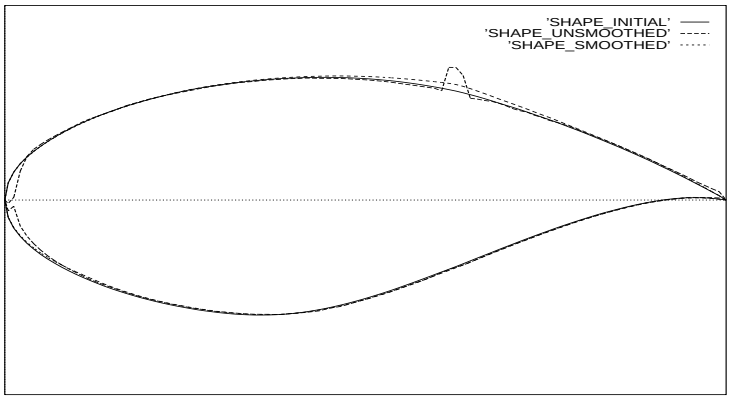

Figure 1: *Smoothed and non-smoothed shapes. We can see that the gradient jumps through shocks and also produces a non-smooth shape in leading edge regions. This is the result of the first iteration of the optimization. If we keep continue the shape becomes more and more irregular.*

Hence, to avoid oscillations, we define the following 'local' smoothing operator over the shape:

$$(I - \varepsilon\Delta)\delta\tilde{x}_w = \delta x_w, \tag{1}$$
$$\delta\tilde{x}_w = \delta x_w = 0 \quad \text{on wedges},$$

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. By 'local' we mean that if the predicted shape is locally smooth, it remains unchanged during this step. The regions where the smoothing is applied are identified using a discontinuity-capturing operator. Furthermore the linear system is solved iteratively and ε is set to zero during the Jacobi loops if

$$\frac{\delta_{ij}(\delta x_w)}{(\delta x_w)_T} < 10^{-3}, \tag{2}$$

where $\delta_{ij}(\delta x_w)$ is the difference between the variations of the two nodes of each segments of a surface triangle (nodes of a boundary edge in 2D) and $(\delta x_w)_T$ the mean variation on this triangle (edge in 2D).

This operation can also be seen as a modification of the scalar product for the Hilbert space in which optimization is performed and might therefore also have a preconditionning effect in the sense that it propagates localized high frequency informations where they are not seen initially. Hence, the impact of the smoothing in a descent algorithm (see below for general pseudo-unsteady systems for x):

$$J^{n+1} \leq J^n + (J'^n_x, \delta x)_0, \quad \delta x = -\rho J'^n_x,$$
$$J^{n+1} \leq J^n - \rho(J'^n_x, J'^n_x)_0 \leq J^n,$$

is located in the scalar product M (based on the projection operator for the gradient into the parameterization space) such that:

$$J^{n+1} \leq J^n - \rho(J'^n_x, J'^n_x)_M \leq J^n - \rho(J'^n_x, J'^n_x)_0.$$

3 Handing Domain Deformations

Once $(\delta\tilde{x}_w)$ known, we have to spread these variations overall the mesh. We give here a few algorithms to perform this task. We ask the deformation method to preserve the positivity and orientation of the elements. In other words, the final mesh has to be conformal. Lets call D the deformation operator applied to the shape deformation (δx_w). We need the mesh deformation δx_m to verify $\delta x_m = \delta\tilde{x_w}$ on the shape. $\delta\tilde{x_w}$ being worth δx_w on the shape and 0 elsewhere. This relation is therefore a boundary condition for D.

3.1 Explicit deformation

This is a simple way to explicitly prescribe the deformation to apply to each mesh node, knowing the shape deformation. The idea is to make the deformation for a node proportional to its distance to the shape. In other words, for an internal node i, we have:

$$\begin{array}{l} (\delta x_m)_i = \frac{1}{\alpha_i} \sum_{k\in\Gamma_w} w_k \alpha_{ki} \, (\delta\tilde{x})_i \\ \delta\tilde{x_m} = \delta\tilde{x} \text{ on } \Gamma_w \end{array} \tag{3}$$

where,

- δx_m is the variation of the mesh nodes,
- w_k is a weight for the contribution of each of the node k of the shape,
- $\alpha_{ki} = \frac{1}{|\tilde{x}_k - \tilde{x}_i|^\beta}$ with β a positive arbitrary parameter,
- $\alpha_i = \sum_{k\in\Gamma_w} w_k \alpha_{ki}$ is the normalization parameter.

In 2D, w_k is equal to the sum of half of each segments size sharing the node k and in 3D equal to 1/3 of the surface triangle sharing the node k. This algorithm has been widely used for the propagation of shape deformation [17, 20, 18].

A justification for this algorithm can be found from the the following integral:

$$\frac{1}{\int_{\Gamma_w} \frac{d\gamma}{|x_m - x_w|^\beta}} \int_{\Gamma_w} \frac{\delta\,(x_w)}{|x_m - x_w|^\beta} \, d\gamma,$$

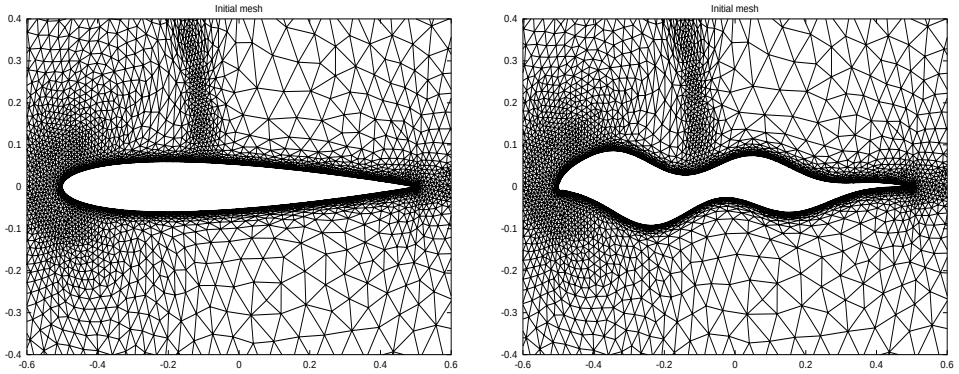

Figure 2: *Example of mesh deformation with the explicit deformation law (left: initial, right: deformed).*

where $\int_{\Gamma_w} \frac{\delta(x_w)}{|x_m - x_w|^\beta}\, d\gamma$ is the convolution product of the Green function $\frac{1}{|x_m|^\beta}$ with the Dirac function δ at point x_w of the shape Γ_w. The biggest is β, the more the propagation is localized around the shape. For practical applications, $\beta = 4$ seems to be a good choice. This choice comes from experience and does not has a theoretical justification.

This algorithm is quite robust but expensive. Indeed , the complexity of the method is proportional to the number of the shape discretization nodes times the number of mesh nodes. This is therefore too expensive to be used in 3D everywhere in the mesh. We usually use this approach close to the shape where the mesh is fine and continue the propagation with iterative methods based on the solution of elliptic systems.

3.2 Adding an elliptic system

Another possible choice is to solve an elliptic equation of the form of (1). Here, the different components are not coupled together.

$$(I - \varepsilon\Delta)\delta x_m = \overline{\delta x_m}, \tag{4}$$

$$\delta x_m = 0 \quad \text{on inflow and outflow boundaries,}$$

$$\frac{\partial \delta x_m}{\partial n} = 0 \quad \text{on slipping boundaries,}$$

$$\delta x_m = \overline{\delta x_m} \quad \text{on wall nodes,}$$

where $\overline{\delta x_m}$ is the extension of δx_w over the mesh defined by

$$\overline{\delta x_m} = \delta x_w \quad \text{for mesh nodes on the wall,}$$

$$\overline{\delta x_m} = 0 \quad \text{for internal nodes.}$$

None of the previous algorithms guarantee the positivity of the elements. To make it so, we need to introduce constraints on the deformations of the mesh nodes or for instance to make the viscosity ε of the previous elliptic system proportional to the inverse of the local mesh size. This is important to avoid mesh degeneration. But, the principle remains the same. We usually combine these two algorithms for close and far mesh fields.

3.3 Injection boundary condition

This is a classical way to account for small shape deformations without performing mesh deformations, by deriving a more complex boundary condition called an equivalent injection boundary condition , to be applied on the original shape. Denote, $\vec{x_w^1}$ the original shape, $\vec{x_w^2}$ the shape after deformation, $\vec{u}$ being the flow velocity and $\vec{n}$ the unit normal to the shape, the slipping boundary condition can be expressed either in a fixed frame or in a one attached and moving with the shape.

Denote by $\vec{n_1}$ and $\vec{n_2}$ the unit normal on each shape (in the fixed and moving frames). In frame 2, the slipping boundary condition reads: $\vec{u_2}.\vec{n_2} = 0$ and in frame 1:

$$\vec{u_2}.\vec{n_2} = \vec{V}.\vec{n_2} = \frac{\vec{x_w^2} - \vec{x_w^1}}{\delta t}.\vec{n_2}.$$

$\vec{V}$ is therefore the speed of the shape in frame 1. Now, if we suppose that the variations of the geometrical quantities dominate the physical ones (i.e. $|\delta x| \sim |\delta \vec{n}| >> |\delta u|$):

$$\vec{V}.\vec{n_2} = \vec{u_2}.\vec{n_2} \sim \vec{u_1}.\vec{n_1} + \vec{u_1}.(\vec{n_2} - \vec{n_1}).$$

This defines an implicit relation for $\vec{u_1}.\vec{n_1}$ which is a new injection boundary condition for u in frame 1 relating the state at time $t + \delta t$ (time step $n + 1$) of the computation to time t (time step n):

$$\vec{u_1}(t + \delta t).\vec{n_1} = -\vec{u_1}(t).(\vec{n_2}(t + \delta t) - \vec{n_1}) + \vec{V}(t + \delta t).\vec{n_2}(t + \delta t).$$

In the same way, an equivalent boundary condition can be derived for the tangential velocity : $\vec{u_2}.\vec{\tau_2} = 0$ in frame 2 and $\vec{u_2}.\vec{\tau_2} = \vec{V}.\vec{\tau_2}$ in the fixed frame. A similar argument as above leads to :

$$\vec{u_1}(t + \delta t).\vec{\tau_1} = -\vec{u_1}(t).(\vec{\tau_2}(t + \delta t) - \vec{\tau_1}) + \vec{V}(t + \delta t).\vec{\tau_2}(t + \delta t).$$

In these expressions, $\vec{\tau_1}, \vec{\tau_2}$ denote the local unit tangent vectors build from $n_1, \vec{n_2}$. These tangent vectors are taken parallel to the local velocity in 3D.

With wall-laws, the above expressions become $\vec{u_2}.\vec{\tau_2} = u_\tau f(y^+)$ in frame 2 and $\vec{u_2}.\vec{\tau_2} = u_\tau f(y^+) + \vec{V}.\vec{\tau_2}$ in the fixed frame, where u_τ denotes the friction velocity and $f(y^+)$ the wall function chosen which has to be adapted to the moving domain

context. In particular, we need to introduce the shape velocity in the implicit relation which defines $u_\tau$numerically. Hence, we have

$$\vec{u_1}(t+\delta t).\vec{\tau_1} = u_\tau f(y^+) - \vec{u_1}(t).(\vec{\tau_2}(t+\delta t) - \vec{\tau_1}) + \vec{V}(t+\delta t).\vec{\tau_2}(t+\delta t).$$

These relations give satisfactory results when the shape curvature and the amount of the deformation are not high. They are therefore suitable for steady or unsteady shape optimization and control problems involving small shape deformations as usually regions of large curvature are left unchanged due to constraints.

Remark for steady configurations:

The previous approach is valid for both steady and unsteady configurations. However, in steady shape optimization, we can proceed to some simplification. In particular, the terms involving the shape speed can be dropped and the geometry is not anymore time dependent. The previous relations are used therefore in the following manner:

$$\vec{u_1}(t+\delta t).\vec{n_1} = -\vec{u_1}(t).(\vec{n_2} - \vec{n_1}),$$

$$\vec{u_1}(t+\delta t).\vec{\tau_1} = u_\tau f(y^+) - \vec{u_1}(t).(\vec{\tau_2} - \vec{\tau_1}).$$

Remark

As we said, for unsteady configurations, when the amount of shape deformation is small, a good choice might be to prescribe equivalent injection boundary condition. However, in multi-disciplinary applications (for instance in aeroelasticity) the amount of shape deformation due the coupling might be large and when introducing multi-disciplinary optimization, equivalent injection boundary conditions are not anymore sufficient. This is why for such applications, an ALE formulation should be preferred to enable for mesh deformation as for steady configurations and to take into account the mesh nodes velocity in the flow equations [7, 8, 9, 11].

3.4 Geometrical constraints

Geometrical constraints are of different types. We consider some of them :

- Defining two limiting surfaces (curves in 2D), shape variations are allowed between them.
- For shapes such as blade, wing or airfoil, the second constraint can be for instance that for the original plan-form to remain unchanged.
- We can also require for instance that some parts of the geometry, such as the leading and trailing edges for wings, remain untouched.

- The previous constraints involve the parameterization points individually. We can introduce two global constraints for volume and thickness. We can ask for the volume and the maximum thickness for any section of the shape to remain unchanged.

There are several ways to take into account equality and inequality constraints. The easiest way to treat the first, second and third constraints is by projection :

$$\delta\tilde{x}_w = \Pi(\delta x_w).$$

The last constraint can be introduced in the optimization problem by penalty in the cost function ($J(x_w)$ being the cost function and $\alpha > 0$) :

$$J(\tilde{x}_w) = J(x_w) + \alpha|V - V_0|.$$

The constraint on the thickness is more difficult to take into account. We proceed as follows :

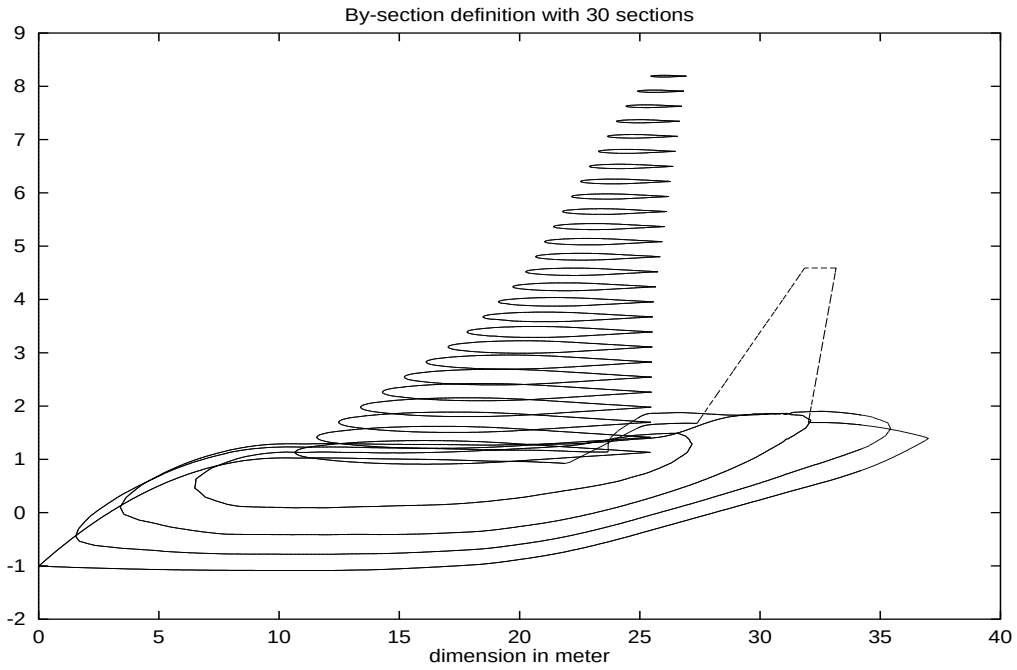

Figure 3: *By-section definition of an aircraft from its unstructured surface mesh.*

we define a by-section (see figure) definition of the shape where the number of sections required is free and depends on the complexity of the geometry,

each node in the parameterization is associated to a section Σ,

for each section, we define the maximum thickness Δ,

$$\Delta(x_w) = x_w \rightarrow \Sigma \rightarrow \Delta,$$

we ask for the shape deformation to be in the kernel of the gradient of Δ, i.e. the maximum thickness to have a local minima :

$$\nabla_{x_w}(\Delta(x_w)).\delta\tilde{x}_w = 0)$$

and therefore that $\Delta x_w = \Delta(x_w + \delta\tilde{x}_w)$. This means that we need a projection operator over $Ker(\nabla_{x_w}(\Delta(x_w)).)$. We can also introduce this constraint in the cost function by penalty :

$$J(\tilde{x}_w) = J(x_w) + \alpha|V - V_0| + \beta|\Delta - \Delta_0|.$$

Given lift constraint:

Most aerodynamic shape optimizations are required to be at given lift. It is interesting to notice that in cruise situation (far from stall), the lift is linear with respect to the angle of incidence. This gives therefore another way to enforce the given lift constraint. Indeed, during optimization the incidence follows the following equation:

$$\theta^{n+1} = \theta^n - \alpha(C_l^n - C_l^{target}),$$

where n is the optimization iteration (see below).

4 Minimization algorithms

This section is devoted to the description of the methods we use in our optimization and control platform. We insist on the integration of the methods and the approximations and simplifications we perform. The aim is to show the results we can expect after these modifications of standard minimization methods. We also emphasis on the impact of the incomplete gradient approach described before, when used with these methods. More details on the optimization methods can be found in dedicated books.

In our platform two approaches are used to integrate the optimization algorithms depending on the fact that the phenomena is time dependent or not.

Let us reconsider the following time dependent optimization or control problem:

$$\min_{x(t)} J(x(t), q(x), U(q), \nabla U(q)), \tag{5}$$

$$E(x(t), q(x), U(q), \nabla U(q)) = 0,$$

$$g_1(x(t)) \leq 0, g_2(q(x)) \leq 0, \quad g_3(q, U(q)) \leq 0,$$

where $x \in R^{n_c}$ describe our parameterization. This can be for optimization problems a geometrical CAD-based model or a CAD-free model described earlier and

for control problems, the amount of the injection/suction velocity for instance; the location of the injection devices is supposed to be given. q describes geometrical entities (normals, surfaces, volumes,...). $U \in R^N$ denotes the flow variables and $E \in R^N$ the time dependent state equation. g_1 defines the constraints on the parameterization, g_2 those on geometrical quantities and g_3 state constraints on U.

4.1 State equations

Let us consider for $E(x(t), q(x), U(q), \nabla U(q))$ a multi-disciplinary possibility where E gathers, for instance, the time dependent Navier-Stokes and the $k-\varepsilon$ turbulence model together with wall-laws for the fluid part and an time dependent elastic model for the structure. This is interesting for the explanation of the different points in the pseudo-unsteady optimization algorithms below. More precisely, we have

$$\frac{\partial U(x(t))}{\partial t} + \nabla_x.(F(x, U(x), \nabla_x U(x)) - S(U(x)) = 0 \quad \text{in} \quad \Omega_{fluid}, \tag{6}$$

where U is the vector of conservative variables (i.e. $U = (\rho, \rho\vec{u}, \rho(C_v T + \frac{1}{2}|\vec{u}|^2))^t$, $\rho k, \rho\varepsilon$), F represents the advective and viscous operators and S the right-hand-side in the $k-\varepsilon$ turbulence model as described in chapter 2 and 3. Ω_{fluid} denotes the fluid domain of resolution. This system has 6 equations in 2D (7 in 3D) and the system is closed using the equation of state $p = p(\rho, T)$.

To predict the behavior of the structure under aerodynamic forces, we consider the following time dependent elastic model for the structural degrees of freedom $X(t) \in R^{ns}$:

$$M\ddot{X} + D\dot{X} + KX = F(U(x(t))|_\Gamma) \quad \text{in} \quad \Omega_{structure}, \tag{7}$$

where M, D and K are the mass, damping and stiffness matrices and describe the structure characteristics. F contains the external forces acting on the structure and in particular the aerodynamic forces defined over $\Gamma = \partial\Omega_{structure} \cap \partial\Omega_{fluid}$.

X represents for instance the nodes coordinates in a finite element mesh. Using this mesh, we define for the structure, as for the fluid, a corresponding CAD-Free definition of the shape: $X|_{\partial\Omega_{structure}}(t)$. This definition is usually different from the one used for the shape represented in the fluid side. This is because the mesh used to solve 6 is different from the one used for 7. We have therefore to define transfer operator from and to $X(t)$, from and to $x(t)$ as we will explain later. More precisely, we need to express how to transfer structural deformation from X to x and aerodynamic forces as well as shape deformation due to optimization from the x to X. This of course, if we choose as control parameters x.

4.2 Pseudo-unsteady control and optimization algorithm

There is a major difference between control and optimization in our approach in the treatment of shape and mesh deformation and it comes from the fact that in control problems, the deformation and therefore its equivalent injection velocity are small. This implies that equivalent injection boundary conditions are quite suitable while shape optimization, where the amount of the deformation is not a priori known, requires a full shape and mesh deformation strategy. The algorithm is presented with a simple steepest descent method with projection for the optimization. We will see later how to introduce more sophisticated methods expressed through other pseudo-unsteady systems.

To solve the above problem (5), we need an equation for $x(t)$. Consider the following simple pseudo-unsteady system representing a descent algorithm considered as an equation for the shape parameterization. Here, the time is fictitious and is similar to the descent parameter.

$$\dot{x} = \Pi(-\nabla_x J).$$

To advance in time (4.2), we use a backward Euler scheme (denotes by δx^p the shape deformation at step p):

$$\delta x^p = \Pi(-\lambda \nabla_{x^p} J^p),$$

where Π is the projection operator over the admissible space.

This is well adapted to incomplete gradients and incomplete intermediate solution of the state equation. λ denotes the descent step size and is seen as a fictitious time. It defines the coupling between the equation for the shape (pseudo-unsteady system of optimization) and the state equation (fluid dynamic equations). The idea is similar to the local time stepping method and is based on the fact that, for steady solutions, we are not interested by intermediate shapes, sensitivities and solutions. Therefore, approximations can be made so long as convergence to a steady state is achieved (by state we mean the shape and the solution) and the cost function is decreased to a minimum.. This remark is both true for shape optimization where we are looking for a steady state to this coupled system and for control where this defines the control law to be applied to the flow by injection.

We denote the control parameter at iteration n by x^n and the unsteady cost function $J(x^n, U(x^n), \nabla_{x^n} U(x^n))$ by $J(x^n)$.

The control space x^0 being defined, the algorithm is as follows:

Coupling loop

1. compute the gradient: $\frac{dJ^p}{dx}$,
2. define the new admissible shape deformation by (4.2),
3. smooth the deformations,

4. If aeroelastic add to the physical structural deformation,

IF small deformations:

5a. define the normals to the deformed shape and
the shape speed in the fixed reference,
6a. define the new injection velocity,

ELSEIF large deformations:

5b. deform the mesh.
6b. compute the new geometrical entities
and mesh speed with consistent approximations
of the boundary conditions and shape deformations for the structure and fluid models :

$$\dot{x}_{mf}^{p+1} = \frac{x_{mf}^{p+1} - x_{mf}^{p}}{\lambda},$$

$$\int_{sf} \sigma_{sf}^{p}.n_{sf}^{p+1} ds = \int_{ss} \sigma_{ss}^{p}.n_{ss}^{p+1} ds, ,$$

$$\int_{sf} \dot{x}_{sf}^{p+1} \sigma_{sf}^{p} ds = \int_{ss} \dot{x}_{ss}^{p+1} \sigma_{ss}^{p} ds.$$

end if

7. advance in time by λ the state equation in Eulerian or ALE formulation: U^{p+1}.
8. If shape optimization and if $J^{p+1} < TOL$, stop.

End of the coupling loop.

Where, subscript mf denote the fluid mesh, and sf and ss denote the quantities (shape discretization and speed, constraint tensor) on the fluid and structure sides respectively.

This compatibility relation enforces action-reaction principle and energy conservation. Of course, the coupling algorithm can be more sophisticated to improve time accuracy [7, 8, 9, 11].

4.3 More sophisticated pseudo-unsteady systems

Consider the following second order time dependent equation for the shape parameterization x.

$$\dot{x} + \epsilon \ \ddot{x} = -F(\Pi, M^{-1}, (\nabla_{xx} J)^{-1}, \nabla_x J), \tag{8}$$

where F is a function of the exact or incomplete gradient and of the inverse of the Hessian of the cost function. It also takes into account the projection over the admissible space Π and the smoothing operator M described above.

To advance in time (4.2), we use a central difference scheme for the first and second order operators. It is more appropriate to use a forward difference scheme

for the first order derivative when targeting complex applications as intermediate states and sensitivities are not accurately computed due to computational effort and this error introduce an extra perturbation to the second order pseudo-unsteady system. This somehow compensates the natural unstable behavior of the method.

Denotes by δx^p the shape deformation at step p, the discretized (8) reads:

$$(\frac{\epsilon}{\lambda^2}+\frac{1}{\lambda})\delta x^{p+1} = \frac{\epsilon}{\lambda^2}\delta x^p - F(\Pi^p, (M^p)^{-1}, (\nabla_{xx}J^p)^{-1}, \nabla_{x^p}J^p).$$

Usually, Π does not depend on p except when using mesh adaptation as seen below.

Through $\epsilon \geq 0$, we can control the deviation of (8) from the steepest descent method. Indeed, with $\epsilon = 0$, we recover the steepest descent with fixed step size if the fictitious time step λ is fixed and if F does not depend on the Hessian. Of course, λ can be tuned to be optimal at each time step and we recover the optimal steepest descent method which necessarily converges to the closest minima.

In the same way, a Newton or quasi-Newton type method can be expressed introducing the inverse of the Hessian or its approximation through a quasi-Newton iterative formula like BFGS or instance. In these methods $(\nabla_{xx}J^p)^{-1}$ is approximated by a symmetric definite positive matrix $\mathbf{H}^p$, constructed iteratively, starting from the identity matrix for instance :

$$\mathbf{H}^{p+1}_{\text{BFGS}} = \mathbf{H}^p + \left(1+\frac{\gamma^{p^{\mathrm{T}}}\mathbf{H}^p\gamma^p}{\delta x^{p^{\mathrm{T}}}\gamma^p}\right)\frac{\delta x^p \delta x^{p^{\mathrm{T}}}}{\delta x^{p^{\mathrm{T}}}\gamma^p} - \frac{1}{2}\frac{\delta x^{p^T}(\mathbf{H}^p+\mathbf{H}^p)\gamma^p}{\delta x^p\gamma^p}, \tag{9}$$

with

$$\gamma^p = \nabla J\left(x^{p+1}\right) - \nabla J\left(x^p\right).$$

This BFGS formula has been used together with the incomplete gradient approach and has significantly improved the convergence of the optimization problem in our case (see below).

If λ is p-dependent and if we remove the first order derivative, we recover methods such as conjugate gradient were F has a particular expression. Indeed, it is easy to check that the conjugate gradient method is equivalent to :

$$x^{p+1} - 2x^p + x^{p-1} = (-\lambda^p||g^p||\sum_{i=1}^{p}\frac{g^i}{||g^i||} + \lambda^{p-1}||g^{p-1}||\sum_{i=1}^{p-1}\frac{g^i}{||g^i||}), \tag{10}$$

which can be written after adding projection and smoothing operators as:

$$\delta x^{p+1} = \Pi(\delta x^p + M^{-1}(-\lambda^p||g^p||\sum_{i=1}^{p}\frac{g^i}{||g^i||} + \lambda^{p-1}||g^{p-1}||\sum_{i=1}^{p-1}\frac{g^i}{||g^i||})), \tag{11}$$

where $\delta x^p = x^p - x^{p-1}$ and $g^i = \nabla^i_x J^i$.

The pseudo-unsteady system (11) is second order without first order term and F involves the integral of the cost function sensitivity:

$$\delta x^{p+1} = \Pi(\delta x^p + M^{-1}(-\lambda^p g^p + ||g^{p-1}|| \sum_{i=1}^{p-1} (\lambda^{p-1} - \lambda^p) \frac{g^i}{||g^i||})), \tag{12}$$

which can be seen as a discretization of (without smoothing and projection operators):

$$\ddot{x} = \frac{-\nabla J(T_2)}{\lambda_2} + \frac{(\lambda_1 - \lambda_2)}{\lambda_2^2} ||\nabla J(T_1)|| \int_0^T \frac{\nabla J(\tau)}{||\nabla J(\tau)||} d\tau,$$

where $\lambda_i = argmin_\lambda (J(T_i) - \lambda \nabla J(T_i))$ and with $T_1 < T_2$.

Note that if λ is fixed in (10) and if $||g^p|| = ||g^{p-1}||$ we have:

$$x^{p+1} - 2x^p + x^{p-1} = \Pi(-\lambda g^p)$$

which describes the motion of a heavy ball without friction and does not has a stationary solution. This shows the importance of an optimal step size in conjugate gradient method. This situation is classical and corresponds to the motion of a ball over a curve defined by an absolute value function smoothed around zero.

In the presence of a first order friction term, (8) is the so called heavy ball method [12, 13]. The aim in this approach is to access different minima of the problem and not only the nearest local minimum by helping to escape from local minima after introduction of second order perturbation term. The difference with the original heavy ball method is based on the fact that here the method is seen as a perturbation of the first order derivative while in the original heavy ball method the steepest descent is seen as a perturbation of the hyperbolic second order system. This reformulation is suitable for numerical experiences as it enables to tune the perturbation to be the weakest possible. Indeed, otherwise especially for complex applications, the optimization process becomes hard to control.

4.3.1 Coupling pseudo-unsteady systems to improve global minimization

Experience shows that despite the heavy ball method can escape from nearest local minima, reaching the global minima requires too many tries with various initial conditions. In addition, introducing too much initial speed for a ball makes the method unstable. The idea is to solve the pseudo-unsteady system (8) from different ball positions (point in the admissible space) and to couple the paths using cross informations involving a global gradient [14]. Consider, q balls $x_j, j = 1, .., q$, following the motion prescribed by q pseudo-unsteady systems:

$$\dot{x_j} + \epsilon \ \ddot{x_j} = -(F_j + G_j), \tag{13}$$

where F_j is as in (8) and G_j a global gradient representing the interaction between balls (recall that each ball is a design configuration). This has also shown a stability improvement. To be able to reach the global minima, the number of balls has to be enough large. A good estimation for this number is given by the dimension of the design space (n). Even with this number the complexity is negligible compared to those of genetic algorithms. Our experience shows that the following choice of G_j gives satisfaction (see example below):

$$(G_j)_i = \sum_{k=1,k\neq j}^{q} \frac{J_j - J_k}{||x_j - x_k||^2}(x_{j_i} - x_{ki}), \quad \text{for} \quad j = 1,..,q, \quad i = 1,..,n.$$

However, in CAD-Free parameterization, n can be quite large and, due to the required computational effort for one simulation, we cannot afford for more than a few (say 3 or 4) shape evolutions at the same time. This approach can be seen therefore as an improvement of the the search capacity of the original algorithm. In addition, the process is suitable for a distributed treatment as in genetic type minimizations.

4.3.2 Global minimization on a simple example

The behavior of the pseudo-unsteady systems (13) with two balls and constant λ and ϵ, is shown in Fig. (4-left) for the minimization of the following Griewank's function. This function has several local minima (the global minimum is reached at (0,0)). For $(x, y) \in]-10, 10[\times]-10, 10[$ consider J defined by:

$$J(x, y) = 1 - \cos(x)\cos(\frac{y}{\sqrt{2}}) + \frac{1}{50}((x - 0.5 * y)^2 + 1.75y^2).$$

The aim is to show that heavy ball method improves global minimum search by helping to escape from local minima (figure 5-left). Finding the global minima requires however several tries. But coupling several heavy balls can help finding the global minima (figure 5) using balls not converging to the global minima individually (figure 4-right). It is also interesting to notice that the steepest descent method with fixed step size also converge to the global minima, but through chaotic paths. This is one advantage of using constant step size rather than optimal when there are several local minima: at the cost of one steepest descent minimization with optimal descent step size we can perform several steepest descent minimizations each time with a fixed step size.

Other similar ideas introducing attraction and repulsion forces between balls in the dynamic systems above have been proposed [14].

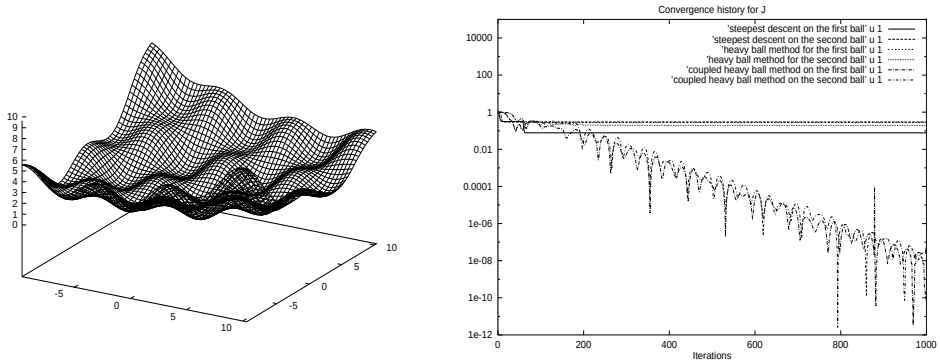

Figure 4: *Pseudo-Unsteady minimization for the Griewank's function (left). Convergence histories for the steepest descent and heavy ball methods starting from two different points, all captured by local minima.*

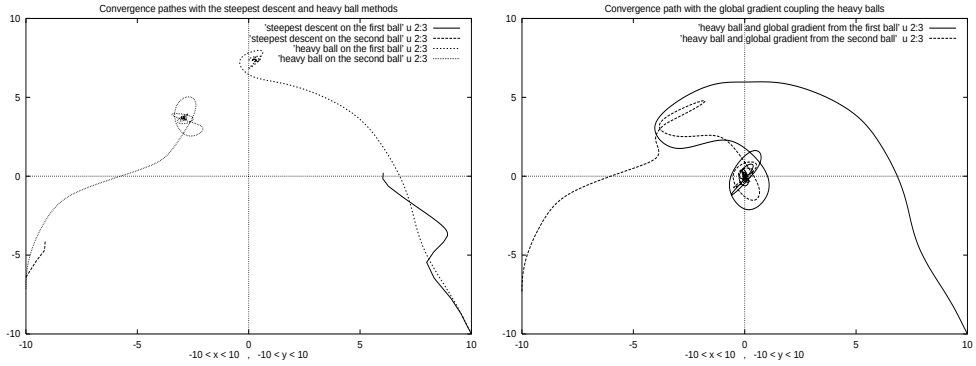

Figure 5: *Paths for the steepest descent and heavy ball methods starting from two different points (left). When coupling the two balls in heavy ball method using the global gradient. The global minimum is reached (right).*

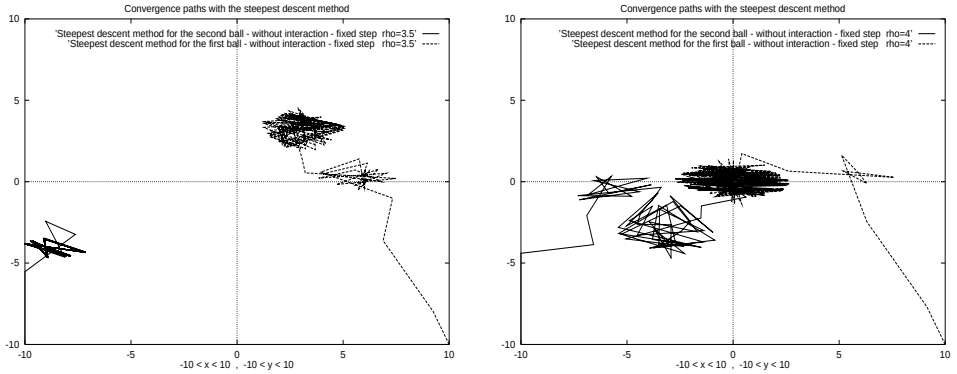

Figure 6: *Paths for the steepest descent method with fixed step size starting from the same points than above. Keeping the step size fix during minimization enables to reach the global minimum (right).*

4.4 Interior point algorithms

To avoid violation or saturation of inequality constraints, the idea behind interior point algorithms is to add penalty term to the cost function involving the distance to the frontier of the admissible space. If J denotes the original cost function, and d the distance of control points (in some given norm) to the boundaries of the admissible space, we can redefine the cost function as:

$$\tilde{J} = J + \chi(\frac{d}{d_0} - 1)^2.$$

As an example, in shape optimization problems, when the deformations have upper and lower bounds, we use the following distance definition:

$$\overline{x_i} < x_i < \underline{x_i}, \quad d = (\sum_{i=1}^{n} (x_i - \overline{x_i})^2 + (x_i - \underline{x_i})^2)^{1/2}.$$

This approach is easy to implement but is limited by the choice of χ.

An alternative IPA algorithm based on a modified BFGS problem to satisfy inequality constraints at each step is given in chapter 1.

4.4.1 Descent step size

As we said, experience shows that tuning the descent step size to be optimal after line search is not always optimal for optimization problems as we are fast captured by the nearest local minima. To avoid this, in our applications, we usually prefer a constant descent step size as shown in the example above. In addition, tuning an optimal descent step size usually requires several state evaluations.

Of course, by doing so we loose the theoretical convergence results.

5 Efficiency with AD

In this section, we describe a few ingredients we need for an efficient use of AD and also to seriously reduce the cost of sensitivity evaluation for some important cost functions.

5.1 Limitations when using AD

As we said, we use the automatic differentiator Odyssée developed at INRIA [21, 22, 23]. Direct differentiation, producing a Jacobian matrix or a gradient vector, as well as reverse mode, computing the cotangent linear application, are implemented in Odyssée (see chapter 4).

The limitations of the reverse mode of AD come from the required memory for Lagrange multipliers and intermediate variables.

The need for intermediate variables storage can be understood by the following argument:

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

$$f(x) = f_3 o f_2 o f_1 o f_0(x).$$

In our shape optimization approach, f_0 smoothes the control point variations, f_1 propagates the control point variations overall the mesh, f_2 computes the flow and f_3 evaluates the cost. The reverse mode produces the Jacobian $D^t f : R \to R^n$, as

$$D^t_x f = D^t_x f_0 o D^t_{f_0(x)} f_1 o D^t_{f_1 o f_0(x)} f_2 o D^t_{f_2 o f_1 o f_0(x)} f_3.$$

We can see that we have to store (or recompute) the intermediate states (i.e. $f_0(x), f_1 o f_0(x)$ and $f_2 o f_1 o f_0(x)$) 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 the cost will growth as the square of the number of iterations. Therefore, they have to be somehow stored.

To give a more precise idea, consider our flow solver which is explicit with one external loop for time integration of size `KT` and internal nested loops.

```
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 affectations like:

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

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

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

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

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

which makes $M \sim 10^{11} Mots \sim 100GO$. We can use an implicit method to reduce the number of time steps and therefore the storage. One alternative here is to use the check-pointing scheme of Griewank [24] to have a base 2 logarithmic growth of the complexity of the algorithm.

5.2 Storage strategies

To understand the need for a storage strategy, let us look at what we get if we store all the intermediate states, no intermediate state and some of them.

Full storage case:

KT states stored, KT forward iterations (i.e. one forward integration) and KT backward iterations (i.e. one backward integration).

No storage case:

0 states stored, $(KT + KT * (KT - 1)/2)$ forward iterations (i.e. one forward integration plus computing the required state during the backward integration each time) and KT backward iterations (i.e. one backward integration).

Intermediate storage:

n states stored non-uniformly over KT states (with $n << KT$), $(KT + \Sigma_{i=1}^{n}(m_i * (m_i - 1)/2))$ forward iterations (i.e. one forward integration plus computing the required state each time starting from the nearest stored state, m_i

being the number of iterations between two closest stored states) and KT backward iterations (i.e. one backward integration). Of course, if the stored states are uniformly distributed, we have $(KT + n(m * (m-1)/2))$ forward iterations, so with one state stored $(n = 1)$, we reduce by a factor of two the extra work $(m = KT/2)$.

We can see that in each case, the backward integration has been done only once.

5.3 Keys points when using AD

These are some remarks coming from the physics of the problem. They are important for a more efficient use of the AD in reverse mode in optimization problems. We present the different steps in the chronological way. We aim to show how we have been brought to the incomplete sensitivity approach we use today.

5.3.1 Steady flows

The first key remark concerns target applications which are steady. In the sense that the history of the computation is not of interest. Actually, this history is often evaluated using low accuracy method and local time stepping. It is obvious therefore that is not necessary to store these intermediate states. In other words, when computing the gradient by 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 steps (KT).

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

5.3.2 Inter-procedural differentiation

The second important point is to use inter-procedural [23] derivation. The idea is to use dynamic memory allocation in computers. We replace what is inside an internal loop by a subroutine and derive this subroutine. This will reduce the required memory to:

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

but will imply extra calls to subroutines [23]. In fact, this is what we need if the memory is declared as static and we need less if we choose to re-allow the memory needed by routines.

5.3.3 Adjoint accuracy for steady state applications

We saw that, when 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 which 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 criteria 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 criteria is satisfied. In the previous code, we need therefore to evaluate the cost function inside the time integration loop but this is cheap.

5.3.4 Localized gradient computation

This remark can be understood from picture 7. We can see that the variations of the internal mesh nodes have no effect on the cost function. Therefore, when computing the gradient, the internal loops presented above on nodes, segments and elements are only computed on a few element layers around the shape.

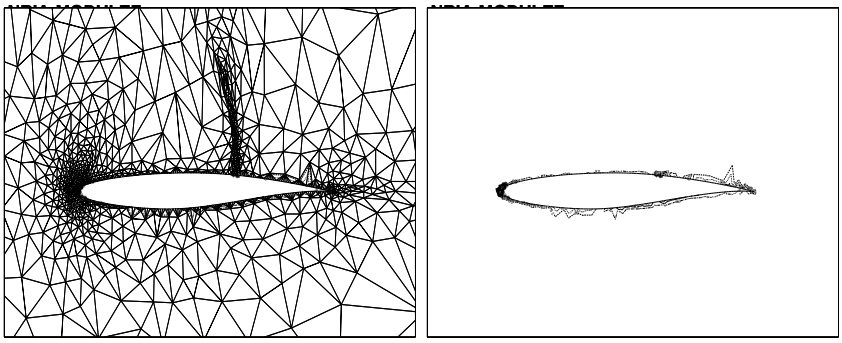

Figure 7: $\frac{\partial J}{\partial x_{mesh}}$. *We can see that the gradient is concentrated along the first element layer along the geometry. This is a key remark to reduce the computational effort.*

6 Incomplete sensitivities and boundary integrals

We noticed that the previous remark was mainly valid for cost functions based on informations on the shape (for instance through boundary integrals). Fortunately, this is often the case for the applications. Therefore, when the cost

function is based on local informations around the shape, the sensitivity of the cost function with respect to the state can be neglected. This does not mean that a precise evaluation of the state is not necessary, but that for a small change in the shape the state will remain almost unchanged, while geometrical quantities have variations of the same order:

$$\delta x \sim |\delta n| \sim O(1), \quad \text{but} \quad \delta U \sim O(\varepsilon).$$

6.1 Incomplete sensitivities and equivalent boundary condition

We recover here the argument behind the equivalent injection boundary condition developed above and widely used to represent small shape deformations. More precisely, consider the sensitivity of the product $\vec{u}.\vec{n}$ with respect to the shape parameterization x. Formally, we have:

$$\frac{d}{dx}(\vec{u}.\vec{n}) = \frac{d\vec{u}}{dx}.\vec{n} + \vec{u}\frac{d\vec{n}}{dx} \sim \vec{u}\frac{d\vec{n}}{dx}.$$

Indeed, the argument above means that $|\frac{d\vec{u}}{dx}| << |\frac{d\vec{n}}{dx}|$. We notice that u has to be accurate for an accurate sensitivity.

6.2 Incomplete gradient: application to advection-diffusion equation

We recall that the idea behind incomplete gradients is that the changes in the state are negligible compared to those of the geometry for small variations of the domain.

Consider as cost function $J = au_x(a)$ and as state equation the following steady advection-diffusion equation:

$$u_x - Pe^{-1}\ u_{xx} = 0, \quad \text{on} \quad]a,1[, \quad u(a) = 0, \quad u(1) = 1.$$

The solution of this equation is

$$u(x) \ = \ \frac{\exp(Pe^{-1}\ a) - \exp(Pe^{-1}\ x)}{\exp(Pe^{-1}\ a) - \exp(Pe^{-1})}.$$

We are looking for $J_a(a) = u_x(a) + a(u_x)_a(a)$. We are in the domain of application of the incomplete sensitivities, where the cost function involves products of state and geometrical quantities and is defined at the boundary.

We can linearize expression (6.2):

$$u_x(x) = \frac{-Pe^{-1}\ \exp(Pe^{-1}\ x)}{\exp(Pe^{-1}\ a) - \exp(Pe^{-1})},$$

and get $(u_x)_a(a)$:

$$(u_x)_a(a) = \frac{-(Pe^{-1}\ \exp(Pe^{-1}\ a))^2}{(\exp(Pe^{-1}\ a) - \exp(Pe^{-1}))^2}.$$

We have therefore,

$$J_a(a) = u_x\ (a)\left(1 + a\frac{Pe^{-1}\ \exp(Pe^{-1}\ a)}{\exp(Pe^{-1}\ a) - \exp(Pe^{-1})}\right)$$
$$= u_x\ (a)\,(1 + a|u_x\ (a)|)\,.$$

We can see that the contribution coming from the state linearization is in Pe^{-2} which means that in applications where $Pe > 1$, the approximate gradient is quite accurate, in all case, the sign is always correct.

The previous analysis holds if $J = f(a)g(u, u_x)$ for instance.

6.3 Incomplete gradient: application to channel flows

Another example we analyzed in [25] concerns the Poiseuille flow in a channel driven by a constant pressure gradient (p_x). The walls are at $y = +/-a$. The flow velocity satisfies:

$$u_{yy} = \frac{p_x}{\nu}, \quad u(-a) = u(a) = 0. \tag{14}$$

The analytical solution satisfying the boundary conditions is: $u(a, y) = \frac{p_x}{2\nu}(y^2 - a^2)$.

We are interested by the sensitivity of the flow rate when the channel thickness changes, using exact and incomplete gradients. The flow rate is given by $J_1(a) = \int_{-a}^{a} u(a, y)dy$. The gradient is given by (using the boundary conditions in (14)):

$$\frac{dJ_1}{da} = \int_{-a}^{a} \partial_a U(a, y)dy = \frac{-2a^2 p_x}{\nu},$$

while the incomplete sensitivity (dropping the state linearization) vanishes.

Now consider the following cost function obtained multiplying the flow rate by a^n:

$$J_2(a) = a^n J_1(a).$$

The gradient of the cost function with respect to the control variable a is:

$$\frac{dJ_2}{da} = na^{n-1}J_1(a) + a^n\frac{dJ_1}{da}.$$

We can see now that we have two terms involved: the first integral is what the incomplete sensitivity analysis gives and the second comes from the linearization of the state. We have,

$$\frac{dJ_2}{da} = -\frac{4na^{n+2}p_x}{6\nu} - \frac{a^{n+2}p_x}{\nu}.$$

We can see that the two contributions have the same sign and are of the same order, of course the geometrical sensitivity dominates for large values of n. This remark is interesting for axisymmetric applications in nozzles for instance.

6.4 Incomplete sensitivities: time dependent phenomena

An important question is the validity of these approximation for time dependent cost functions. Indeed, if the approximation holds, this enables for real time sensitivity definition in the sense that the state and the sensitivities are available in parallel without the need of solving an adjoint problem.

6.5 Newton law: pressure distribution prediction for bluff bodies

A worst case analysis for inviscid flows comes from the Newton formula for the pressure distribution for high curvature bodies using the inflow velocity $\vec{u_\infty}$ and the local normal to the shape $\vec{n}$:

$$p \sim (\frac{\vec{n}.\vec{u_\infty}}{|\vec{u_\infty}|})^2.$$

This is also called the cosines-square law. In that case, the drag coefficient is easily expressed through local shape-based geometrical informations:

$$C_d = \int_{shape} cos^2(\vec{n}.\frac{\vec{u_\infty}}{|\vec{u_\infty}|})\vec{n}.\frac{\vec{u_\infty}}{|\vec{u_\infty}|}d\sigma.$$

We can see that linearization leads to equal order sensitivities for small variations of the normal or the inflow. This is the worst case as we know that small changes in the geometry in high curvature area (leading edge) have important effects on the flow, much more than changes in area where the shape is flat.

Now, consider a local inviscid drag evaluation involving the pressure on the shape, the inward normal and the inflow velocity ($C_d = pu_\infty.n$), x being our control parameter, sensitivity analysis gives:

$$\frac{d}{dx}(pu_\infty.n) = \frac{dp}{dx}(u_\infty.n) + p\frac{dn}{dx}.u_\infty \sim p\frac{dn}{dx}.u_\infty.$$

Here, the same argument has been used to drop the term in dp/dx because $\frac{dp}{dx} << \frac{dn}{dx}$. We notice again that p (the state) has to be accurate.

This shows for instance the importance of including viscous effects in the state evaluation for optimization [2] but not necessarily to take them into account in the sensitivities.

These remarks are essential, as usually users are penalized by the cost of sensitivity evaluations which forces them to use coarser meshes in optimization than the meshes they can afford for simulation.

6.6 General formulation and validity domain

Consider the general simulation loop, involved in (5), leading from shape parameterization to the cost function:

$$J(x) : x \to q(x) \to U(q(x)) \to J(x, q(x), U(q(x))).$$

If both the cost function and control space are defined on the shape (or a same part of it) and if J is of the form

$$J(x) = \int_{\text{shape or part of the shape}} f(x)g(u)d\gamma,$$

then the incomplete sensitivity approach holds.

More precisely, the gradient of J has different ingredients:

$$\frac{dJ}{dx} = \frac{\partial J}{\partial x} + \frac{\partial J}{\partial q}\frac{\partial q}{\partial x} + \frac{\partial J}{\partial U}\frac{\partial U}{\partial q}\frac{\partial q}{\partial x}.$$

The incomplete sensitivity approach means that we can drop the last term in (6.6):

$$\frac{dJ}{dx} \sim \frac{\partial J}{\partial x} + \frac{\partial J}{\partial q}\frac{\partial q}{\partial x}.$$

This means that for cost function based on boundary integrals, we can only work on the shape for sensitivity evaluation. It is not necessary to linearize the mesh deformation or mesh generation tools (see below) and the state equation. This is therefore one step further compared to the localized gradient strategy explained above.

6.7 Multi-Level construction

From expression (6.5) above it is clear that it is more accurate to have an accurate state evaluation and an approximate gradient than to try to compute an accurate gradient based on a wrong state as obtained on a coarse mesh:

$$|\frac{d}{dx}(p.n) - p(\text{fine})\frac{dn}{dx}| < |\frac{d}{dx}(p.n) - \frac{dp}{dx}(\text{coarse}).n + p(\text{coarse})\frac{dn}{dx}|.$$

The left hand side is the difference between exact and incomplete gradient computed on a fine mesh.

This error is often present and is due to the fact that the cost of iterative minimization and gradient evaluations force to use coarser meshes than what would have been used for a pure simulation.

One possibility to avoid this difficulty is to use different level of refinement for the state and the gradient. This is the idea behind multi-level shape optimization where the gradient is only computed on the coarse level of a multi-grid construction and where the state comes from the finer level [16]:

$$\frac{d}{dx}(p.n)(\text{fine level}) = I(\frac{dp}{dx}(\text{coarse level})).n + p(\text{fine level})\frac{dn}{dx}.$$

The first term of the left hand side is the interpolation of the gradient computed on the coarse grid over the fine level.

6.8 Back to finite differences or complex variables method

Consider again the relation above leading to J from x and define another relation such that

$$\overline{J(x,U)} = J(x) = (x,U) \rightarrow q(x) \rightarrow J(x,q(x),U).$$

In the definition of $\overline{J}$, U is left unchanged. Knowing that with finite differences, most of the cost comes from sensitivity evaluation for the state, this means that FD or CVM become viable choices:

$$\frac{dJ}{dx}(i) = \frac{J(x_i+\varepsilon) - J(x)}{\varepsilon} \sim \frac{\overline{J(x_i+\varepsilon,U)} - \overline{J(x,U)}}{\varepsilon},$$

where $x_i + \varepsilon$ means a small change in the i-th component of x.

6.9 Coupled configurations

Another important remark concerns coupled situations. We notice that for these applications also, the state equation for the structure part can be also removed from the gradient evaluation. More precisely, one iteration of our Fluid/Structure interaction can be seen as:

$$x \rightarrow U \rightarrow J \rightarrow X \rightarrow U \rightarrow J,$$

which describes the variations of respectively x the shape coordinates, U the state, J the cost function (aerodynamic coefficients), X the structure parameterization coordinates. The previous remarks mean that once the state and the final cost evaluated using this loop, in the sensibility evaluation, we only take into account for direct dependencies between J and x. Examples for the application of these remarks are given in chapter 8 and 9 where we use a simple elastic model which can be easily differentiated. But this remark becomes more interesting if we use a commercial code for the structure solution.

6.10 Incomplete sensitivities and the Hessian

An a posteriori validation of this argument comes from the fact that this approximation also works with BFGS algorithms [3] and not only for steepest descent methods. Indeed, this former method might converge if the sign of the incomplete gradient is correct. On the other hand, if we use our approximate gradient to set

up the Hessian in BFGS, not only the sigh has to be correct but also its value. Otherwise, the eigenvalues of the Hessian matrix should or at least might have the wrong sign.

Here we show one example of shape optimization comparing BFGS and steepest descent (SD) method both using fixed descent step size and using our incomplete sensitivities. The cost function is given by

$$J(x, q(x), U(q)) = \frac{C_d}{C_d^\circ} + \alpha \; max\left(0, \frac{(C_l^\circ - C_l)}{C_l^\circ}\right) + \beta \; max\left(0, \frac{(V^\circ - V)}{V^\circ}\right) \quad (15)$$

where α and β are penalty parameters, C_d and C_l are respectively the aerodynamic drag and lift coefficients and V the volume (0 indicates the corresponding value for the initial configuration). The idea is to reduce the drag and increase the volume and lift.

The initial geometry is an AGARD wing with an aspect ratio of 3.3 and a sweep angle of 45 degrees at the leading edge line. The computational mesh has 120000 tetrahedra, 2800 grid points being located on the surface wing. The upper surface grid for the baseline configuration is shown in Fig. (8). The flow is characterized by a free-stream Mach number of 2 and an angle of attack of 2 degrees. The flow solver requires nearly 300 iterations to drop the residual by three orders of magnitude with a value of the CFL number equal to 1.3. The initial values of the lift coefficient, C_l°, of the drag coefficient, C_d°, and of the volume, V°, are 0.04850, 0.00697 and 0.030545 respectively. Fig. (8) shows the convergence history of the objective function *J(x)* versus the required work, for the BFGS method and the SD method. The work required by the Hessian computation is equal to 0.027 and is equivalent to the work required to perform 8 flow iterations. The convergence plots contain several slope discontinuities, due to constraint activations. In the present test case, the volume and lift constraints represent more stringent design conditions than in the previous test case. The BFGS method exhibits a more rapid reduction of the objective function after the initial iterations. The objective function has been reduced by 23.5%. The BFGS method reaches the minimum in 150 iterations, while the SD method requires 450 iterations. The final values for the coefficients above are with the BFGS method ($Cl = 0.0495, Cd = 0.005, V = 0.03048$) and with the SD method ($Cl = 0.1473, Cd = 0.0188, V = 0.06018$). Hence, the drag coefficient has been reduced by 28%, the lift coefficient has been increased by 2%, while the volume remained almost unchanged.

Our experience with quasi-Newton methods and incomplete gradients indicate two conclusions: first, that the BFGS method is more efficient than the SD method and does not suffer too much from the approximation in the evaluation of the gradient, and then that the approximation of the gradient seems to be more effective in sub-sonic and transonic ranges [3, 4, 5].

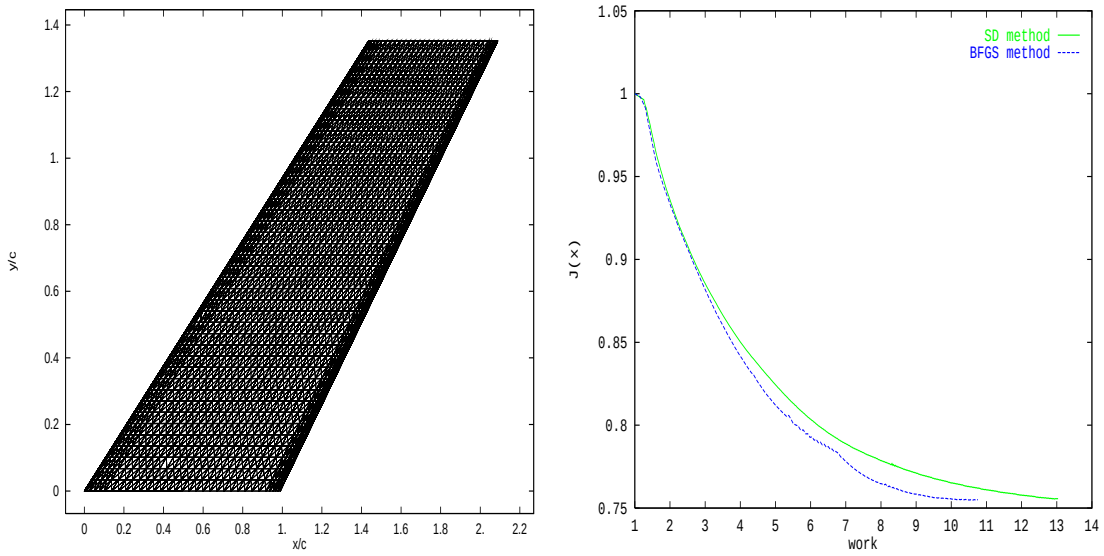

Figure 8: *AGARD wing: upper surface discretization (left). Convergence for the objective function (right). One unit of work corresponds to the time to converge one single flow analysis.*

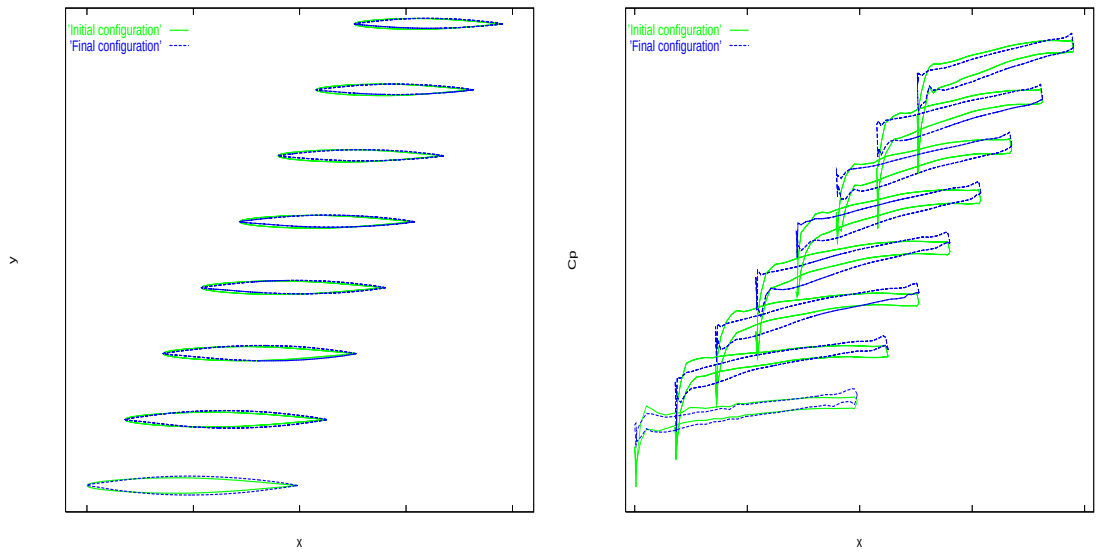

Figure 9: *AGARD wing: Initial and optimized wing shapes (left), BFGS and SD gives the same answer. Initial and optimized pressure coefficient distributions (right).*

6.11 Redefinition of cost functions

To be able to use the incomplete sensitivity approach, it is important to be in the validity domain presented above. This is often the case for cost functions based on aerodynamic coefficients like lift and drag. But if we consider a classical inverse design cost function,

$$J(x) = 0.5 \int_\Gamma (f(u) - f(u)_{target})^2 d\gamma,$$

the dependence in the geometry is missed in the cost function. For these problems, we use the following modified cost function:

$$J(x) = 0.5 \int_\Gamma (f(u) - f(u)_{target})^2 ((\frac{\vec{U_\infty} + \vec{U_\infty^\perp}}{||\vec{U_\infty}||}).\vec{n})^2 d\gamma.$$

We show examples of cost function and constraints redefinition in chapter 7 for blade optimization, sonic boom and wave drag reductions and heat transfer optimization. Of course, this is done only as a basis for the gradient evaluation and not for the cost function itself.

7 Mesh Adaptation and Optimization

This paragraph is devoted to the description of one example of coupling between shape optimization tools and mesh adaptation by metric control. The idea is to introduce mesh independence issues, as for direct simulations, in the context of shape optimization. Indeed, local mesh adaptation by metric control is a powerful tool for getting mesh independent results at a reduced cost [26, 27, 28].

When mesh adaptation is used, not only the mesh in the computational domain, but also the number and positions of the discretization points over the shape will change during optimization. This means that the shape and unstructured mesh deformation tools should take into account these transformations. In addition, in the CAD-Free framework, a change in shape discretization implies a change in the design space which we should avoid so as to keep solving the initial optimization problem.

Another remark concerns the change in the mesh connectivities in Delaunay type meshes. This is difficult to take into account into the gradients because an edge swap results in a non-differentiable change of the cost function. This is why optimization is usually performed on meshes with fixed connectivities and on a given control space and shape discretization (i.e. no control parameters or new discretization points on the shape are created during optimization). However, as for direct problems, the need for adapted meshes is clear in design problems.

7.1 Delaunay Mesh Generator

Given a positive definite matrix $M(x)$ we define a variable metric by $\|x-y\|^2 = (x-y)^T M(x)(x-y)$. M-circles, i.e. circles with respect to the variable metric, are ellipses in the Euclidean space.

A given triangulation is said to satisfy the M-Delaunay criteria if for all inner edges the quadrangle made by the its two adjacent triangles are such that the fourth vertex is outside the M-circle passing by the 3 other vertices.

The adaptive Delaunay mesh generator of [31] is based on five steps:

1. discretize the Boundary of Ω, the domain for which we seek a triangulation.

2. Build a M-Delaunay triangulation of the convex hull of all the boundary nodes (no internal point),

3. add an internal points to all edges which are longer (in the variable metric) than the prescribed length,

4. rebuild the triangulation with internal nodes now, using the Delaunay criteria and the variable metric,

5. remove the triangles which are outside Ω,

6. goto 3 until the elements have the required quality.

It can be shown that the Delaunay mesh is the nearest to a quasi-equilateral mesh, in the sense that the smallest angle in the triangulation is maximized. So, if the local metric is Euclidian, the mesh elements are isotropic. But, anisotropy is introduced by way of the local metric [31, 30].

7.2 Metric definition

As we said, if we want the mesh to be adapted to the solution, we need to define the metric at each point of the domain and use it in the Delaunay algorithm above. This is usually what is required by an unstructured mesh generator having adaptation capacities [31].

The definition of the metric is based on the Hessian of the state variables of the problem. Indeed, for a P^1 Lagrange discretization of a variable u, the interpolation error is bounded by:

$$\mathcal{E} = |u - \Pi_h u|_0 \leq c h^2 |D^2 u|_0, \tag{16}$$

where h is the element size, $\Pi_h u$ the P^1 interpolation of u and $D^2 u$ its Hessian matrix. This matrix is symmetric,

$$D^2 u = \begin{pmatrix} \partial^2 u/\partial x^2 & \partial^2 u/\partial x \partial y \\ \partial^2 u/\partial x \partial y & \partial^2 u/\partial y^2 \end{pmatrix}$$

$$= \mathcal{R} \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \mathcal{R}^{-1},$$

where R is the eigenvectors matrix of D^2u and λ_i its eigenvalues (always real). Using this information, we introduce the following metric tensor $\mathcal{M}$:

$$\mathcal{M} = \mathcal{R} \begin{pmatrix} \tilde{\lambda}_1 & 0 \\ 0 & \tilde{\lambda}_2 \end{pmatrix} \mathcal{R}^{-1}, \tag{17}$$

where

$$\tilde{\lambda_i} = min(max(|\lambda_i|, \frac{1}{h^2_{max}}), \frac{1}{h^2_{min}}),$$

with h_{min} and h_{max} being the minimal and maximal edge lengths allowed in the mesh.

Now, if we generate, by a Delaunay procedure, an equilateral mesh with edges of length of 1 in the metric $\mathcal{M}/(c\mathcal{E})$, the interpolation error $\mathcal{E}$ is equi-distributed over the edges of length a_i if

$$\frac{1}{c\mathcal{E}} a_i^T M a_i = 1. \tag{18}$$

7.2.1 Three keys points

The previous definition is not sufficient for a suitable metric definition in the following configurations: 1) systems, 2) boundary layers, 3) multiple-scales phenomena.

Systems: For systems, the previous approach leads to a metric for each variable and we should take the intersection of all these metrics. More precisely, for two metrics, we find an approximation of their intersection by the following procedure:

Let λ_i^j and v_i^j, $i, j = 1, 2$ the eigen-values and eigen-vectors of $\mathcal{M}_j$, $j = 1, 2$. The intersection metric ($\hat{\mathcal{M}}$) is defined by

$$\hat{\mathcal{M}} = \frac{\hat{\mathcal{M}}_1 + \hat{\mathcal{M}}_2}{2}. \tag{19}$$

where $\hat{\mathcal{M}}_1$ (resp. $\hat{\mathcal{M}}_2$) has the same eigenvectors than $\mathcal{M}_1$, (resp. $\mathcal{M}_2$) but with eigenvalues defined by:

$$\tilde{\lambda}_i^1 = \max(\lambda_i^1, {v_i^1}^T \mathcal{M}_2 v_i^1), \quad i = 1, 2. \tag{20}$$

The the previous algorithm is easy to extend to the case of several variables. Here, one difficulty comes from the fact that we work with variables with different physical meaning and scale (for instance pressure, density and velocity). We will see that a relative rather than the global error estimation avoids this problem.

Boundary layers The computation of the Hessian is done by interpolation via the Green formula with Neumann boundary condition. For instance, given ρ, ρ_x is found in W_h, a finite dimensional approximation space. by solving approximately

$$\int_\Omega \rho_x w = -\int_\Omega \rho w_x \quad \forall w \in W_h$$

and then similarly for ρ_{xx}.

However, this does not lead to a suitable mesh for boundary layers. Indeed, the distance of the first layer of nodes to the wall will be quite irregular. Another important ingredient therefore, is a mixed Dirichlet-Neumann boundary condition for the different components of the metric on wall nodes for viscous computations. More precisely, the eigenvectors for these nodes are the normal and tangent unit vectors and the eigenvalue corresponding to the normal eigenvector is a prescribed value depending on the Reynolds number. The tangential eigenvalue comes from the metric of the solution.

More precisely, along the wall the previous metric $\mathcal{M}(x)$ is replaced by a new metric $\hat{\mathcal{M}}(x)$:

$$\hat{\mathcal{M}}(x) = T\Lambda T^{-1},$$

where

$$\Lambda = diag(\frac{1}{h_n^2}, \lambda_\tau) \quad \text{and} \quad T = (\vec{n}(x), \vec{\tau}(x)).$$

The refinement along the wall can now be monitored through h_n. This allows for instance for shocks and boundary layers to interact. This metric is propagated through the flow by a smoothing operator.

Multi-scale phenomena Difficulties appear when we try to compute multi-scale phenomena, such as turbulent flows by this approach. For instance, when we have several eddies with variable energy, it is difficult to capture the weaker ones, especially if there are shocks involved. We notice that (16) leads to a global error while we would like to have a relative one. We propose the following estimation which takes into account not only the dimension of the variables but also their magnitude:

$$\tilde{\mathcal{E}} = |\frac{u - \Pi_h u}{\max(|\Pi_h u|, \epsilon)}|_0 \leq ch^2 |\frac{D^2 u}{\max(|\Pi_h u|, \epsilon)}|_0, \tag{21}$$

where we have introduced the local value of the variable in the norm. ϵ is a cut-off to avoid numerical difficulties and also to define the difference between the orders of magnitude of the smallest and largest scales we try to capture. Indeed, when a phenomena falls below ϵ, it will not be captured. This is similar to look for a more precise estimation in regions where the variable is small. Another important consequence of this estimation is that it removes the dimensionality problems when intersecting metrics coming from different quantities.

More details on the ingredients used in the metric definition for inviscid and viscous laminar and turbulent flows involving shocks and boundary layers can be found in [26, 27, 28, 29].

7.3 Adaptive optimization algorithm

The following algorithms show how to couple shape optimization and mesh adaptation. To simplify the notations, the adaptive optimization algorithm is given with only one optimization after each adaptation, but several iterations of optimization can be made on the same mesh. The algorithms are presented with the simplest steepest descent method with projection. We will see that the definition of the projection operator is not an easy task.

At step i of adaptation, we denote the mesh, the solution, the metric and the cost $J(x_i, U(x_i))$ by $\mathcal{H}_i$, $\mathcal{S}_i$, $\mathcal{M}_i$ and $J(x_i)$.

Algorithm A1

$$\mathcal{H}_0, \mathcal{S}_0, \text{ given,}$$

Adaptation loop: DO $\quad i = 0, .., iadapt$

$$\text{define the control space (wall nodes) : } \mathcal{H}_i \rightarrow x_i,$$

$$\text{compute the gradient : } (x_i, \mathcal{H}_i, \mathcal{S}_i) \rightarrow \frac{dJ(x_i)}{dx_i},$$

$$\text{define the deformation: } \tilde{x}_i = P_i(x_i - \lambda \frac{dJ(x_i)}{dx_i}),$$

$$\text{deform the mesh : } (\tilde{x}_i, \mathcal{H}_i) \rightarrow \tilde{\mathcal{H}}_i,$$

$$\text{update the solution over the deformed mesh : } (\tilde{\mathcal{H}}_i, \mathcal{S}_i) \rightarrow \tilde{\mathcal{S}}_i,$$

$$\text{compute the metric : } (\tilde{\mathcal{H}}_i, \tilde{\mathcal{S}}_i) \rightarrow \mathcal{M}_i,$$

$$\text{generate the new mesh using this metric : } (\tilde{\mathcal{H}}_i, \mathcal{M}_i) \rightarrow \mathcal{H}_{i+1},$$

$$\text{interpolate the previous solution over the new mesh : } (\tilde{\mathcal{H}}_i, \tilde{\mathcal{S}}_i, \mathcal{H}_{i+1}) \rightarrow \overline{\mathcal{S}}_{i+1},$$

$$\text{compute the new solution over this mesh : } (\mathcal{H}_{i+1}, \overline{\mathcal{S}}_{i+1}) \rightarrow \mathcal{S}_{i+1},$$

END DO.

In the previous algorithm, the projection operator P_i changes after each adaptation as the control space changes. Therefore, the convergence issue is not clear. In the same way, adaption to more sophisticated pseudo-unsteady system based algorithm involving more than two shape parameterizations seems difficult. To avoid this problem, we define an exhaustive control space X suitable for a good description of any shape deformations. This can be associated to the first mesh used in the adaptation loop. In addition , this mesh has just to be refined in the

vicinity of the wall (and can be coarse elsewhere) An example of this is given is picture 10. We will see that we can also have just a surface mesh.

We then use the following algorithm:

Algorithm A2

$\mathcal{H}_0, \mathcal{S}_0, X$ given,

Adaptation loop: DO $\quad i = 0, .., iadapt$

identify the wall nodes: $\mathcal{H}_i \to x_i$,

compute the gradient : $(x_i, \mathcal{H}_i, \mathcal{S}_i) \to \dfrac{dJ(x_i)}{dx_i}$,

interpolate the gradient $\dfrac{dJ}{dX} = \Pi_i(\dfrac{dJ(x_i)}{dx_i})$,

define the deformation: $\tilde{X} = P(X - \lambda \dfrac{dJ}{dX})$,

interpolate back the deformation $\tilde{x}_i = \Pi_i^{-1}(\tilde{X})$,

the rest as in A1 :

deforme the mesh : $(\tilde{x}_i, \mathcal{H}_i) \to \tilde{\mathcal{H}}_i$,

update the solution over the deformed mesh : $(\tilde{\mathcal{H}}_i, \mathcal{S}_i) \to \tilde{\mathcal{S}}_i$,

compute the metric : $(\tilde{\mathcal{H}}_i, \tilde{\mathcal{S}}_i) \to \mathcal{M}_i$,

generate the new mesh using this metric : $(\tilde{\mathcal{H}}_i, \mathcal{M}_i) \to \mathcal{H}_{i+1}$,

interpolate the previous solution over the new mesh : $(\tilde{\mathcal{H}}_i, \tilde{\mathcal{S}}_i, \mathcal{H}_{i+1}) \to \overline{\mathcal{S}}_{i+1}$,

compute the new solution over this mesh : $(\mathcal{H}_{i+1}, \overline{\mathcal{S}}_{i+1}) \to \mathcal{S}_{i+1}$,

END DO.

Here the projection P is defined once for all, but the interpolation operator Π_i has to be redefined at each adaptation. However, we can follow the convergence of $|\frac{dJ}{dX}|$ as X is defined once for all.

Another interesting feature of algorithm A2 is that it allows adaptation within the optimization loop. Indeed, as we said, in A1 we made the adaptation outside the optimization loop to avoid differentiating the mesh generator. Another way to avoid this is to use our approximate gradient approach, where we need only informations over the shape. Therefore we can avoid taking into account dependencies created by the mesh generation tool inside the domain. However, we still need to keep the exhaustive control space X to be able to analyze the convergence of the algorithm. Hence, the last algorithm coupling all these ingredients is given by:

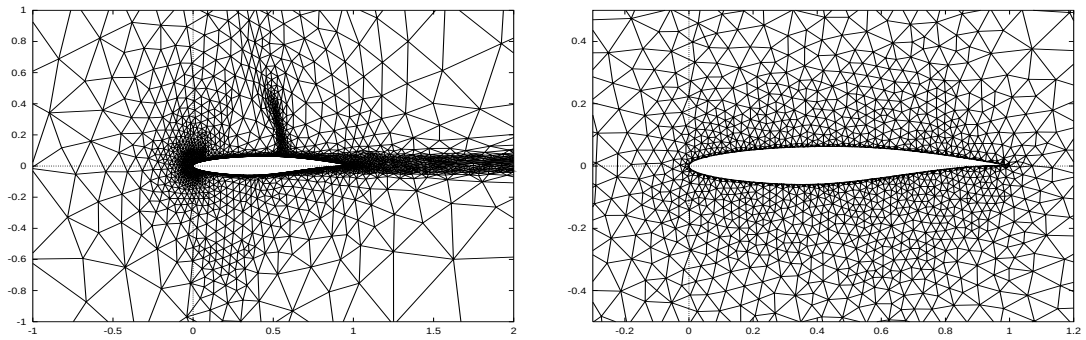

Figure 10: *The initial coarse mesh in the adaptation loop which is also used as background mesh to take into account the shape deformation. This is because we only need the shape, but we do not want to introduce an extra data structure. The exhaustive control space X is defined on this mesh at the beginning of the optimization loop. On the left we have an intermediate adapted mesh for a transonic turbulent drag reduction problem over a RAE 2822 profile.*

Algorithm A3

$\mathcal{H}_0, \mathcal{S}_0, X$ given,

Optimization loop : DO $\quad i = 0, .., ioptim$

identify the wall nodes: $\mathcal{H}_i \rightarrow x_i$,

compute the approximate gradient : $\quad (x_i, \partial\mathcal{H}_i, \partial\mathcal{S}_i) \rightarrow \dfrac{d\tilde{J}(x_i)}{dx_i}$,

interpolate the gradient: $\quad \dfrac{d\tilde{J}}{dX} = \Pi_i(\dfrac{d\tilde{J}(x_i)}{dx_i})$,

define the deformation: $\tilde{X} = P(X - \lambda\dfrac{d\tilde{J}}{dX})$,

interpolate back the deformation$\tilde{x}_i = \Pi_i^{-1}(\tilde{X})$,

deforme the mesh : $\quad (\tilde{x}_i, \mathcal{H}_i) \rightarrow \tilde{\mathcal{H}}_i$,

update the solution over the deformed mesh : $\quad (\tilde{\mathcal{H}}_i, \mathcal{S}_i) \rightarrow \tilde{\mathcal{S}}_i$,

Adaptation loop : DO $\quad j = 0, .., iadapt$

compute the metric : $(\tilde{\mathcal{H}}_j, \tilde{\mathcal{S}}_j) \rightarrow \mathcal{M}_j$,

generate the new mesh using this metric : $(\tilde{\mathcal{H}}_j, \mathcal{M}_j) \rightarrow \mathcal{H}_{j+1}$,

interpolate the previous solution over the new mesh : $(\tilde{\mathcal{H}}_j, \tilde{\mathcal{S}}_j, \mathcal{H}_{j+1}) \rightarrow \overline{\mathcal{S}}_{j+1}$,

compute the new solution over this mesh : $(\mathcal{H}_{j+1}, \overline{\mathcal{S}}_{j+1}) \rightarrow \mathcal{S}_{j+1}$,

END DO adaptation.

END DO optimization.

Where $\partial\mathcal{H}_i$ means mesh boundary informations and $\partial\mathcal{S}_i$) means the solution restricted to $\partial\mathcal{H}_i$.

Examples of the application of A2 and A3 are shown in the next chapters.

References

[1] B. Mohammadi (1997),*"Practical Applications to Fluid Flows of Automatic Differentiation for Design Problems"* , VKI lecture series, 1997-05.

[2] B. Mohammadi (1997), *"A New Optimal Shape Design Procedure for Inviscid and Viscous Turbulent Flows"*, Int. J. for Numerical Meth. in Fluid, vol. 25, 183-203.

[3] B. Mohammadi, N. Petruzzelli, *"Incomplete Sensitivities and BFGS Methods for 3D Aerodynamical Shape Design"*, Inria Report 3633.

[4] A. Dadone, B. Mohammadi and N. Petruzzelli, *"3D Aerodynamic Shape Design using Hessians based on Incomplete Sensitivities"*, IJCFD, 1999.

[5] G. Medic, B. Mohammadi and M. Stanciu, *"Global Plateform for Shape Optimization and Flow Control"*, CMAME, (submitted) 2000.

[6] G. Medic, B. Mohammadi, S. Moreau, (1998), *"Optimal Airfoil and Blade Design in Compressible and Incompressible Flows"*, AIAA-98-2898.

[7] J. Donea,*"An ALE Finite Element Method for Transient Fluid-Structure Interactions"*, Comp. Meth. App. Mech. Eng., (1982) num. 33, pp. 689-723.

[8] C. Farhat, M. Lesoinne, *"On the Accuracy, Stability and Performance of the Solution of Three-dimensional Nonlinear Transient Aeroelastic Problems by Partitioned Procedures"*, AIAA-96-1388.

[9] B. Nkonga, H. Guillard, *"Godunoc Type Method on Non-Structured Meshes for Three Dimensional Moving Boundary-Problems"*, Comp. Methods in Applied Mech. Eng. (1994), num. 113, pp:183-204.

[10] O. Pironneau (1984), *"Optimal Shape Design for Elliptic Systems"*, Springer-Verlag, New York.

[11] S. Piperno, C. Farhat, B. Larrouturou,*"Partitioned Procedures for the Transient Solution of Coupled Aeroelastic Problems"*, Comp. Meth. Appl. Mech. Eng. (2995), num. 124, pp. 97-101.

[12] H. Attouch, X. Goudou, P. Redont,*"The Heavy Ball with Friction Method"* , Advances in Contemporary Math., (1999).

[13] H. Attouch, R. Cominetti, (1996),*"A Dynamical Approach to Convex Minimization Coupling Approximation with the Steepest Descent Method"*, J. Differential Equations, 128 (2) pp. 519-540.

[14] H. Attouch, M. Cabot, M. Masmoudi, B. Mohammadi, P. Redont, (2000) *"Coupling Dynamic Approaches for Global Minimization"*, To appear Journal of Convex Analysis.

[15] B. Mohammadi (1999),*"Flow Control and Shape Optimization in Aeroelastic Configurations"*, AIAA 99-0182.

[16] F. Beux, A. Dervieux (1997),*"Multi-level Shape Optimization"* ,

[17] A. Marrocco, O. Pironneau (1987)*"Optimum design with Lagrangian Finite Element"*, Comp. Meth. Appl. Mech . Eng. 15-3.

[18] , G. Medic, B. Mohammadi, N. Petruzzelli, M. Stanciu, *"3D Optimal Shape Design for Complex Flows: Application to turbomachinery"*, AIAA-99-0833.

[19] B. Mohammadi and O. Pironneau, *"New Tools for Optimum Shape Design* , CFD Review, Special Issue, 1995.

[20] E. Laporte (1998), *Shape Optimization for Unsteady Configurations*, Thesis, Ecole Polytechnique, (1998).

[21] J.C. Gilbert, G. Le Vey, J. Masse (1991),*"La différentiation automatique de fonctions représentées par des programmes"*, Rapport de Recherche INRIA 1557.

[22] N. Rostaing-Schmidt (1993), *"Différentiation automatique: Application à un problème d'optimisation en météorologie"*, Ph.D. Thesis, University of Nice.

[23] C. Faure (1996), *"Splitting of Algebraic Expressions for Automatic Differentiation"*, In proc. of the Second SIAM Inter. Workshop on Computational Differentiation, Santa Fe.

[24] A. Griewank (1995),*"Achieving Logarithmic Growth of Temporal and Spatial Complexity in Reverse Automatic Differentiation,"* Optimization Methods and Software, Vol. 1, pp. 35-54.

[25] M. Stanciu and B. Mohammadi, (1999) *"Plateform for Multi-Model Design"*, Ercoftac journal.

[26] H. Borouchaki, M.J. Castro-Diaz, P.L. George, F. Hecht, B. Mohammadi (1996), *"Anisotropic adaptive mesh generation in two dimensions for CFD"*, 5th Inter. Conf. on Numerical Grid Generation in Computational Field Simulations, Mississipi State Univ.

[27] M. Castro-Diaz, F. Hecht and B. Mohammadi (1995), *"Anisotropic Grid Adaptation for Inviscid and Viscous Flows Simulations"*, IJNMF.

[28] H. Borouchaki, P.L.George, B. Mohammadi, (1996) *"Delaunay Mesh Generation Governed by Metric Specifications. Part II: Applications"*, Finite Element in Analysis and Design, special issue on mesh adaptation.

[29] F. Hecht, B. Mohammadi (1997),*"Mesh Adaptation by Metric Control for Multi-scale Phenomena and Turbulence"*, AIAA paper 97-0859.

[30] P.L. George, F. Hecht, E. Saltel (1990), *"Fully automatic mesh generator for 3d domains of any shape",* Impact of Comp. in Sci. and Eng., 2, pp.187-218.

[31] P.L. George (1991), *"Automatic mesh generation. Applications to finite element method"*, Wiley.