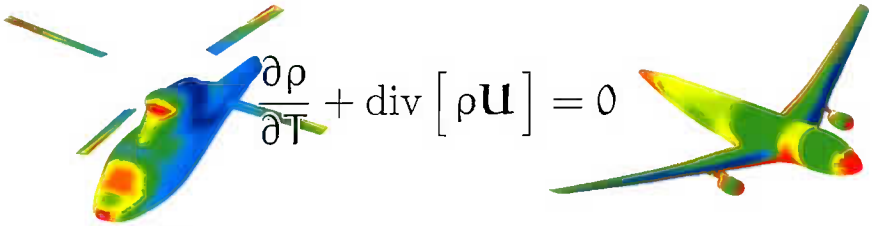
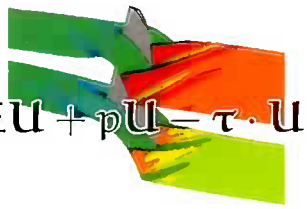
Theoretical Manual and User Manual for the *elsA_Opt* shape optimisation module



$$\frac{\partial \rho}{\partial T} + \mathrm{div}\left[\rho \mathbf{U}\right] = 0$$

$$\frac{\partial \rho \mathbf{U}}{\partial T} + \mathrm{div}\left[\rho \mathbf{U} \otimes \mathbf{U} + p\mathbf{I} - \boldsymbol{\tau}\right] = 0$$



$$\frac{\partial \rho E}{\partial T} + \mathrm{div}\left[\rho E \mathbf{U} + p\mathbf{U} - \boldsymbol{\tau} \cdot \mathbf{U} + \mathbf{q}\right] = 0$$

Quality	Author	For the reviewers	Approver
Function		Head of quality	
Name	P. Trontin, J.Peter, F. Renac	A.-M. Vuillot	J. Peter
Visa			

Software management : ELSA SCM
Applicability date : immediate
Diffusion : see last page

HISTORY

version edition	DATE	CAUSE and/or NATURE of EVOLUTION
1.0	December 18, 2009	First version.

CONTENTS

Empty page

Part I

THEORETICAL ELEMENTS

Empty page

1. OPTIMIZATION OF AERODYNAMIC SHAPES : ADJOINT VECTOR METHOD AND DIRECT DIFFERENTIATION METHOD

Contributor(s) : P. Trontin, J. Peter, F. Renac

1.1 Aerodynamic shape optimization using numerical simulation

1.1.1 Introduction

Aerodynamic shape optimization consists in seeking the shape, among a set of parameterized aerodynamic shapes, that significantly improves an objective function while satisfying a set of constraints. Some applications of this discipline are of considerable industrial importance. Of course, the most significant example is the aircraft drag reduction under constraints related to lift, moments and geometry.
A large range of numerical methods have been considered in the literature, essentially gradient, simplex, genetic and inverse methods. Among gradient-based methods, it is advisable to distinguish three approaches according to the type of gradient calculation of the objective and the constraints functions with respect to the design parameters : calculation by finite differences, calculation by the adjoint vector method, calculation by solving the linearized equations.
The purpose here is exclusively to describe the equations of two methods for calculating the gradients by differential calculations : the discrete adjoint method and the (discrete) direct differentiation method. These methods, used since the early 1990s, make it possible to calculate gradients without going through the calculation of as many flow fields as shape parameters, being different in this way from the old method of calculating the gradients by finite differences.
A strong modularity is required for the organization of gradient computations. The various sensivities are computed by three kinds of tools:

- *elsA* for the computation of the aerodynamic flow field, the solution of the adjoint or linearized equation and the actual computation of function gradients ;
- a software evaluating the objective and constraint functions and their derivatives with respect to the flow field (boundary and cell-centered values) and mesh coordinates. This software is FFD41 for civil aircraft applications, X-OPT for turbomachinery applications, FM-OPT for rotor applications ;
- a solid-surface and volume grid generation module. Most of the time a volume mesh corresponding to a new set of parameters is constructed by mesh deformation of the original volume mesh and/or the new and original solid-surface mesh. This module supplies the first two modules with the new volume meshes and also the derivatives of the grid coordinates with respect to the design parameters ;

This report is not intended to provide new theoretical elements relative to the grid-generation methods, the conditions of optimality under constraints or the algorithms of descent by gradient. Relevant literature about these topics can be found in [3]. Instead, we describe the function gradient computational methods and the inputs and outputs of the tools cited above.

1.1.2 Notations

Let us note α the vector of mesh control parameters. This vector describes the shape of the object around which a CFD simulation is carried out. The integer n_c indicates the dimension of α. The coordinates of the

grid are denoted X(α) ; we also assume that the derivative of the grid generation module, $dX/d\alpha$, is available. More precisely the coordinates of the mesh points of the solid surfaces, $S(\alpha)$, are an intermediate variable between α and $X(\alpha)$. Thus, we have :

$$\frac{dX}{d\alpha} = \frac{dX}{dS}\frac{dS}{d\alpha}$$

The vector of dimension n_a of flow field variables at cell centers is denoted W. The vector W_b denotes the values of the flow field at the boundaries of the domain. It is supposed that W_b is a function of the adjacent-cell aerodynamic variables and the mesh $X(\alpha)$: $W_b = W_b(W, X)$.
The discrete equations for fluid dynamics are noted [1]

$$R(W, X) = 0$$

(a set of n_a nonlinear equations with n_a unknowns). R is supposed to be C^k [2]. Under the assumption of the implicit function theorem $(\det\left[\frac{\partial R}{\partial W}(W(\alpha), X(\alpha))\right] \neq 0)$, this relation defines the flow as an implicit function of the grid, and consequently as a function of the design parameters. We can, from now on, note $W(\alpha)$ and rewrite the equations of fluid dynamics and their differentials :

$$R(W(\alpha), X(\alpha)) = 0$$

$$\frac{\partial R}{\partial W}\frac{dW}{d\alpha_i} = -\frac{\partial R}{\partial X}\frac{dX}{d\alpha_i} \qquad \frac{\partial R}{\partial W}\frac{dW}{d\alpha} = -\frac{\partial R}{\partial X}\frac{dX}{d\alpha}$$

(left: column vector equality and one linear system of size n_a, right: rectangular $n_a \times n_c$ matrix equality and n_c linear systems of size n_a)
$\mathcal{J}$ is the objective function :

$$\mathcal{J}(\alpha) = J(W_b(W(\alpha), X(\alpha)), W(\alpha), X(\alpha))$$

It will be seen that it is essential to use these dependences to make the derivatives products whose factors are calculated in different modules apparent. In the same way, the constraint functions $\mathcal{G}(\alpha)_k$ are expressed in the form $\mathcal{G}(\alpha)_k = g_k(W_b(W(\alpha), X(\alpha)), W(\alpha), X(\alpha))$. The integer n_t indicates the number of the constraints.
The vectors (R, W, λ adjoint vector...) are column vectors, the gradients of scalar functions with respect to vectors ($\frac{\partial J}{\partial X}$, $\frac{\partial J}{\partial W}$, $\frac{\partial g_k}{\partial W}$...) are line vectors.

1.2 Gradient computation methods for shape optimization.

1.2.1 Finite differences.

As a reminder, we briefly describe the finite difference method, which is as old as CFD and does not require any specific coding in a CFD software. A finite difference step size is chosen for each design parameter. Let it be denoted $\delta\alpha_i$. The mesh $X(\alpha)$ and all the meshes $X(\alpha+\delta\alpha_i)$ $i \in [1, n_c]$ are constructed. The corresponding

[1] We could have written $R(Wb, W, X) = 0$ and further $(\frac{\partial R}{\partial W} + \frac{\partial R}{\partial W}\frac{\partial W_b}{\partial W})\frac{dW}{d\alpha_i} = -(\frac{\partial R}{\partial X} + \frac{\partial R}{\partial W_b}\frac{\partial W_b}{\partial X})\frac{dX}{d\alpha_i}$. As all the derivatives of R are computed by elsA, this precision is not very essential. It is very different for the arguments of J because the derivatives of the function with respect to the three arguments - $\frac{\partial J}{\partial W}$, $\frac{\partial J}{\partial W_b}$ $\frac{\partial J}{\partial X}$ - are supplied to elsA by the objective/constraints module.

[2] Practically, R is only C^1

flows fields $W(\alpha)$, $W(\alpha+\delta\alpha_i)\ \ i \in [1, n_c]$ are computed. The function gradients are computed by following formula :

$$\frac{d\mathcal{J}(\alpha)}{d\alpha_i} \simeq \frac{J(W_b(W(\alpha+\delta\alpha_i), X(\alpha+\delta\alpha_i)), W(\alpha+\delta\alpha_i), X(\alpha+\delta\alpha_i)) - J(W_b(W(\alpha), X(\alpha)), W(\alpha), X(\alpha))}{\delta\alpha_i}$$

Consequently, this method requires (n_c+1) flow computations for first order accurate gradient evaluation and ($2 \times n_c$+1) flow computations for a second order one. Moreover the determination of the best values of $d\alpha_i$ is tricky when the residual of the space discretization for fluid dynamic equations cannot be driven to zero. Practically, the calculation of $X(\alpha+\delta\alpha_i)$ is replaced by the one of $X(\alpha)+\delta\alpha_i\frac{dX}{d\alpha}$. So, the use and the storage of $X(\alpha+\delta\alpha_i)$ and $\frac{dX}{d\alpha}$ simultaneously is avoided.

1.2.2 Calculation of the gradients by the (discrete) direct differentiation method.

Actually, the direct differentiation method has only been considered in its discrete form. Consequently, the term 'discrete' is not really necessary to qualify it. By differentiating the equations of fluid dynamics with respect to one of the design parameters α_i, we obtain :

$$\frac{\partial R}{\partial W}\frac{dW}{d\alpha_i} = -\frac{\partial R}{\partial X}\frac{dX}{d\alpha_i} \tag{1.1}$$

By solving these equations, we get all n_c column vectors of $\frac{dW}{d\alpha}$. We can then express the total derivative of the objective and of the constraints with respect to one design parameter (scalar equality) :

$$\frac{d\mathcal{J}}{d\alpha_i} = \frac{\partial J}{\partial X}\frac{dX}{d\alpha_i} + \frac{\partial J}{\partial W_b}\frac{dW_b}{dX}\frac{dX}{d\alpha_i} + (\frac{\partial J}{\partial W} + \frac{\partial J}{\partial W_b}\frac{dW_b}{dW})\frac{dW}{d\alpha_i}$$

$$\frac{d\mathcal{G}_k}{d\alpha_i} = \frac{\partial g_k}{\partial X}\frac{dX}{d\alpha_i} + \frac{\partial g_k}{\partial W_b}\frac{dW_b}{dX}\frac{dX}{d\alpha_i} + (\frac{\partial g_k}{\partial W} + \frac{\partial g_k}{\partial W_b}\frac{dW_b}{dW})\frac{dW}{d\alpha_i}$$

We can also express the total derivative of the functions with respect to the vector of design parameters (line vector equality, size n_c) :

$$\begin{aligned}
\nabla_\alpha\mathcal{J} &= \frac{\partial J}{\partial X}\frac{dX}{d\alpha} + \frac{\partial J}{\partial W_b}\frac{dW_b}{dX}\frac{dX}{d\alpha} + (\frac{\partial J}{\partial W} + \frac{\partial J}{\partial W_b}\frac{dW_b}{dW})\frac{dW}{d\alpha} \\
\nabla_\alpha\mathcal{G}_k &= \frac{\partial g_k}{\partial X}\frac{dX}{d\alpha} + \frac{\partial g_k}{\partial W_b}\frac{dW_b}{dX}\frac{dX}{d\alpha} + (\frac{\partial g_k}{\partial W} + \frac{\partial g_k}{\partial W_b}\frac{dW_b}{dW})\frac{dW}{d\alpha}
\end{aligned} \tag{1.2}$$

If we substitute the term $\frac{dW}{d\alpha}$ in the previous relations, the gradient of the objective and the constraints is written (line vector equality) :

$$\nabla_\alpha\mathcal{J} = \frac{\partial J}{\partial X}\frac{dX}{d\alpha} + \frac{\partial J}{\partial W_b}\frac{dW_b}{dX}\frac{dX}{d\alpha} - (\frac{\partial J}{\partial W} + \frac{\partial J}{\partial W_b}\frac{dW_b}{dW})\frac{\partial R}{\partial W}^{-1}\frac{\partial R}{\partial X}\frac{dX}{d\alpha}$$

$$\nabla_\alpha\mathcal{G}_k = \frac{\partial g_k}{\partial X}\frac{dX}{d\alpha} + \frac{\partial g_k}{\partial W_b}\frac{dW_b}{dX}\frac{dX}{d\alpha} - (\frac{\partial g_k}{\partial W} + \frac{\partial g_k}{\partial W_b}\frac{dW_b}{dW})\frac{\partial R}{\partial W}^{-1}\frac{\partial R}{\partial X}\frac{dX}{d\alpha}$$

1.2.3 Calculation of the gradients by the discrete adjoint method

The adjoint method has been considered in a discrete form (adjoint equation of discrete scheme) and a continuous form (discretization of the continuous adjoint equations of the continuous fluid dynamics equations). Only the equations of the discrete adjoint method is described here (see the description of the continuous adjoint method in the Appendix). We choose to introduce the equations of the adjoint method from the transposed of the previous relations, rather than from the differential of a Lagrangian. The transposition of the last two equations of the previous section leads to the relations (column vector equality) :

$$\nabla_\alpha \mathcal{J}(\alpha)^T = (\frac{\partial J}{\partial X}\frac{dX}{d\alpha})^T + (\frac{\partial J}{\partial W_b}\frac{dW_b}{dX}\frac{dX}{d\alpha})^T - (\frac{\partial R}{\partial X}\frac{dX}{d\alpha})^T(\frac{\partial R}{\partial W})^{-T}(\frac{\partial J}{\partial W} + \frac{\partial J}{\partial W_b}\frac{dW_b}{dW})^T$$

$$\nabla_\alpha \mathcal{G}_k(\alpha)^T = (\frac{\partial g_k}{\partial X}\frac{dX}{d\alpha})^T + (\frac{\partial g_k}{\partial W_b}\frac{dW_b}{dX}\frac{dX}{d\alpha})^T - (\frac{\partial R}{\partial X}\frac{dX}{d\alpha})^T(\frac{\partial R}{\partial W})^{-T}(\frac{\partial g_k}{\partial W} + \frac{\partial g_k}{\partial W_b}\frac{dW_b}{dW})^T$$

(the inverse of a transposed matrix is the transposed of the inverse matrix. That is why the notation A^{-T} is relevant). The adjoint vector λ for the calculation of the gradient of the objective is defined by :

$$(\frac{\partial R}{\partial W})^T\lambda = -(\frac{\partial J}{\partial W_b}\frac{dW_b}{dW} + \frac{\partial J}{\partial W})^T$$

By using the above formula for $\nabla_\alpha \mathcal{J}(\alpha)^T$, we can rewrite this quantity as follows (column vector equality) :

$$\nabla_\alpha \mathcal{J}(\alpha)^T = (\frac{\partial J}{\partial X}\frac{dX}{d\alpha})^T + (\frac{\partial J}{\partial W_b}\frac{dW_b}{dX}\frac{dX}{d\alpha})^T + (\frac{\partial R}{\partial X}\frac{dX}{d\alpha})^T\lambda$$

or for the line form :

$$\nabla_\alpha \mathcal{J}(\alpha) = \frac{\partial J}{\partial X}\frac{dX}{d\alpha} + \frac{\partial J}{\partial W_b}\frac{dW_b}{dX}\frac{dX}{d\alpha} + \lambda^T\frac{\partial R}{\partial X}\frac{dX}{d\alpha}$$

Note that we must solve a similar equation for each constraint function :

$$(\frac{\partial R}{\partial W})^T\lambda_k = -(\frac{\partial g_k}{\partial W_b}\frac{dW_b}{dW} + \frac{\partial g_k}{\partial W})^T$$

which leads to the expression of the gradient :

$$\nabla_\alpha \mathcal{G}_k = \frac{\partial g_k}{\partial X}\frac{dX}{d\alpha} + \frac{\partial g_k}{\partial W_b}\frac{dW_b}{dX}\frac{dX}{d\alpha} + \lambda_k^T\frac{\partial R}{\partial X}\frac{dX}{d\alpha}$$

Now, another method is presented to derive the gradients by the discrete adjoint method. A zero-term which corresponds to the product of the adjoint vector λ with the Eq. (1.1) is added to the equation (1.2).

$$\begin{aligned}
\nabla_\alpha \mathcal{J} &= \frac{\partial J}{\partial X}\frac{dX}{d\alpha} + \frac{\partial J}{\partial W_b}\frac{dW_b}{dX}\frac{dX}{d\alpha} + (\frac{\partial J}{\partial W} + \frac{\partial J}{\partial W_b}\frac{dW_b}{dW})\frac{dW}{d\alpha} + \lambda^T\left(\frac{\partial R}{\partial W}\frac{dW}{d\alpha} + \frac{\partial R}{\partial X}\frac{dX}{d\alpha}\right)\\
\nabla_\alpha \mathcal{G}_k &= \frac{\partial g_k}{\partial X}\frac{dX}{d\alpha} + \frac{\partial g_k}{\partial W_b}\frac{dW_b}{dX}\frac{dX}{d\alpha} + (\frac{\partial g_k}{\partial W} + \frac{\partial g_k}{\partial W_b}\frac{dW_b}{dW})\frac{dW}{d\alpha} + \lambda_k^T\left(\frac{\partial R}{\partial W}\frac{dW}{d\alpha} + \frac{\partial R}{\partial X}\frac{dX}{d\alpha}\right)
\end{aligned}$$

$$\begin{aligned}
\nabla_\alpha \mathcal{J} &= \frac{\partial J}{\partial X}\frac{dX}{d\alpha} + \frac{\partial J}{\partial W_b}\frac{dW_b}{dX}\frac{dX}{d\alpha} + (\frac{\partial J}{\partial W} + \frac{\partial J}{\partial W_b}\frac{dW_b}{dW} + \lambda^T\frac{\partial R}{\partial W})\frac{dW}{d\alpha} + \lambda^T\frac{\partial R}{\partial X}\frac{dX}{d\alpha})\\
\nabla_\alpha \mathcal{G}_k &= \frac{\partial g_k}{\partial X}\frac{dX}{d\alpha} + \frac{\partial g_k}{\partial W_b}\frac{dW_b}{dX}\frac{dX}{d\alpha} + (\frac{\partial g_k}{\partial W} + \frac{\partial g_k}{\partial W_b}\frac{dW_b}{dW} + \lambda_k^T\frac{\partial R}{\partial W})\frac{dW}{d\alpha} + \lambda_k^T\frac{\partial R}{\partial X}\frac{dX}{d\alpha})
\end{aligned}$$

λ and λ_k to eliminate the terms corresponding to $\frac{dW}{d\alpha}$, that is:

$$(\frac{\partial R}{\partial W})^T\lambda = -(\frac{\partial J}{\partial W_b}\frac{dW_b}{dW} + \frac{\partial J}{\partial W})^T$$

$$(\frac{\partial R}{\partial W})^T\lambda_k = -(\frac{\partial g_k}{\partial W_b}\frac{dW_b}{dW} + \frac{\partial g_k}{\partial W})^T$$

And we find the same expressions for $\nabla_\alpha \mathcal{J}(\alpha)$ and $\nabla_\alpha \mathcal{G}_k(\alpha)$ as before. This method has been employed for the calculation of the continuous adjoint equations (see annexes).

1.2.4 Discrete adjoint equations: derivation with respect to the coordinates of the grid X

With the classical gradient computation methods, the storage of the derivatives $\frac{dX}{d\alpha_i}$ for every control parameter α_i requires large memory resources. To avoid excessive memory usage, the derivatives of the discret equations R have to be calculated with respect to the mesh X, as well as the total derivatives of the objective function $\frac{dJ}{dX}$ and constraint functions $\frac{dG_k}{dX}$. The final product with $\frac{dX}{d\alpha_i}$ is then carried out by another computer with higher memory capacities and possibly lower computational speed.
Here, the derivative $(\frac{\partial R}{\partial X}\frac{dX}{d\alpha})$ is not estimated by finite differences, but as a product of differentials. Therefore, the gradients can be written as follows:

$$\nabla_\alpha \mathcal{J}(\alpha) = \frac{\partial J}{\partial X}\frac{dX}{d\alpha} + \frac{\partial J}{\partial W_b}\frac{\partial W_b}{\partial X}\frac{dX}{d\alpha} + \lambda^T(\frac{\partial R}{\partial X}\frac{dX}{d\alpha}) = (\frac{\partial J}{\partial X} + \frac{\partial J}{\partial W_b}\frac{\partial W_b}{\partial X} + \lambda^T\frac{\partial R}{\partial X})\frac{dX}{d\alpha}$$

$$\nabla_\alpha \mathcal{G}_k = \frac{\partial g_k}{\partial X}\frac{dX}{d\alpha} + \frac{\partial g_k}{\partial W_b}\frac{\partial W_b}{\partial X}\frac{dX}{d\alpha} + \lambda_k^T(\frac{\partial R}{\partial X}\frac{dX}{d\alpha}) = (\frac{\partial g_k}{\partial X} + \frac{\partial g_k}{\partial W_b}\frac{\partial W_b}{\partial X} + \lambda_k^T\frac{\partial R}{\partial X})\frac{dX}{d\alpha}$$

So, the derivatives of J and G_k with respect to X are given by:

$$\frac{dJ}{dX} = \frac{\partial J}{\partial X} + \frac{\partial J}{\partial W_b}\frac{\partial W_b}{\partial X} + \lambda^T\frac{\partial R}{\partial X}$$

$$\frac{dG_k}{dX} = \frac{\partial g_k}{\partial X} + \frac{\partial g_k}{\partial W_b}\frac{\partial W_b}{\partial X} + \lambda_k^T\frac{\partial R}{\partial X}$$

In practice, the calculation is carried out as follows:

$$\frac{dJ}{d\alpha_i} = \frac{dJ}{dX}\frac{dX}{dS}\frac{dS}{d\alpha_i} = (\frac{dJ}{dX}\frac{dX}{dS})\frac{dS}{d\alpha_i}$$

where S is the surface mesh. Most often, S is directly driven by α whereas the volume mesh X depends on S by some mesh deformation methods. This allows the use of several sets of parameters α. Indeed, the calculation of $\nabla_X \mathcal{J}$ or $\nabla_X \mathcal{G}_k$ is independent of α as explained previously.
This method cannot be applied to the context of the direct differentiation method. Indeed, the linearized equation is given by:

$$\frac{\partial R}{\partial W}\frac{\partial W}{\partial X} + \frac{\partial R}{\partial X} = 0$$

The difficulty comes from the storage of $\frac{\partial W}{\partial X}$ because of its huge size.

1.2.5 Relative cost of the adjoint vector method and direct differentiation method.

The classical analysis is based on the number of linear systems of size n_a to be solved. Let us note that for the two methods, the linear systems to be solved involve the same matrix. This complicates the effective calculation cost if a multiple right-hand side inversion method is considered.
These reservation being made, let us note that the adjoint method requires the inversion of (n_t+1) systems (the number of functions to be differentiated, one more than the number of constraints), while the direct differentiation method requires the solution of n_c systems (as many as control parameters).
In the industrial applications of aerodynamic shape optimization, the number of constraints is definitely lower than the number of shape parameters. The most effective method is thus the adjoint vector method.

1.3 Differentiation of aerodynamics functions w.r.t. numerical, physical and boundary condition parameters

In the computational framework of gradient computation for shape optimization, some other derivatives of interest may be computed, namely derivatives of aerodynamic functions w.r.t. numerical parameters (typically artificial dissipation coefficients), physical parameters (typically Reynolds number) or boundary conditions parameters (typically Mach number or angle of attack of the far field).

1.3.1 Differentiation w.r.t. numerical and physical parameters

Here, β is a numerical or a physical control parameter. Therefore the equation of fluid dynamics is given by :

$$R(W, X, \beta) = 0$$

If $\det\left[\dfrac{\partial R}{\partial W}\right] \neq 0$ and R is C^k, then, according to the implicit function theorem, we can find locally a C^k function ϕ so that $W = \phi(\beta, X)$ (written $W = W(\beta, X)$ hereafter). Therefore, the equation of fluid dynamics can be written :

$$R(W(X, \beta), X, \beta) = 0 \tag{1.3}$$

The objective and constraint equations are given by:

$$\mathcal{J}(\beta) = J(X, W(X, \beta), W_b(W(X, \beta), X), \beta)$$

$$\mathcal{G}_k(\beta) = g_k(X, W(X, \beta), W_b(W(X, \beta), X), \beta)$$

The explicit dependence of J w.r.t. β appears in the case of the Reynolds number influence on the viscous friction. The gradients are given by:

$$\nabla_\beta \mathcal{J} = \frac{\partial J}{\partial \beta} + (\frac{\partial J}{\partial W} + \frac{\partial J}{\partial W_b}\frac{\partial W_b}{\partial W})\frac{\partial W}{\partial \beta}$$

$$\nabla_\beta \mathcal{G}_k = \frac{\partial g_k}{\partial \beta} + (\frac{\partial g_k}{\partial W} + \frac{\partial g_k}{\partial W_b}\frac{\partial W_b}{\partial W})\frac{\partial W}{\partial \beta}$$

From Eq. (1.3), the linearized equation is written:

$$\frac{\partial R}{\partial W}\frac{\partial W}{\partial \beta} + \frac{\partial R}{\partial \beta} = 0 \tag{1.4}$$

In the context of the direct differentiation method, Eq. (1.4) is solved to find $\frac{\partial W}{\partial \beta}$, and then the gradients can be written:

$$\nabla_\beta \mathcal{J} = \frac{\partial J}{\partial \beta} - (\frac{\partial J}{\partial W_b}\frac{\partial W_b}{\partial W} + \frac{\partial J}{\partial W}) \left(\frac{\partial R}{\partial W}\right)^{-1} \frac{\partial R}{\partial \beta}$$

$$\nabla_\beta \mathcal{G}_k = \frac{\partial g_k}{\partial \beta} - (\frac{\partial g_k}{\partial W_b}\frac{\partial W_b}{\partial W} + \frac{\partial g_k}{\partial W}) \left(\frac{\partial R}{\partial W}\right)^{-1} \frac{\partial R}{\partial \beta}$$

In the context of the discrete adjoint method, a zero term composed of the product of λ^T by Eq. (1.4) is added to the functions $\nabla_\beta \mathcal{J}$ and $\nabla_\beta \mathcal{G}_k$:

$$\nabla_\beta \mathcal{J} = \frac{\partial J}{\partial \beta} + (\frac{\partial J}{\partial W} + \frac{\partial J}{\partial W_b}\frac{\partial W_b}{\partial W})\frac{\partial W}{\partial \beta} + \lambda^T(\frac{\partial R}{\partial W}\frac{\partial W}{\partial \beta} + \frac{\partial R}{\partial \beta})$$

$$\nabla_\beta \mathcal{G}_k = \frac{\partial g_k}{\partial \beta} + (\frac{\partial g_k}{\partial W} + \frac{\partial g_k}{\partial W_b}\frac{\partial W_b}{\partial W})\frac{\partial W}{\partial \beta} + \lambda_k^T(\frac{\partial R}{\partial W}\frac{\partial W}{\partial \beta} + \frac{\partial R}{\partial \beta})$$

Terms involving $\frac{\partial W}{\partial \beta}$ are eliminated so that the equations for adjoint vectors λ and λ_k are given by:

$$\frac{\partial J}{\partial W} + \frac{\partial J}{\partial W_b}\frac{\partial W_b}{\partial W} + \lambda^T \frac{\partial R}{\partial W} = 0$$

$$\frac{\partial g_k}{\partial W} + \frac{\partial g_k}{\partial W_b}\frac{\partial W_b}{\partial W} + \lambda_k^T \frac{\partial R}{\partial W} = 0$$

Finally, the gradients of the objective and constraint functions are given by:

$$\nabla_\beta \mathcal{J} = \frac{\partial J}{\partial \beta} + \lambda^T \frac{\partial R}{\partial \beta}$$

$$\nabla_\beta \mathcal{G}_k = \frac{\partial g_k}{\partial \beta} + \lambda_k^T \frac{\partial R}{\partial \beta}$$

1.3.2 Differentiation w.r.t. boundary condition parameters

For a clearer presentation, it is necessary here to write the explicit dependence of R w.r.t. W_b. Boundary condition data are control parameters here. They are written δ.

$$R(W_b(W, X, \delta), W, X) = 0 \tag{1.5}$$

For the objective and constraint functions, we have:

$$\mathcal{J}(\delta) = J(W_b(W, X, \delta), W, X)$$

$$\mathcal{G}_k(\delta) = g_k(W_b(W, X, \delta), W, X)$$

If $\det \left[\frac{\partial R}{\partial W_b}\frac{\partial W_b}{\partial W} + \frac{\partial R}{\partial W}\right] \neq 0$, then W can be written as a fuction of δ. Thus, the objective and constraint functions are given by:

$$\mathcal{J}(\delta) = J(W_b(W(\delta), X, \delta), W(\delta), X)$$

$$\mathcal{G}_k(\delta) = g_k(W_b(W(\delta), X, \delta), W(\delta), X)$$

and their gradients by:

$$\begin{aligned}\nabla_\delta \mathcal{J} &= \frac{\partial J}{\partial W_b}\frac{\partial W_b}{\partial \delta} + (\frac{\partial J}{\partial W} + \frac{\partial J}{\partial W_b}\frac{\partial W_b}{\partial W})\frac{\partial W}{\partial \delta} \\ \nabla_\delta \mathcal{G}_k &= \frac{\partial g_k}{\partial W_b}\frac{\partial W_b}{\partial \delta} + (\frac{\partial g_k}{\partial W} + \frac{\partial g_k}{\partial W_b}\frac{\partial W_b}{\partial W})\frac{\partial W}{\partial \delta}\end{aligned} \tag{1.6}$$

From Eq. (1.5), the linearized equation is written:

$$\left[\frac{\partial R}{\partial W} + \frac{\partial R}{\partial W_b}\frac{\partial W_b}{\partial W}\right]\frac{\partial W}{\partial \delta} + \frac{\partial R}{\partial W_b}\frac{\partial W_b}{\partial \delta} = 0 \tag{1.7}$$

In the context of the direct differentiation method, Eq. (1.7) is solved to find $\frac{\partial W}{\partial \delta}$. The gradients computed from Eq. (1.6) can also be expressed as:

$$\nabla_\delta \mathcal{J} = \frac{\partial J}{\partial W_b}\frac{\partial W_b}{\partial \delta} - (\frac{\partial J}{\partial W} + \frac{\partial J}{\partial W_b}\frac{\partial W_b}{\partial W})\left[\frac{\partial R}{\partial W} + \frac{\partial R}{\partial W_b}\frac{\partial W_b}{\partial W}\right]^{-1}\frac{\partial R}{\partial W_b}\frac{\partial W_b}{\partial \delta}$$

$$\nabla_\delta \mathcal{G}_k = \frac{\partial g_k}{\partial W_b}\frac{\partial W_b}{\partial \delta} - (\frac{\partial g_k}{\partial W} + \frac{\partial g_k}{\partial W_b}\frac{\partial W_b}{\partial W})\left[\frac{\partial R}{\partial W} + \frac{\partial R}{\partial W_b}\frac{\partial W_b}{\partial W}\right]^{-1}\frac{\partial R}{\partial W_b}\frac{\partial W_b}{\partial \delta}$$

In the context of the discrete adjoint method, a zero term composed of the product of λ^T by Eq. (1.7) is added to the functions $\nabla_\delta \mathcal{J}$ and $\nabla_\delta \mathcal{G}_k$:

$$\nabla_\delta \mathcal{J} = \frac{\partial J}{\partial W_b}\frac{\partial W_b}{\partial \delta} + (\frac{\partial J}{\partial W} + \frac{\partial J}{\partial W_b}\frac{\partial W_b}{\partial W})\frac{\partial W}{\partial \delta} + \lambda^T\left(\left[\frac{\partial R}{\partial W} + \frac{\partial R}{\partial W_b}\frac{\partial W_b}{\partial W}\right]\frac{\partial W}{\partial \delta} + \frac{\partial R}{\partial W_b}\frac{\partial W_b}{\partial \delta}\right)$$

$$\nabla_\delta \mathcal{G}_k = \frac{\partial g_k}{\partial W_b}\frac{\partial W_b}{\partial \delta} + (\frac{\partial g_k}{\partial W} + \frac{\partial g_k}{\partial W_b}\frac{\partial W_b}{\partial W})\frac{\partial W}{\partial \delta} + \lambda_k^T\left(\left[\frac{\partial R}{\partial W} + \frac{\partial R}{\partial W_b}\frac{\partial W_b}{\partial W}\right]\frac{\partial W}{\partial \delta} + \frac{\partial R}{\partial W_b}\frac{\partial W_b}{\partial \delta}\right)$$

Terms involving $\frac{\partial W}{\partial \delta}$ are cancelled so that the equations for the adjoint vectors λ and λ_k are given by:

$$\frac{\partial J}{\partial W} + \frac{\partial J}{\partial W_b}\frac{\partial W_b}{\partial W} + \lambda^T\left(\frac{\partial R}{\partial W} + \frac{\partial R}{\partial W_b}\frac{\partial W_b}{\partial W}\right) = 0$$

$$\frac{\partial g_k}{\partial W} + \frac{\partial g_k}{\partial W_b}\frac{\partial W_b}{\partial W} + \lambda_k^T\left(\frac{\partial R}{\partial W} + \frac{\partial R}{\partial W_b}\frac{\partial W_b}{\partial W}\right) = 0$$

Finally, the gradients of the objective and constraint functions are given by:

$$\nabla_\delta \mathcal{J} = \frac{\partial J}{\partial W_b}\frac{\partial W_b}{\partial \delta} + \lambda^T\frac{\partial R}{\partial W_b}\frac{\partial W_b}{\partial \delta}$$

$$\nabla_\delta \mathcal{G}_k = \frac{\partial g_k}{\partial W_b}\frac{\partial W_b}{\partial \delta} + \lambda_k^T\frac{\partial R}{\partial W_b}\frac{\partial W_b}{\partial \delta}$$

1.4 Differentiated systems of equations. Solution of the adjoint and direct equations.

1.4.1 Differentiated systems of equations. Classical "frozen-μ_t" assumption for RANS simulations

When dealing with inviscid flows and Euler equations, the equations of the previous sections are exactly the one solved in $elsA$. The discrete equations for fluid dynamics $R(W,X)=0$ are a set of n_a coupled non-linear equations and n_a is equal to five times the number of cells of the mesh (as $elsA$ is a cell-centred finite volume code using 3D-volumes even for 2D problems).

When dealing with turbulent flows modeled by RANS equations, several situations are possible :
• A transport-equations turbulence model is considered.
In this case $R(W,X)=0$ is a set of n_a coupled non-linear equations, n_a being six (Spalart-Almaras model) or seven (two transport-equation models) times the number of cells of the mesh. The (almost) exact linearization of all the equations is feasible, but the resulting matrix is extremely hard to invert. Hence, for almost all gradient computations for (RANS) flows, the so-called "frozen-μ_t assumption" is done. This means that only the first five equations of $R(W,X)=0$ are linearized with respect to the mean-flow conservative variables, assuming that μ_t is constant at the center of the cells when the shape changes. In this case the jacobian that is inverted could be be noted $\frac{\partial R_5}{\partial W_5}$ for the sake of clarity ; nevertheless, for the sake of simplicity, it will still be noted $\frac{\partial R}{\partial W}$. Obviously the "frozen-$\mu_t$ assumption" induces inaccuracy in the values of the gradient of the aerodynamic functions with respect to the design parameters. It is observed that this bias is larger at lower Reynolds numbers and when the shape parameters affect the solid shape near a shock.
Besides, the discrete flux and source terms of two specific turbulence models have been linearized - the Wilcox (k-ω) model and the Launder-Sharma variant of the $k-\epsilon$ model. For each linearization accurate gradients (compared to "frozen-μ_t" assumption) were computed for a 2D flow. At present, the ill-conditioning of the jacobian matrix when involving the linearization of the turbulence model restricts the actual usage of this option to 2D flows.

• An algebraic turbulence model is considered.
In particular, Michel et al. turbulence model is still currently used for turbomachinery computations. Just as before, the "frozen-μ_t" assumption can be used to enhance robustness of the jacobian inversion.
Besides, Michel et al. turbulence model was differentiated by C.T. Phâm during his PhD thesis.

1.4.2 Iterative solution of the linear systems

The jacobian matrix $(\frac{\partial R}{\partial W})$ (as defined in the previous section) and its transpose are large, sparse, multi-banded matrices. Thus, their inverse can not be computed by a direct method, at least for large 2D and 3D problems. Some king of iterative strategy (conjugate gradient method, relaxation) has to be implemented. A classical strategy consists in solving the linear system using a Newton-type or relaxation algorithm. An approximate jacobian, noted $(\frac{\partial R}{\partial W})^{(APP)}$ appears on the left hand-side of the algorithm equation. This matrix can be equal or very similar to the approximate jacobian used as implicit matrix for steady state computations with backward-Euler schemes. On the right-hand side of the algorithm equation is the term that has to be driven to zero. The true jacobian $(\frac{\partial R}{\partial W})$ appears in that right-part of the equation. When considering a complicated set of equations (RANS equations for exemple), it can by replaced by an accurate approximation noted $(\frac{\partial R}{\partial W})^{(ACC)}$. The equation for the adjoint method is then :

$$\frac{\partial R}{\partial W}^{T(APP)} \left(\lambda^{(l+1)} - \lambda^{(l)}\right) = -\left(\frac{\partial R}{\partial W}^{T(ACC)} \lambda^{(l)} + (\frac{\partial J}{\partial W_b}\frac{dW_b}{dW} + \frac{\partial J}{\partial W})^T\right)$$

and for the method of the direct differentiation method :

$$\frac{\partial R}{\partial W}^{(APP)} \left((\frac{dW}{d\alpha})^{(l+1)} - (\frac{dW}{d\alpha})^{(l)}\right) = -\left(\frac{\partial R}{\partial W}^{(ACC)} \frac{dW}{d\alpha}^{(l)} + (\frac{\partial R}{\partial X}\frac{dX}{d\alpha})_{(fd)}\right)$$

(l) being the iteration index of Newton-relaxation method.

1.4.3 Recursive projection method

We consider the family of iterations for inverting a linear system $\mathbf{Ax} = \mathbf{b}$ of the form

$$\mathbf{Mx}^{(l+1)} = \mathbf{Nx}^{(l)} + \mathbf{b} \tag{1.8}$$

where the matrices $\mathbf{M}$ and $\mathbf{N}$ define a splitting of the matrix $\mathbf{A} = \mathbf{M} - \mathbf{N}$. For instance, the backward Euler scheme (see section 1.4.2) for the discrete direct differentiation method reads

$$\mathbf{A} = \frac{\partial \mathbf{R}}{\partial \mathbf{W}}^{(ACC)}, \quad \mathbf{x} = \frac{d\mathbf{W}}{d\alpha}, \quad \mathbf{b} = -\left(\frac{\partial \mathbf{R}}{\partial \mathbf{X}}\frac{d\mathbf{X}}{d\alpha}\right)^{(fd)},$$

and the implicit operator $\mathbf{M}$ is defined by

$$\mathbf{M} = \frac{\partial \mathbf{R}}{\partial \mathbf{W}}^{(APP)}.$$

Whether or not the iteration (1.8) converges to the solution $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ depends on the spectral radius of the iteration matrix

$$\mathbf{\Phi} = \mathbf{I} - \mathbf{M}^{-1}\mathbf{A}. \tag{1.9}$$

We next introduce introduce the Recursive Projection Method from Shroff and Keller[4] for the stabilization of unstable recurcive fixed point iteration procedures.

Stabilization procedure

Let $\mathbf{A}, \mathbf{\Phi} \in \mathbb{R}^{N\times N}$ where N represents the size of the linear system to be solved. Suppose the iteration (1.8) diverges thus implying that there are $m \geq 1$ eigenvalues of $\mathbf{\Phi}$ with a modulus greater than unity :

$$|\lambda_1| \geq \cdots \geq |\lambda_m| \geq 1. \tag{1.10}$$

Define the subspace $\mathbb{P} = \text{span}\{\mathbf{e_1}, \ldots, \mathbf{e_m}\}$ spaned by the eigenvectors associated to the eigenvalues λ_i, $i = 1, \ldots, m$, and $\mathbb{Q} = \mathbb{P}^\perp$ its orthogonal supplement in $\mathbb{R}^N$. These subspaces define the unique decomposition:

$$\forall \mathbf{x} \in \mathbb{R}^N, \quad \exists!(\mathbf{x}_p, \mathbf{x}_q) \in \mathbb{P}\times\mathbb{Q} : \quad \mathbf{x} = \mathbf{x}_p + \mathbf{x}_q.$$

The orthogonal projectors onto the subspaces $\mathbb{P}$ and $\mathbb{Q}$ are denoted $\mathbf{P}$ and $\mathbf{Q}$ respectively. Let $\mathbf{V} \in \mathbb{R}^{N\times m}$ be a matrix whose columns consitute an orthonormal basis for $\mathbb{P}$, then the projectors are defined by

$$\mathbf{P} = \mathbf{VV}^T, \quad \mathbf{Q} = \mathbf{I} - \mathbf{VV}^T.$$

Note that since $\mathbb{P}$ is an invariant subspace of $\mathbf{\Phi}$, we have

$$\mathbf{Q\Phi P} = 0. \tag{1.11}$$

The RPM iteration reads

$$\begin{aligned}
\mathbf{x}_q^{(l+1)} &= (\mathbf{I}-\mathbf{P})\mathbf{F}(\mathbf{x}_p^{(l)}+\mathbf{x}_q^{(l)}),\\
\mathbf{x}_p^{(l+1)} &= \mathbf{x}_p^{(l)} + (\mathbf{I}-\mathbf{P\Phi P})^{-1}(\mathbf{PF}(\mathbf{x}_p^{(l)}+\mathbf{x}_q^{(l)}) - \mathbf{x}_p^{(l)}),\\
\mathbf{x}^{(l+1)} &= \mathbf{x}_p^{(l+1)} + \mathbf{x}_q^{(l+1)},
\end{aligned} \tag{1.12}$$

with $\mathbf{x}_p^{(0)} = \mathbf{P}\mathbf{x}^{(0)}$ and $\mathbf{x}_q^{(0)} = (\mathbf{I}-\mathbf{P})\mathbf{x}^{(0)}$. According to Shroff and Keller, the RPM iteration (1.12) converges even when the original iteration (1.8) diverges.
The implementation of the scheme (1.12) requires the construction of the projectors $\mathbf{P}$ and $\mathbf{Q}$ and hence the evaluation of the orthonormal basis $\mathbf{V}$. In the next sections, we will present the method for computing $\mathbf{V}$.

Construction of the basis $\mathbf{V}$

Consider the vector $\mathbf{v}_1 = \Delta\mathbf{x}_q^{(l-k+1)}$ where $\Delta\mathbf{x}_q^{(j)} = \mathbf{x}_q^{(j+1)} - \mathbf{x}_q^{(j)}$ and the matrix $\hat{\mathbf{A}} = \mathbf{Q\Phi Q}$. Define the Krylov subspace of dimension k generated by $\mathbf{v}_1$ and $\hat{\mathbf{A}}$:

$$\mathbb{K}_k = \text{Vect}\{\mathbf{v}_1, \hat{\mathbf{A}}\mathbf{v}_1, \ldots, \hat{\mathbf{A}}^{k-1}\mathbf{v}_1\}.$$

According to (1.12a), we have $\Delta\mathbf{x}_q^{(j)} = \hat{\mathbf{A}}^j\Delta\mathbf{x}_q^{(0)}$. Let $\mathbf{K}_k = (\mathbf{v}_1, \hat{\mathbf{A}}\mathbf{v}_1, \ldots, \hat{\mathbf{A}}^{k-1}\mathbf{v}_1)$ be a matrix whose columns span the subspace $\mathbb{K}_k$. Compute the QR factorization

$$\mathbf{K}_k = \mathbf{Q}_k\mathbf{R}_k$$

where the columns of $\mathbf{Q}_k$ form an orthonormal basis of $\mathbb{K}_k$ and where the absolute values of the diagonal elements of the upper triangular matrix $\mathbf{R}_k$ are sorted in a decreasing order.
If iteration (1.8) fails to converge, vectors for $\mathbf{V}$ are chosen on the basis of the following condition. For the largest $1 \leq j \leq k-1$ such that

$$\left|\frac{R_{j,j}}{R_{j+1,j+1}}\right| > \kappa,$$

the first j columns of $\mathbf{Q}_k$ are added to $\mathbf{V}$:

$$\mathbf{V} \leftarrow (\mathbf{V}, \mathbf{Q}_j).$$

The vectors $\mathbf{Q}_j$ are normalized by construction and lie in $\mathbb{Q}$ so they are orthogonal to the previous basis $\mathbf{V}$. Hence, further orthogonalization is not necessary.
Note that the parameter $\kappa > 1$ stands for the Krylov acceptance ratio and its inverse represents a bound for the residual of the computed eigenspace of $\hat{\mathbf{A}}$.

Decreasing size of the basis $\mathbf{V}$

It is important that equation (1.11) holds for the success of the RPM, thus requiring that $\mathbb{P}$ remains an invariant subspace of $\mathbf{\Phi}$. This motivates a procedure for selecting eigemodes that do not fulfill criterion (1.10). For that purpose, let $\mathbf{H} = \mathbf{V}^T\mathbf{\Phi}\mathbf{V} \in \mathbb{R}^{m\times m}$ and let λ be an eigenvalue of $\mathbf{H}$ and $\mathbf{y}$ its associated eigenvector. Then, we have

$$\mathbf{V}^T(\mathbf{P\Phi P x} - \lambda\mathbf{x}) = 0$$

where $\mathbf{x} = \mathbf{V}\mathbf{y}$ represents an approximate eigenvector of $\mathbf{P\Phi P}$. The matrix $\mathbf{H}$ is of small size and its eigenanalysis is cheap, it is done by using the Lapack routine DGEEV. When only $\hat{m} < m$ eigenvalues of $\mathbf{H}$ are outside the disk $K_\delta = \{\mathbf{z} \in \mathbb{C} : |\mathbf{z}| < 1 - \delta\}$, with $0 \leq \delta \leq 1$, we compute a new basis $\hat{\mathbf{V}}$ from the $\hat{m}$ associated eigenvectors for $\mathbf{H}$. Then, the linear span of $\mathbf{V}\hat{\mathbf{V}}$ is a good approximation of the new eigenbasis and we use its Gram-Schmidt orthogonalization to replace the old basis:

$$\mathbf{V} \leftarrow \texttt{Gram\_Schmidt}(\mathbf{V}\hat{\mathbf{V}}).$$

1.4.4 Differentiated fluxes. Implicit matrices.

Differentiated fluxes

So far, we have supposed that the space discretization residual is a differentiable function of the grid and the aerodynamic flow field. It is natural, in order to avoid any differentiation problem, to choose a differentiable inviscid fluid flux formula. Nevertheless, flux formulas that are not differentiable on a limited domain (typically involving an absolute value), do not lead to big issues. The following flux formulas may be used in the analysis computation ; they will be differentiated with the appropriate parameters :
• Mean flow.
- Roe flux. Either its basic order one version or its extension to second order using the MUSCL formula with Van Albada limiting function ;
- Centered flux (either skew-symmetric or divergence form) plus scalar artificial dissipation (Jameson et al. 1981). Only a limited number of options and boundary treatments of the scalar dissipation are differentiated ;
- Viscous flux with a gradient at cell-centers corrected at interfaces (so called $5p\_cor$ formula)
All these formula are differentiated with respect flow field and geometry. They may hence be used in all gradient computation mode (adjoint direct mode with parameters and total derivation w.r.t mesh nodes)
• Turbulence quantities transport equations.
- Roe order one flux (no linearization of the VTD extension) ;
- Viscous flux with shord stencil gradient computation ($3p$ and $5p\_cor$ formula) ;
- Source term (no option proposed for analysis computation) of the differentiated models.
As explained before the full linearization of (RANS)+turbulence model equations is rarely used. It is only available mode with parameters ($k-\omega$ Wilcox adjoint and direct,$k-\epsilon$ Launder-Sharma direct).

Definition of the exact and approximate jacobian

The implicit matrices $\dfrac{\partial R}{\partial W}^{(APP)}$ are those defined for steady state computations using backward-Euler schemes : the matrix version of LU-RELAX or LU-SSOR methods, which are descibed in details in this Manual. When using the adjoint vector method, the exact transpose of these matrices is considered. In order to give an exhaustive description of approximate jacobian matrices, it is to be added that multiplying the convective terms of LU-RELAX matrix or LU-SSOR matrix by a factor $\frac{3}{2}$ is an option, which is often used when second order MUSCL upwind flux is considered for the analysis calculation.

1.5 Inputs and outputs of the tools involved.

The optimization chain is not described in details, but the inputs and outputs of the different modules for the gradient computation are described here more precisely.

1.5.1 Grid generator module

A solid-surface and volume grid generator module is needed. Most of the time the volume mesh $X(\alpha^1)$ corresponding to a new set of parameter α_1 is constructed by mesh deformation from the original volume mesh, $X(\alpha^0)$, and the new and original solid-surface mesh - $S(\alpha^1)$, $S(\alpha^0)$. So, this module computes by some method the new volume meshes $X(\alpha^1)$ from α^1. This mesh is an input of the aerodynamic/gradient module and the objective and constraints module. The module also has to compute the derivatives $\frac{dX}{d\alpha_i}(\alpha^1)$ (differentiation of mesh with respect to the design parameters). These derivatives are inputs of *elsA* computations.

1.5.2 Objective and constraints module.

This module computes function values $\mathcal{J}(\alpha), \mathcal{G}_k(\alpha)$ and their derivatives with respect to the flow field and mesh - $\frac{\partial J}{\partial W}, \frac{\partial J}{\partial W_b}, \frac{\partial J}{\partial X}, \frac{\partial g_k}{\partial W}, \frac{\partial g_k}{\partial W_b}, \frac{\partial g_k}{\partial X}$ - from the aerodynamic flow field $W(\alpha)$ - supplied by *elsA* and the mesh $X(\alpha)$ supplied by the grid generation module. The derivatives of the functions with respect to W_b, W and X are supplied to *elsA*. This software is FFD41 for civil aircraft applications, X-OPT for turbomachinery applications and FM-OPT for rotor applications.

1.5.3 elsA

The software computes the flow field $W(\alpha)$ corresponding to a new mesh $X(\alpha)$. It uses the sensitivies of the mesh - $\frac{dX}{d\alpha_i}$ and the sensivities of the objective constraints and objective - $\frac{\partial J}{\partial W}, \frac{\partial J}{\partial W_b}, \frac{\partial J}{\partial X}, \frac{\partial g_k}{\partial W}, \frac{\partial g_k}{\partial W_b}, \frac{\partial g_k}{\partial X}$ - to solve the adjoint equation and to gather all the gradient terms of the adjoint or differentiation method. The function gradients are supplied to the optimization module.

1.6 Annexes

1.6.1 Continuous adjoint equations

In this approach the adjoint of the governing equations in continuous form with respect to a given objective function is derived, before being discretized. A complete description can be found in Giles and Pierce [2]. To present the method, the Euler equations in body-fitted coordinates in two dimensions are considered. They can be written using Cartesian coordinates:

$$\frac{\partial w}{\partial t} + \frac{\partial f(w)}{\partial x} + \frac{\partial g(w)}{\partial y} = 0 \tag{1.13}$$

where

$$w = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{pmatrix} \qquad f(w) = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho H u \end{pmatrix} \qquad g(w) = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho H v \end{pmatrix}$$

It is assumed that the problem in physical space with a body-fitted structured grid, can be transformed into computational coordinates (ξ, η), see Figure 1.1, in such a way that the transformation of D_{xy} to $D_{\xi\eta}$ is direct,

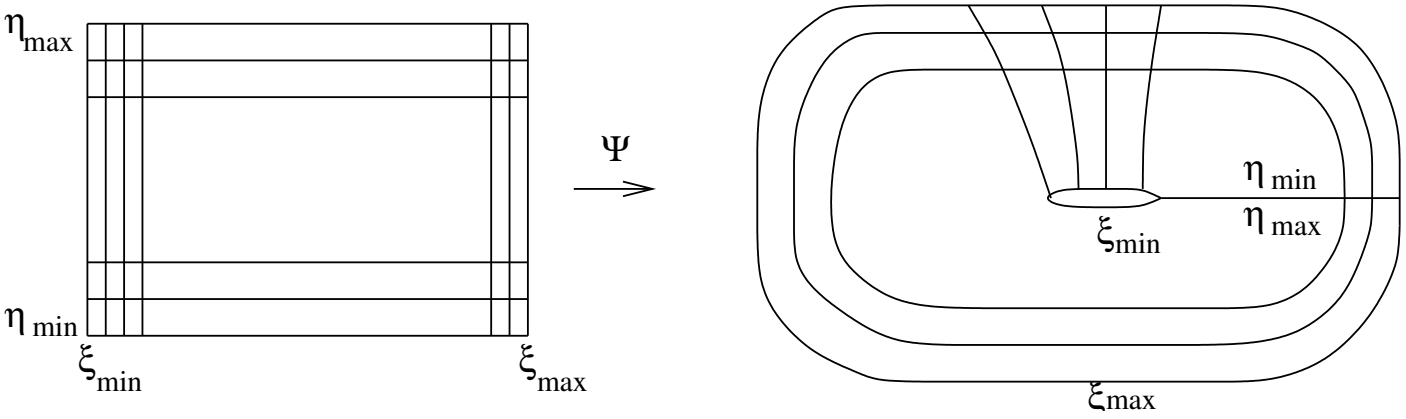


Figure 1.1: Body-fitted to computational coordinate domain transformation.

that $D_{\xi\eta}$ is a rectangular domain $[\xi_{min}, \xi_{max}] \times [\eta_{min}, \eta_{max}]$ and that ξ_{min} corresponds to the profile surface. Let K be the determinant of the Jacobian of the coordinate transformation:

$$K(\xi,\eta) = \det \begin{pmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{pmatrix}$$

Equation (1.13) can be written:

$$\frac{\partial K w}{\partial t} + \frac{\partial}{\partial \xi}\left(f\frac{\partial y}{\partial \eta} - g\frac{\partial x}{\partial \eta}\right) + \frac{\partial}{\partial \eta}\left(-f\frac{\partial y}{\partial \xi} + g\frac{\partial x}{\partial \xi}\right) = 0 \tag{1.14}$$

The aerodynamic field and the fluxes in the computational coordinates are given by:

$$\begin{pmatrix} U \\ V \end{pmatrix} = \frac{1}{K}\begin{pmatrix} \frac{\partial y}{\partial \eta} & -\frac{\partial x}{\partial \eta} \\ -\frac{\partial y}{\partial \xi} & \frac{\partial x}{\partial \xi} \end{pmatrix}\begin{pmatrix} u \\ v \end{pmatrix}$$

$$W = K\begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{pmatrix} \quad F(W) = K\begin{pmatrix} \rho U \\ \rho U u + p\frac{\partial \xi}{\partial x} \\ \rho U v + p\frac{\partial \xi}{\partial y} \\ \rho U H \end{pmatrix} \quad G(W) = K\begin{pmatrix} \rho V \\ \rho V u + p\frac{\partial \eta}{\partial x} \\ \rho V v + p\frac{\partial \eta}{\partial y} \\ \rho V H \end{pmatrix}.$$

Then, the Euler equations in the computational coordinates are given by:

$$\frac{\partial F(W)}{\partial \xi} + \frac{\partial G(W)}{\partial \eta} = 0, \tag{1.15}$$

The continuous adjoint equations can now be derived as follows: first write the first variation of the flow equations with respect to the design parameter α. Referring to Eq. (1.14) there are two distinct types of variation: the flux terms $f(w)$ and $g(w)$ vary with α, because the flow changes in the transformed coordinate space when the shape changes, and independently all metric terms also depend on α:

$$f(w) \to f(w) + \frac{\partial f}{\partial w}\frac{dw}{d\alpha}\delta\alpha, \quad \frac{\partial x}{\partial \eta} \to \frac{\partial x}{\partial \eta} + \frac{\partial^2 x}{\partial \eta \partial \alpha}\delta\alpha.$$

If the flux Jacobians $a(w) = \mathrm{d}f(w)/\mathrm{d}w$ and $b(w) = \mathrm{d}g(w)/\mathrm{d}w$ are introduced, then the linearized equation corresponding to (1.14) is:

$$\frac{\partial}{\partial\xi}\left\{\left(a(w)\frac{\partial y}{\partial\eta}-b(w)\frac{\partial x}{\partial\eta}\right)\frac{dw}{d\alpha}\right\} \quad +\frac{\partial}{\partial\eta}\left\{\left(-a(w)\frac{\partial y}{\partial\xi}+b(w)\frac{\partial x}{\partial\xi}\right)\frac{dw}{d\alpha}\right\}$$
$$+\frac{\partial}{\partial\xi}\left\{f(w)\frac{\partial^2 y}{\partial\eta\partial\alpha}-g(w)\frac{\partial^2 x}{\partial\eta\partial\alpha}\right\} \quad +\frac{\partial}{\partial\eta}\left\{-f(w)\frac{\partial^2 y}{\partial\xi\partial\alpha}+g(w)\frac{\partial^2 x}{\partial\xi\partial\alpha}\right\}=0, \tag{1.16}$$

The objective function formulated in the new coordinate system is:

$$\mathcal{J}(\alpha)=\int_{\xi_{min}} J_1(w)\frac{\partial y}{\partial\eta}\,d\eta+\int_{D_{\xi\eta}} J_2(w)K(\xi,\eta)\,d\xi\,d\eta, \tag{1.17}$$

where the domain of integration is now independent of α. Then:

$$\begin{aligned}\frac{d\mathcal{J}}{d\alpha}(\alpha) &= \int_{\xi_{min}}\left(\frac{dJ_1(w)}{dw}\frac{dw}{d\alpha}\frac{\partial y}{\partial\eta}+J_1(w)\frac{\partial^2 y}{\partial\eta\partial\alpha}\right)d\eta \\ &\quad+\int_{D_{\xi\eta}}\left(\frac{dJ_2(w)}{dw}\frac{dw}{d\alpha}K(\xi,\eta)+J_2(w)\frac{\partial K(\xi,\eta)}{\partial\alpha}\right)d\xi\,d\eta.\end{aligned} \tag{1.18}$$

Let a_1 and a_2 be the flux Jacobian in the computational mesh directions ξ and η.

$$a_1(w,\xi,\eta)=\left(a(w)\frac{\partial y}{\partial\eta}-b(w)\frac{\partial x}{\partial\eta}\right),\quad a_2(w,\xi,\eta)=\left(-a(w)\frac{\partial y}{\partial\xi}+b(w)\frac{\partial x}{\partial\xi}\right). \tag{1.19}$$

The idea behind the following procedure is to add to Eq. (1.18) the inner product of the linearized governing equations with an arbitrary four-component Lagrange multiplier λ, analogously to the discrete case. Then we search for a condition on λ for the gradient to be independent of the $\mathrm{d}w/\mathrm{d}\alpha$ terms. In this case we assume that the flow and adjoint solutions, w and λ, are once continuously differentiable with respect to the computational coordinates, i.e. $w,\lambda\in C^1(D_{\xi\eta})^4$. Note that this is a very different restriction to that necessary in the discrete case. Proceeding from Eq. (1.16) we have:

$$\int_{D_{\xi\eta}}\lambda^T\left\{\frac{\partial}{\partial\xi}\left(a_1(w,\xi,\eta)\frac{dw}{d\alpha}\right)+\frac{\partial}{\partial\eta}\left(a_2(w,\xi,\eta)\frac{dw}{d\alpha}\right)\right\}d\xi\,d\eta \quad +$$
$$\int_{D_{\xi\eta}}\lambda^T\left\{\frac{\partial}{\partial\xi}\left(f(w)\frac{\partial^2 y}{\partial\eta\partial\alpha}-g(w)\frac{\partial^2 x}{\partial\eta\partial\alpha}\right)+\frac{\partial}{\partial\eta}\left(-f(w)\frac{\partial^2 y}{\partial\xi\partial\alpha}+g(w)\frac{\partial^2 x}{\partial\xi\partial\alpha}\right)\right\}d\xi\,d\eta \quad =0.$$

Using integration by parts, and the fact that the flow sensitivity and coordinate derivatives such as $\partial^2 y/\partial\xi\partial\alpha$ are taken to be zero at the farfield we have:

$$-\int_{D_{\xi\eta}}\frac{\partial\lambda^T}{\partial\xi}a_1(w,\xi,\eta)\frac{dw}{d\alpha}\,d\xi\,d\eta-\int_{D_{\xi\eta}}\frac{\partial\lambda^T}{\partial\eta}a_2(w,\xi,\eta)\frac{dw}{d\alpha}\,d\xi\,d\eta$$
$$-\int_{D_{\xi\eta}}\frac{\partial\lambda^T}{\partial\xi}\left(f(w)\frac{\partial^2 y}{\partial\eta\partial\alpha}-g(w)\frac{\partial^2 x}{\partial\eta\partial\alpha}\right)-\frac{\partial\lambda^T}{\partial\eta}\left(-f(w)\frac{\partial^2 y}{\partial\xi\partial\alpha}+g(w)\frac{\partial^2 x}{\partial\xi\partial\alpha}\right)d\xi\,d\eta$$
$$+\int_{\xi_{min}}\lambda^T a_1(w,\xi,\eta)\frac{dw}{d\alpha}\,d\eta+\int_{\xi_{min}}\lambda^T\left(f(w)\frac{\partial^2 y}{\partial\eta\partial\alpha}-g(w)\frac{\partial^2 x}{\partial\eta\partial\alpha}\right)d\eta \quad =0.$$

Adding this expression to (1.18) and extracting terms multiplying $dw/d\alpha$ we obtain, from the volume and surface integrals respectively:

$$\left\{ \begin{array}{rl} \frac{dJ_2(w)}{dw} K(\xi,\eta) - \frac{\partial \lambda^T}{\partial \xi} a_1(w,\xi,\eta) - \frac{\partial \lambda^T}{\partial \eta} a_2(w,\xi,\eta) & = 0, \quad \text{on } D_{\xi,\eta}, \\ \lambda^T a_1(w,\xi,\eta) + \frac{dJ_1(w)}{dw} \frac{\partial y}{\partial \eta} & = 0, \quad \text{on } \xi_{min}, \end{array} \right. \tag{1.20}$$

the *continuous adjoint* equations and boundary conditions. Finally, the gradient of $\mathcal{J}$ may be written:

$$\begin{aligned} \frac{d\mathcal{J}(d\alpha)}{d\alpha} = & \int_{\xi_{min}} J_1(w) \frac{\partial^2 y}{\partial \eta \partial \alpha} \, d\eta + \int_{\xi_{min}} \lambda^T \left(f(w) \frac{\partial^2 y}{\partial \eta \partial \alpha} - g(w) \frac{\partial^2 x}{\partial \eta \partial \alpha} \right) d\eta \\ & - \int_{D_{\xi\eta}} \frac{\partial \lambda^T}{\partial \xi} \left(f(w) \frac{\partial^2 y}{\partial \eta \partial \alpha} - g(w) \frac{\partial^2 x}{\partial \eta \partial \alpha} \right) d\xi \, d\eta \\ & - \int_{D_{\xi\eta}} \frac{\partial \lambda^T}{\partial \eta} \left(-f(w) \frac{\partial^2 y}{\partial \xi \partial \alpha} + g(w) \frac{\partial^2 x}{\partial \xi \partial \alpha} \right) d\xi \, d\eta \\ & + \int_{D_{\xi\eta}} J_2(w) \frac{\partial K(\xi,\eta)}{\partial \alpha} \, d\xi \, d\eta. \end{aligned}$$

1.6.2 Second-order dervivatives using discrete methods

Second-derivatives of $\mathcal{J}$ are presented now. Whereas there are only two possibilities between direct or adjoint methods at first order, there are here four possibilities, corresponding to the two methods of solving for the first and second derivatives. The convention of denoting them DD.DD, DD.AV, AV.DD and AV.AV is followed where DD stands for direct differentiation and AV adjoint vector.

Because of the complexity of the vector algebra we introduce i, j, to index degrees of freedom of the flow solution, m, n those of the mesh, and a, b to index entries in the objective function and design variable vectors, and use summation convention. Then:

$$\frac{\mathrm{d}\mathcal{J}_c}{\mathrm{d}\alpha_a} = \frac{\partial J_c}{\partial X_m} \frac{\mathrm{d}X_m}{\mathrm{d}\alpha_a} + \frac{\partial J_c}{\partial W_i} \frac{\mathrm{d}W_i}{\mathrm{d}\alpha_a},$$

and second derivatives are

$$\begin{aligned} \frac{\mathrm{d}^2\mathcal{J}_c}{\mathrm{d}\alpha_a \mathrm{d}\alpha_b} = & \frac{\partial^2 J_c}{\partial X_m \partial X_n} \frac{\mathrm{d}X_m}{\mathrm{d}\alpha_a} \frac{\mathrm{d}X_n}{\mathrm{d}\alpha_b} + \frac{\partial J_c}{\partial X_m} \frac{\mathrm{d}^2 X_m}{\mathrm{d}\alpha_a \mathrm{d}\alpha_b} \\ & + \frac{\partial^2 J_c}{\partial W_i \partial X_m} \left(\frac{\mathrm{d}X_m}{\mathrm{d}\alpha_a} \frac{\mathrm{d}W_i}{\mathrm{d}\alpha_b} + \frac{\mathrm{d}X_m}{\mathrm{d}\alpha_b} \frac{\mathrm{d}W_i}{\mathrm{d}\alpha_a} \right) \\ & + \frac{\partial^2 J_c}{\partial W_i \partial W_j} \frac{\mathrm{d}W_i}{\mathrm{d}\alpha_a} \frac{\mathrm{d}W_j}{\mathrm{d}\alpha_b} + \frac{\partial J_c}{\partial W_i} \frac{\mathrm{d}^2 W_i}{\mathrm{d}\alpha_a \mathrm{d}\alpha_b}, \end{aligned} \tag{1.21}$$

so that there are then $n_\alpha(n_\alpha + 1)/2$ second-order flow derivatives $\mathrm{d}W/\mathrm{d}\alpha\mathrm{d}\alpha$ required to evaluate the second-derivative tensor.

DD.DD

Given the linearized equation:

$$\frac{\partial R}{\partial W}\frac{dW}{d\alpha} = -\frac{\partial R}{\partial X}\frac{dX}{d\alpha} \tag{1.22}$$

The most direct approach is to solve Eq. (1.22) for each $\mathrm{d}W/\mathrm{d}\alpha$ term, and derive the corresponding second-order equation by differentiating (1.22) with respect to α:

$$\left(\frac{\partial R}{\partial W_i}\right)\frac{\mathrm{d}W_i}{\mathrm{d}\alpha_a} = -\frac{\partial R}{\partial X_m}\frac{\mathrm{d}X_m}{\mathrm{d}\alpha_a}, \tag{1.23}$$

$$\begin{aligned}\left(\frac{\partial R}{\partial W_i}\right)\frac{\mathrm{d}^2W_i}{\mathrm{d}\alpha_a\mathrm{d}\alpha_b} &= -\frac{\partial^2 R}{\partial X_m\partial X_n}\frac{\mathrm{d}X_m}{\mathrm{d}\alpha_a}\frac{\mathrm{d}X_n}{\mathrm{d}\alpha_b} - \frac{\partial R}{\partial X_m}\frac{\mathrm{d}^2X_m}{\mathrm{d}\alpha_a\mathrm{d}\alpha_b} \\ &\quad -\frac{\partial^2 R}{\partial X_m\partial W_i}\left(\frac{\mathrm{d}X_m}{\mathrm{d}\alpha_a}\frac{\mathrm{d}W_i}{\mathrm{d}\alpha_b}+\frac{\mathrm{d}X_m}{\mathrm{d}\alpha_b}\frac{\mathrm{d}W_i}{\mathrm{d}\alpha_a}\right) - \frac{\partial^2 R}{\partial W_i\partial W_j}\frac{\mathrm{d}W_i}{\mathrm{d}\alpha_a}\frac{\mathrm{d}W_j}{\mathrm{d}\alpha_b},\end{aligned} \tag{1.24}$$

giving a total of $\frac{n_\alpha(n_\alpha+1)}{2}+n_\alpha$ linear systems to be solved. Once the flow derivatives found, they may be substituted into (1.21) to obtain the desired gradient.

DD.AV

The second method consists in adding to (1.21) the product of equation (1.24) by an arbitrary adjoint vector Λ in a similar procedure to that presented in the section devoted to the first-order gradient computation. Λ is then chosen to eliminate the second order flow sensitivity $\mathrm{d}W/\mathrm{d}\alpha_a\mathrm{d}\alpha_b$:

$$\begin{aligned}\frac{\mathrm{d}^2\mathcal{J}}{\mathrm{d}\alpha_a\mathrm{d}\alpha_b} &= \frac{\partial^2 J}{\partial X_m\partial X_n}\frac{\mathrm{d}X_m}{\mathrm{d}\alpha_a}\frac{\mathrm{d}X_n}{\mathrm{d}\alpha_b} + \frac{\partial J}{\partial X_m}\frac{\mathrm{d}^2X_m}{\mathrm{d}\alpha_a\mathrm{d}\alpha_b} \\ &\quad +\frac{\partial^2 J}{\partial W_i\partial X_m}\left(\frac{\mathrm{d}X_m}{\mathrm{d}\alpha_a}\frac{\mathrm{d}W_i}{\mathrm{d}\alpha_b}+\frac{\mathrm{d}X_m}{\mathrm{d}\alpha_b}\frac{\mathrm{d}W_i}{\mathrm{d}\alpha_a}\right) + \frac{\partial^2 J}{\partial W_i\partial W_j}\frac{\mathrm{d}W_i}{\mathrm{d}\alpha_a}\frac{\mathrm{d}W_j}{\mathrm{d}\alpha_b} \\ &\quad +\frac{\partial J}{\partial W_i}\frac{\mathrm{d}^2W_i}{\mathrm{d}\alpha_a\mathrm{d}\alpha_b} + \Lambda^T\frac{\partial R}{\partial W_i}\frac{\mathrm{d}^2W_i}{\mathrm{d}\alpha_a\mathrm{d}\alpha_b} \\ &\quad +\Lambda^T\frac{\partial^2 R}{\partial X_m\partial X_n}\frac{\mathrm{d}X_m}{\mathrm{d}\alpha_a}\frac{\mathrm{d}X_n}{\mathrm{d}\alpha_b} + \Lambda^T\frac{\partial R}{\partial X_m}\frac{\mathrm{d}^2X_m}{\mathrm{d}\alpha_a\mathrm{d}\alpha_b} \\ &\quad +\Lambda^T\frac{\partial^2 R}{\partial X_m\partial W_i}\left(\frac{\mathrm{d}X_m}{\mathrm{d}\alpha_a}\frac{\mathrm{d}W_i}{\mathrm{d}\alpha_b}+\frac{\mathrm{d}X_m}{\mathrm{d}\alpha_b}\frac{\mathrm{d}W_i}{\mathrm{d}\alpha_a}\right) + \Lambda^T\frac{\partial^2 R}{\partial W_i\partial W_j}\frac{\mathrm{d}W_i}{\mathrm{d}\alpha_a}\frac{\mathrm{d}W_j}{\mathrm{d}\alpha_b}.\end{aligned}$$

The adjoint equation to be solved turns out to be the same as for the first order gradient computation,

$$\frac{\partial R}{\partial W}^T\Lambda = -\frac{\partial J}{\partial W}^T,$$

where Λ is independent of α again. Having solved the n_α direct equations for the $\mathrm{d}W/\mathrm{d}\alpha$ terms, and $n_{\mathcal{J}}$ adjoint equations for the Λ, i.e. $n_\alpha+n_{\mathcal{J}}$ linear systems, the derivative tensor is:

$$\begin{aligned}\frac{\mathrm{d}^2\mathcal{J}}{\mathrm{d}\alpha_a\mathrm{d}\alpha_b} &= \frac{\partial^2 J}{\partial X_m \partial X_n}\frac{\mathrm{d}X_m}{\mathrm{d}\alpha_a}\frac{\mathrm{d}X_n}{\mathrm{d}\alpha_b} + \frac{\partial J}{\partial X_m}\frac{\mathrm{d}^2 X_m}{\mathrm{d}\alpha_a\mathrm{d}\alpha_b} \\ &+ \frac{\partial^2 J}{\partial W_i \partial X_m}\left(\frac{\mathrm{d}X_m}{\mathrm{d}\alpha_a}\frac{\mathrm{d}W_i}{\mathrm{d}\alpha_b} + \frac{\mathrm{d}X_m}{\mathrm{d}\alpha_b}\frac{\mathrm{d}W_i}{\mathrm{d}\alpha_a}\right) + \frac{\partial^2 J}{\partial W_i \partial W_j}\frac{\mathrm{d}W_i}{\mathrm{d}\alpha_a}\frac{\mathrm{d}W_j}{\mathrm{d}\alpha_b} \\ &+ \Lambda^T\frac{\partial^2 R}{\partial X_m \partial X_n}\frac{\mathrm{d}X_m}{\mathrm{d}\alpha_a}\frac{\mathrm{d}X_n}{\mathrm{d}\alpha_b} + \Lambda^T\frac{\partial R}{\partial X_m}\frac{\mathrm{d}^2 X_m}{\mathrm{d}\alpha_a\mathrm{d}\alpha_b} \\ &+ \Lambda^T\frac{\partial^2 R}{\partial X_m \partial W_i}\left(\frac{\mathrm{d}X_m}{\mathrm{d}\alpha_a}\frac{\mathrm{d}W_i}{\mathrm{d}\alpha_b} + \frac{\mathrm{d}X_m}{\mathrm{d}\alpha_b}\frac{\mathrm{d}W_i}{\mathrm{d}\alpha_a}\right) + \Lambda^T\frac{\partial^2 R}{\partial W_i \partial W_j}\frac{\mathrm{d}W_i}{\mathrm{d}\alpha_a}\frac{\mathrm{d}W_j}{\mathrm{d}\alpha_b}\end{aligned} \tag{1.25}$$

where no $\mathrm{d}W/\mathrm{d}\alpha\mathrm{d}\alpha$ terms appear.

AV.DD

The starting point is the familiar adjoint expression

$$\frac{\mathrm{d}J}{\mathrm{d}\alpha_a} = \frac{\partial J}{\partial X}\frac{\mathrm{d}X}{\mathrm{d}\alpha_a} + \Lambda^T\frac{\partial R}{\partial X}\frac{\mathrm{d}X}{\mathrm{d}\alpha_a}, \qquad \frac{\partial R}{\partial W}^T\Lambda = -\frac{\partial J}{\partial W}^T,$$

which defines Λ as a function of $W(\alpha)$ and $X(\alpha)$, and hence α. Differentiating both expressions with respect to a design parameter α_b therefore gives:

$$\begin{aligned}\frac{\mathrm{d}^2 J}{\mathrm{d}\alpha_a\mathrm{d}\alpha_b} &= \frac{\partial^2 J}{\partial X_i \partial X_j}\frac{\mathrm{d}X_i}{\mathrm{d}\alpha_a}\frac{\mathrm{d}X_j}{\mathrm{d}\alpha_b} + \frac{\partial^2 J}{\partial X_i \partial W_j}\frac{\mathrm{d}X_i}{\mathrm{d}\alpha_a}\frac{\mathrm{d}W_j}{\mathrm{d}\alpha_b} + \frac{\partial J}{\partial X_i}\frac{\mathrm{d}^2 X_i}{\mathrm{d}\alpha_a\mathrm{d}\alpha_b} \\ &+ \left(\frac{\mathrm{d}\Lambda^T}{\mathrm{d}\alpha_b}\right)\frac{\partial R}{\partial X_i}\frac{\mathrm{d}X_i}{\mathrm{d}\alpha_a} \\ &+ \Lambda^T\left(\frac{\partial^2 R}{\partial X_i \partial X_j}\frac{\mathrm{d}X_i}{\mathrm{d}\alpha_a}\frac{\mathrm{d}X_j}{\mathrm{d}\alpha_b} + \frac{\partial^2 R}{\partial X_i \partial W_j}\frac{\mathrm{d}X_i}{\mathrm{d}\alpha_a}\frac{\mathrm{d}W_j}{\mathrm{d}\alpha_b} + \frac{\partial R}{\partial X_i}\frac{\mathrm{d}^2 X_i}{\mathrm{d}\alpha_a\mathrm{d}\alpha_b}\right)\end{aligned} \tag{1.26}$$

and

$$\begin{aligned}\frac{\partial R_i}{\partial W_l}\frac{\mathrm{d}\Lambda_i}{\mathrm{d}\alpha_b} &= -\frac{\partial^2 R_i}{\partial W_l \partial W_j}\frac{\mathrm{d}W_j}{\mathrm{d}\alpha_b}\Lambda_i - \frac{\partial^2 R_i}{\partial W_l \partial X_j}\frac{\mathrm{d}X_j}{\mathrm{d}\alpha_b}\Lambda_i \\ &- \frac{\partial^2 J}{\partial W_l \partial W_j}\frac{\mathrm{d}W_j}{\mathrm{d}\alpha_b} - \frac{\partial^2 J}{\partial W_l \partial X_j}\frac{\mathrm{d}X_j}{\mathrm{d}\alpha_b}.\end{aligned} \tag{1.27}$$

Therefore n_α flow derivatives are required, $n_\mathcal{J}$ adjoint solutions, and $n_\mathcal{J} \times n_\alpha$ adjoint derivatives, giving $n_\alpha + n_\mathcal{J} + n_\alpha n_\mathcal{J}$ linear problems to be solved.

AV.AV

A second adjoint variable Γ is introduced. The product of Γ with (1.26) is added to (1.27), allowing the elimination of the derivatives of the first adjoint variable Λ, if Γ satisfies

$$\frac{\partial R}{\partial W}\Gamma = -\frac{\partial R}{\partial X}\frac{\partial X}{\partial \alpha}.$$

But this is exactly the direct equation, so $\Gamma = \mathrm{d}W/\mathrm{d}\alpha$, and the expression for the derivative reduces to that of DD.AV (1.25). Again only the original $n_\alpha + n_\mathcal{J}$ systems must be solved, as for DD.AV.

1.6.3 Non-stationary approach

The time interval $[0, T]$ is divided into n_f intervals of the same size Δt such as $[0, T] = \bigcup_{i=0}^{n_f-1}[t_i, t_{i+1}]$. The flow field is supposed to be known at $t = 0$. Time integration is performed with implicit non linear schemes.

$$\frac{1}{\Delta t}[W^1_{i,j}V^1_{i,j}] - \frac{1}{\Delta t}[W^0_{i,j}V^0_{i,j}] + R(W^1_{i,j}) = 0 \tag{1.28}$$

$$\frac{3}{2\Delta t}[W^n_{i,j}V^n_{i,j}] - \frac{2}{\Delta t}[W^{n-1}_{i,j}V^{n-1}_{i,j}] + \frac{1}{2\Delta t}[W^{n-2}_{i,j}V^{n-2}_{i,j}] + R(W^n_{i,j}) = 0 \quad n \in [2, n_f] \tag{1.29}$$

An adjoint vector λ^n, $n \in [1, n_f]$ is associated to every equation. $\lambda^n_{i,j}$ is the value of λ^n in the (i, j) cell. The control is characterized by n_f vectors $(\alpha^1, ..., \alpha^{n_f})$. The components of the vector α represent boundary condition or geometric parameters. Therefore, α influences the calculation of R and V. As a first approach, it is supposed that every α^k is a scalar which represents a boundary condition. The application of the implicit function theorem leads to:

$$W^1 = W^1(\alpha^1)\ \ W^2 = W^2(\alpha^2, \alpha^1)\ ...\ \ W^j = W^j(\alpha^j, ..., \alpha^2, \alpha^1)$$

The objective function is given by:

$$\mathcal{J} = \sum_{n=1}^{n=n_f} J(W^n, \alpha^n)$$

The differential of $\mathcal{J}$ is:

$$d\mathcal{J} = \sum_{n=1}^{n=n_f}[\frac{\partial J}{\partial W}(W^n, \alpha^n)dW^n + \frac{\partial J}{\partial \alpha^n}d\alpha^n]$$

which can be rewritten as a function of the variation of the control:

$$d\mathcal{J} = \sum_{n=1}^{n=n_f}[\frac{\partial J}{\partial W}(W^n, \alpha^n)(\sum_{l=1}^{l=n}\frac{\partial W^n}{\partial \alpha^l}d\alpha^l) + \frac{\partial J}{\partial \alpha^n}d\alpha^n] \tag{1.30}$$

When the size of every vector α^i is n_p, we have:

$$d\mathcal{J} = \sum_{n=1}^{n=n_f}[\frac{\partial J}{\partial W}(W^n, \alpha^n)(\sum_{l=1,p=1}^{l=n,l=n_p}\frac{\partial W^n}{\partial \alpha^l_p}d\alpha^l_p) + \sum_{p=1}^{p=n_p}\frac{\partial J}{\partial \alpha^n_p}d\alpha^n_p]$$

Given the complexity of the calculations, only a two-dimensional configuration described by a structured mesh is considered.

No variation in the shape. Scalar α. Direct differentiation method

α^i is supposed to be a scalar for every $i \in [1, n_f]$. Neither cell volume nor any geometric quantity involved in the calculation of R is altered by α. The method consists in the calculation of $\frac{\partial W^n}{\partial \alpha^l}$ $(l \leq n)$ from Eq. (1.29). The equations to differentiate are:

$$\frac{1}{\Delta t}[W^1_{i,j}V_{i,j}] - \frac{1}{\Delta t}[W^0_{i,j}V_{i,j}] + R(W^1_{i,j}, \alpha^1) = 0$$

$$\frac{3}{2\Delta t}[W^n_{i,j}V_{i,j}] - \frac{2}{\Delta t}[W^{n-1}_{i,j}V_{i,j}] + \frac{1}{2\Delta t}[W^{n-2}_{i,j}V_{i,j}] + R(W^n_{i,j}, \alpha^n) = 0 \qquad n \in [2, n_f]$$

The differentiation of the previous equations is given by:

$$\frac{V}{\Delta t}\frac{dW^1}{d\alpha_1} + \frac{\partial R}{\partial W}(W^1, \alpha^1)\frac{dW}{d\alpha^1} + \frac{\partial R}{\partial \alpha_1}(W^1, \alpha^1) = 0$$

$$\begin{aligned}
-\frac{2V}{\Delta t}\frac{\partial W^1}{\partial \alpha^1} + \frac{3V}{2\Delta t}\frac{\partial W^2}{\partial \alpha^1} \quad & + \frac{\partial R}{\partial W}(W^2, \alpha^2)\frac{\partial W^2}{\partial \alpha^1} = 0 \\
\frac{3V}{2\Delta t}\frac{\partial W^2}{\partial \alpha^2} \quad & + \frac{\partial R}{\partial W}(W^2, \alpha^2)\frac{\partial W^2}{\partial \alpha^1} + \frac{\partial R}{\partial \alpha^2}(W^2, \alpha^2) = 0
\end{aligned}$$

$$\begin{aligned}
\frac{V}{2\Delta t}\frac{\partial W^1}{\partial \alpha^1} - \frac{2V}{\Delta t}\frac{\partial W^2}{\partial \alpha^1} + \frac{3V}{2\Delta t}\frac{\partial W^3}{\partial \alpha^1} \quad & + \frac{\partial R}{\partial W}(W^3, \alpha^3)\frac{\partial W^3}{\partial \alpha^1} = 0 \\
-\frac{2V}{\Delta t}\frac{\partial W^2}{\partial \alpha^2} + \frac{3V}{2\Delta t}\frac{\partial W^3}{\partial \alpha^2} \quad & + \frac{\partial R}{\partial W}(W^3, \alpha^3)\frac{\partial W^3}{\partial \alpha^2} = 0 \\
\frac{3V}{2\Delta t}\frac{\partial W^3}{\partial \alpha^3} \quad & + \frac{\partial R}{\partial W}(W^3, \alpha^3)\frac{\partial W^3}{\partial \alpha^3} + \frac{\partial R}{\partial \alpha}(W^3, \alpha^3) = 0
\end{aligned}$$

$$\begin{aligned}
\frac{V}{2\Delta t}\frac{\partial W^{n_f-2}}{\partial \alpha^1} - \frac{2V}{\Delta t}\frac{\partial W^{n_f-1}}{\partial \alpha^1} + \frac{3V}{2\Delta t}\frac{\partial W^{n_f}}{\partial \alpha^1} \quad & + \frac{\partial R}{\partial W}(W^{n_f}, \alpha^{n_f})\frac{\partial W^{n_f}}{\partial \alpha^1} = 0 \\
\frac{V}{2\Delta t}\frac{\partial W^{n_f-2}}{\partial \alpha^2} - \frac{2V}{\Delta t}\frac{\partial W^{n_f-1}}{\partial \alpha^2} + \frac{3V}{2\Delta t}\frac{\partial W^{n_f}}{\partial \alpha^2} \quad & + \frac{\partial R}{\partial W}(W^{n_f}, \alpha^{n_f})\frac{\partial W^{n_f}}{\partial \alpha^2} = 0 \\
\cdots\cdots\cdots\cdots\cdots\cdots \quad & \cdots\cdots\cdots\cdots \\
\frac{V}{2\Delta t}\frac{\partial W^{n_f-2}}{\partial \alpha^{n_f-2}} - \frac{2V}{\Delta t}\frac{\partial W^{n_f-1}}{\partial \alpha^{n_f-2}} + \frac{3V}{2\Delta t}\frac{\partial W^{n_f}}{\partial \alpha^{n_f-2}} \quad & + \frac{\partial R}{\partial W}(W^{n_f}, \alpha^{n_f})\frac{\partial W^{n_f}}{\partial \alpha^{n_f-2}} = 0 \\
-\frac{2V}{\Delta t}\frac{\partial W^{n_f-1}}{\partial \alpha^{n_f-1}} + \frac{3V}{2\Delta t}\frac{\partial W^{n_f}}{\partial \alpha^{n_f-1}} \quad & + \frac{\partial R}{\partial W}(W^{n_f}, \alpha^{n_f})\frac{\partial W^{n_f}}{\partial \alpha^{n_f-1}} = 0 \\
\frac{3V}{2\Delta t}\frac{\partial W^{n_f}}{\partial \alpha^{n_f}} \quad & + \frac{\partial R}{\partial W}(W^{n_f}, \alpha^{n_f})\frac{\partial W^{n_f}}{\partial \alpha^{n_f}} + \frac{\partial R}{\partial \alpha}(W^{n_f}, \alpha^{n_f}) = 0
\end{aligned}$$

The direct method consists in solving these $\frac{n_f(n_f+1)}{2}$ equations to calculate the sensitivities $\frac{\partial W^n}{\partial \alpha^p}$ $p \leq n$. Finally, the computation of the gradient of the objective function (1.30) is explicit.

No variation in the shape. Vector-value of α. Direct differentiation method

Previous calculations can be extended in the case where α is a vector of dimension n_p. Differentiations are carried out for every α_p^i, $i \in [1, n_f]$, $p \in [1, n_p]$. Therefore there are $\frac{n_f(n_f+1)}{2} \times n_p$ equations to solve.

No variation in the shape. Scalar α. Adjoint method

The variation of the aerodynamic field under the variation of the control $(d\alpha^1, ..., d\alpha^{n_f})$ is given by:

$$\frac{V}{\Delta t} dW_{i,j}^1 + \frac{\partial R_{i,j}}{\partial W}(W^1, \alpha^1) dW^1 + \frac{\partial R}{\partial \alpha^1}(W_{i,j}^1, \alpha^1) d\alpha^1 = 0$$

$$\frac{3V}{2\Delta t} dW_{i,j}^n - \frac{2V}{\Delta t} dW_{i,j}^{n-1} + \frac{V}{2\Delta t} dW_{i,j}^{n-2} + \frac{\partial R_{i,j}}{\partial W}(W^n, \alpha^n) dW^n + \frac{\partial R_{i,j}}{\partial \alpha^n}(W^n, \alpha^n) d\alpha^n = 0 \qquad n \in [2, n_f]$$

The equations of the adjoint method are given by adding the product of the constraints by their adjoint vectors to the derivative of the objective function:

$$d\mathcal{L} = \sum_{n=1}^{n=n_f} \left(\frac{\partial J(W^n, \alpha^n)}{\partial W^n} dW^n + \frac{\partial J(W^n, \alpha^n)}{\partial \alpha^n} d\alpha^n\right)$$
$$+ \sum_{i,j} \lambda_{i,j}^1 \left(\frac{V_{i,j}}{\Delta t} dW_{i,j}^1 + \frac{\partial R_{i,j}}{\partial W}(W^1, \alpha^1) dW^1 + \frac{\partial R_{i,j}}{\partial \alpha^1}(W_{i,j}^1, \alpha^1) d\alpha^1\right)$$
$$+ \sum_{n=2}^{n=n_f} \sum_{i,j} \lambda_{i,j}^n \left(\frac{3V_{i,j}}{2\Delta t} dW_{i,j}^n - \frac{2V_{i,j}}{\Delta t} dW_{i,j}^{n-1} + \frac{V_{i,j}}{2\Delta t} dW_{i,j}^{n-2} + \frac{\partial R_{i,j}}{\partial W}(W^n, \alpha^n) dW^n + \frac{\partial R_{i,j}}{\partial \alpha^n}(W^n, \alpha^n) d\alpha^n\right)$$

or, written in a vector form:

$$d\mathcal{L} = \sum_{n=1}^{n=n_f} \left(\frac{\partial J(W^n, \alpha^n)}{\partial W^n} dW^n + \frac{\partial J(W^n, \alpha^n)}{\partial \alpha^n} d\alpha^n\right)$$
$$+ \lambda^{1\,T} . \left(\frac{V}{\Delta t} dW_{i,j}^1 + \frac{\partial R}{\partial W}(W^1, \alpha^1) dW^1 + \frac{\partial R}{\partial \alpha^1}(W^1, \alpha^1) d\alpha^1\right)$$
$$+ \sum_{n=2}^{n=n_f} \lambda^{n\,T} . \left(\frac{3V}{2\Delta t} dW^n - \frac{2V}{\Delta t} dW^{n-1} + \frac{V}{2\Delta t} dW^{n-2} + \frac{\partial R}{\partial W}(W^n, \alpha^n) dW^n + \frac{\partial R}{\partial \alpha^n}(W^n, \alpha^n) d\alpha^n\right)$$

The multiplicative factor of the terms $dW_{i,k}^n$ are set to 0, and the adjoint equations are given (in column vector notation) by:

$$
\begin{aligned}
(\lambda^1 - 2\lambda^2)\frac{V}{\Delta t} + (\frac{\partial R}{\partial W})^T(W^1,\alpha^1)\lambda^1 &= -\frac{\partial J(W^1,\alpha^1)}{\partial W^1}^T \\
(\frac{3}{2}\lambda^2 - 2\lambda^3 + \frac{1}{2}\lambda^4)\frac{V}{\Delta t} + (\frac{\partial R}{\partial W})^T(W^2,\alpha^2)\lambda^2 &= -\frac{\partial J(W^2,\alpha^2)}{\partial W^2}^T \\
(\frac{3}{2}\lambda^3 - 2\lambda^4 + \frac{1}{2}\lambda^5)\frac{V}{\Delta t} + (\frac{\partial R}{\partial W})^T(W^3,\alpha^3)\lambda^3 &= -\frac{\partial J(W^3,\alpha^3)}{\partial W^3}^T \\
&\cdots\cdots\cdots\cdots \\
(\frac{3}{2}\lambda^{n_f-2} - 2\lambda^{n_f-1} + \frac{1}{2}\lambda^{n_f})\frac{V}{\Delta t} + (\frac{\partial R}{\partial W})^T(W^{n_f-2},\alpha^{n_f-2})\lambda^{n_f-2} &= -\frac{\partial J(W^{n_f-2},\alpha^{n_f-2})}{\partial W^{n_f-2}}^T \\
(\frac{3}{2}\lambda^{n_f-1} - 2\lambda^{n_f})\frac{V}{\Delta t} + (\frac{\partial R}{\partial W})^T(W^{n_f-1},\alpha^{n_f-1})\lambda^{n_f-1} &= -\frac{\partial J(W^{n_f-1},\alpha^{n_f-1})}{\partial W^{n_f-1}}^T \\
(\frac{3}{2}\lambda^{n_f})\frac{V}{\Delta t} + (\frac{\partial R}{\partial W})^T(W^{n_f},\alpha^{n_f})\lambda^{n_f} &= -\frac{\partial J(W^{n_f},\alpha^{n_f})}{\partial W^{n_f}}^T
\end{aligned}
$$

Then, the derivative of the objective function can be written:

$$
d\mathcal{J} = \sum_{n=1}^{n=n_f} \frac{\partial J(W^n,\alpha^n)}{\partial \alpha^n} d\alpha^n + \sum_{n=1}^{n=n_f} \lambda^{n\,T} \frac{\partial R}{\partial \alpha^n}(W^n,\alpha^n) d\alpha^n
$$

The solution is performed backward in time from t^{n_f} to t^1.

1.6.4 Function evaluation, adjoint vector and mesh refinement

This paragraph describes the work of Venditti and Darmofal [5] deriving from an article written by Pierce and Giles [2]. This work is about the differences observed on the function $J(X,W)$ between two meshes of different sizes. The coarser one will be noted H and the finer one h. On one hand, the difference $J_h - J_H$ can be used to correct the function calculated on the coarse grid. On the other hand, it can be used to identify under-resolved zones for an automatic mesh refinement. This approach differs from the classical methods where the mesh is refined in large gradients zones.
The problem to solve is the following one: how to perform J on a coarse grid H with an accuracy comparable to a calculation performed on a fine grid h without computing the solution on the fine grid h ? Let W_h, J_h and R_h be the estimations of W, J and R on the grid h, and W_H, J_H and R_H be the estimations of W, J and R on the grid H. The aim is to find a relationship between J_h and J_H from W_H and without performing W_h. The extension of W_H on the grid h is noted W_h^H:

$$
W_h^H = A_h^H \cdot W_H
$$

J is supposed not to depend on the mesh X. A first order expansion in Taylor series of $J(W_h)$ around W_h^H gives:

$$
J_h(W_h) = J_h(W_h^H) + \frac{\partial J_h}{\partial W_h}\Big|_{W_h^H}(W_h - W_h^H) + \mathcal{O}((W_h - W_h^H)^2)
$$

where W_h is a solution of $R_h(W_h) = 0$. The first order Taylor expansion of $R_h(W_h)$ gives:

$$
R_h(W_h) = 0 = R_h(W_h^H) + \frac{\partial R_h}{\partial W_h}\Big|_{W_h^H}(W_h - W_h^H) + \mathcal{O}((W_h - W_h^H)^2)
$$

If $\det\left(\frac{\partial R_h}{\partial W_h}\Big|_{W_h^H}\right) \neq 0$, we have:

$$W_h - W_h^H = -\left[\frac{\partial R_h}{\partial W_h}\Big|_{W_h^H}\right]^{(-1)} R_h(W_h^H) + \mathcal{O}((W_h - W_h^H)^2)$$

The adjoint problem can be written:

$$\left[\frac{\partial R_h}{\partial W_h}\Big|_{W_h^H}\right]^T \lambda_h\big|_{W_h^H} = -\left[\frac{\partial J_h}{\partial W_h}\Big|_{W_h^H}\right]^T \quad (\lambda_h\big|_{W_h^H})^T \frac{\partial R_h}{\partial W_h}\Big|_{W_h^H} = -\frac{\partial J_h}{\partial W_h}\Big|_{W_h^H}$$

and the extension of λ_H on the fine grid, noted λ_h^H is:

$$\lambda_h^H = B_h^H \cdot \lambda_H$$

The value of the expected $J_h(W_h)$ is:

$$J_h(W_h) = J_h(W_h^H) + (\lambda_h\Big|_{W_h^H})^T R_h(W_h^H) + \mathcal{O}((W_h - W_h^H)^2)$$

$$J_h(W_h) = J_h(W_h^H) + \underbrace{(\lambda_h^H)^T R_h(W_h^H)}_{\text{correction}} + \underbrace{((\lambda_h\Big|_{W_h^H})^T - (\lambda_h^H)^T) R_h(W_h^H)}_{\text{error of correction}} + \mathcal{O}((W_h - W_h^H)^2)$$

To improve the accuracy of the objective function, a natural way is to use the correction term $(\lambda_h^H)^T R_h(W_h^H)$ as a parameter for mesh refinement [1].
Venditti and Darmofal carried out analysis on the error corrective term to derive user-friendly formulations for mesh refinement. The use of $R_h^\lambda(\lambda) = \left[\frac{\partial R_h}{\partial W_h}\Big|_{W_h^H}\right]^T \lambda - \left(\frac{\partial J_h}{\partial W_h}\Big|_{W_h^H}\right)^T$ allows to write for the error of correction:

$$EC_1 = \left(\left(\lambda_h\Big|_{W_h^H}\right)^T - (\lambda_h^H)^T\right) R_h(W_h^H) + \mathcal{O}((W_h - W_h^H)^2)$$

which can be rewritten in two different ways:

$$EC_2 = \left(R_h^\lambda(\lambda_h^H)\right)^T \left[\frac{\partial R_h}{\partial W_h}\Big|_{W_h^H}\right]^{-1} R_h(W_h^H) + \mathcal{O}((W_h - W_h^H)^2)$$

$$EC_3 = \left(R_h^\lambda(\lambda_h^H)\right)^T (W_h - W_h^H) + \mathcal{O}((W_h - W_h^H)^2)$$

The expression EC_2 shows that the error of correction can be seen like the product of the residuals between mechanics and adjoint equations. The main terms of EC_1 and EC_2 are the same. The objective of an adaptation process can be to reduce simultaneously the residuals of both fluid and adjoint calculations on the fine grid -R_h^λ and R^h- for the extended fields λ_h^H and W_h^H. The method proposed in [5] is to reduce the errors of correction and to dispatch them on the grid in an uniform way by using formulas EC_1 and EC_3. An approximation of $EC_1 + EC_3$ on every cell of the coarse grid (subscript k) is used (the subscript $l(k)$ corresponds to the fine grid cells which are included in the element k).

$$\epsilon_k = \frac{1}{2}\sum_{l(k)} \left(\left| \left[Q_h^H \lambda_H - L_h^H \lambda_H\right]^T_{l(k)} \left[R_h(L_h^H U_H)\right]_{l(k)} \right| + \left| \left[Q_h^H W_H - L_h^H W_H\right]^T_{l(k)} \left[R_h^\lambda(L_h^H \lambda_H)\right]_{l(k)} \right| \right)$$

L_h^H and Q_h^H refer to as respectively the linear and quadratic extension operators between the fine grid and the coarse one. The terme $\lambda_h\Big|_{W_h^H}$ is not computed in an explicit way, but approximated by $Q_h^H \lambda_H$, which is a more precise term than $L_h^H \lambda_H$ giving an approximation of λ_h^H. This allows to build a hierarchy between $\lambda_h\Big|_{W_h^H}$ and λ_h^H.

BIBLIOGRAPHY

[1] R. Becker and R. Rannacher. Weighted a posteriori error control in FE methods. *preprint*, 1, 1996. 29

[2] M.B. Giles and N.A. Pierce. An introduction to the adjoint approach to design. *Flow, Turbulence and Combustion*, 65(3):393–415, 2000. 19, 28

[3] J. Peter, M. Marcelet, and S. Burguburu. Introduction à l'optimisation de forme en aérodynamique et quelques exemples d'application. *Publication ONERA, 2006*. 7

[4] G.M. Shroff and H.B. Keller. Stabilization of unstable procedures: the recursive projection method. *SIAM J. Numer. Anal.*, 30(4):1099–1120. 16

[5] D.A. Venditti and D.L. Darmofal. Grid adaptation for functional outputs: application to two-dimensional inviscid flows. *Journal of Computational Physics*, 176(1):40–69, 2002. 28, 29

Optimization

Empty page

Part II

USER INTERFACE ELEMENTS

Empty page

1.7 The shapeopt description class

The user interface for the `elsA_Opt` module, gathering the additional functionality for shape optimisation for the *elsA* software, is defined by the shapeopt description class.
Please refer to `/ELSA/MU-98057` for generic documentation on the *elsA* description classes.

1.8 Attributes of the shapeopt class

▷ `opt_gradcompmode` .opt_gradcompmode

Specifies which method is used for the computation of the derivative of the scalar functions with respect to the shape parameters.
default value : `linearized`

```
<shapeopt>.set('opt_gradcompmode', '<opt_gradcompmode>')
```

`<opt_gradcompmode> =`

adjoint_mesh	: compute the total derivative of aerodynamic functions with respect to mesh nodes. Output file is dFdX_00n_00p.tp (n:function number, p:domain number) ;
adjoint_param	: compute the derivatives of aerodynamic functions with respect to shape parameters in adjoint mode ;
linearized_param	: compute the derivatives of aerodynamic functions with respect to shape parameters in direct mode.

Ex. :

```
num1.set('opt_gradcompmode', 'linearized_param')
```

▷ `opt_nbcontrol` .opt_nbcontrol

Defines the number of parameters controlling the shape of the solid object.

```
<shapeopt>.set('opt_nbcontrol', <int>)
```

Ex. :

```
opt1.set('opt_nbcontrol', 10)
```

▷ `opt_nbfunction` .opt_nbfunction

Defines the number of functions to be differentiated with respect to the design parameters.

```
<shapeopt>.set('opt_nbfunction', <int>)
```

Ex. :

```
opt1.set('opt_nbfunction', 3)
```

▷ `opt_fileway` .opt_fileway

Directory containing auxiliary files $\frac{dX}{d\alpha_i}$, $\frac{\partial J}{\partial W}$, $\frac{\partial J}{\partial W_b}$, $\frac{\partial J}{\partial X}$.
default value : `.`

```
<shapeopt>.set('opt_fileway', '<string>')
```

▷ `opt_compgrad` .opt_compgrad

Activates/Desactivates the computation of the gradient of the functions.
default value : `active`

```
<shapeopt>.set('opt_compgrad', '<opt_compgrad>')
```

`<opt_compgrad>` =

`inactive` : no computation of the gradients after resolution of adjoint or linearized equation ;
`active` : computation of the gradients after resolution of adjoint or linearized equation.

Ex. :

```
num1.set('opt_compgrad', 'active')
```

▷ `opt_newtonstep` .opt_newtonstep

Defines the number of iterations (which can be seen as Newton steps) of the iterative resolution of adjoint or direct equation.

```
<shapeopt>.set('opt_newtonstep', <int>)
```

Ex. :

```
opt1.set('opt_newtonstep', 3)
```

▷ `opt_relaxcycle` .opt_relaxcycle

Defines the number of LU relaxation sweeps used at each iteration of the resolution procedure of adjoint or linearized equation. Possible values are $2, 4, 6, 8, 10$.

```
<shapeopt>.set('opt_relaxcycle', <int>)
```

Ex. :

```
opt1.set('opt_relaxcycle', 6)
```

▷ `opt_nbdparam` .opt_nbdparam

Defines the number of increments $d\alpha$ used in the finite difference computation of $\frac{\partial R}{\partial X}\frac{dX}{d\alpha}$ (geometric sensitivity of the explicit residual). It also the order of accuracy of the formula. The standart choice is the second order formula with $d\alpha_{i,1} = -d\alpha_{i,2} = \delta\alpha_i$ corresponding to

$$\frac{\partial R}{\partial X}\frac{dX}{d\alpha_i} \simeq \frac{R(W(\alpha), X(\alpha) + \delta\alpha_i \frac{dX}{d\alpha_i})) - R(W(\alpha), X(\alpha) - \delta\alpha_i \frac{dX}{d\alpha_i}))}{2\delta\alpha_i}$$

(see also next definition)

```
<shapeopt>.set('opt_nbdparam', <int>)
```

Ex. :

```
opt1.set('opt_nbdparam', 2)
```

▷ `opt_dparamfile` .opt_dparamfile

File name for formated Tecplot file in which the increment $d\alpha_{i,j}$ are read (see last definition).

```
<shapeopt>.set('opt_dparamfile', '<string>')
```

▷ `opt_blocksfororder1` .opt_blocksfororder1

Ordered list of block numbers for order one linearization of upwind mean flow convective flux. To be used only for second order analysis computation and in case of robustness problem of gradient computation (first DesBlock declaration corresponds to number 1 and so on)

```
<shapeopt>.set('opt_blocksfororder1', '<string>')
```

▷ `opt_conv_orderone` .opt_conv_orderone

Force order one linearization for mean flow upwind convective flux for all blocks. To be used only for second order analysis computation and in case of robustness problem of gradient computation.
default value : `inactive`

```
<shapeopt>.set('opt_conv_orderone', '<opt_conv_orderone>')
```

`<opt_conv_orderone>` =

`active` : force order one linearization for convection terms ;
`inactive` : linearize exactly the convection term of the analysis computation.

▷ `opt_list_def_blocks` .opt_list_def_blocks

For complex configurations with design parameters deforming only a limited number of blocks. List of deformed block numbers. For non deformed blocks NO $\frac{dX}{d\alpha_i}$ files full of zero need to be provided.

```
<shapeopt>.set('opt_list_def_blocks', '<string>')
```

▷ `opt_useartdiss` .opt_useartdiss

Flag to include an artificial dissipation term in $(\frac{\partial R}{\partial W})^{EXA}$ matrix.
default value : `inactive`

```
<shapeopt>.set('opt_useartdiss', '<opt_useartdiss>')
```

`<opt_useartdiss> =`

`active` : use artificial dissipation ;
`inactive` : do not use artificial dissipation.

▷ `opt_artdisstype` .opt_artdisstype

Artificial dissipation type in $(\frac{\partial R}{\partial W})^{EXA}$.
default value : `0`

```
<shapeopt>.set('opt_artdisstype', <opt_artdisstype>)
```

`<opt_artdisstype> =`

`0` : artificial dissipation type 0 (no dissipation) ;
`1` : artificial dissipation type 1 (no sensor) ;
`2` : artificial dissipation type 2 (five sensors based on adjoint variables gradient) ;
`3` : artificial dissipation type 3 (one sensor based on adjoint variables gradient);
`4` : artificial dissipation type 4 (sensor relying on static pressure gradient).

▷ `opt_k2` .opt_k2

Artificial dissipation 2nd order diff coeff in $(\frac{\partial R}{\partial W})^{EXA}$ matrix.
default value : `0.5`

```
<shapeopt>.set('opt_k2', <float>)
```

▷ `opt_k4` .opt_k4

Artificial dissipation 4th order diff coeff. in $(\frac{\partial R}{\partial W})^{EXA}$ matrix.
default value : `0.032`

```
<shapeopt>.set('opt_k4', <float>)
```

▷ `opt_gradfile` .opt_gradfile

File for gradient extraction (formated Tecplot file).

```
<shapeopt>.set('opt_gradfile', '<string>')
```

▷ `opt_imp_adjtype` .opt_imp_adjtype

Implicit stage.
default value : `0`

```
<shapeopt>.set('opt_imp_adjtype', <opt_imp_adjtype>)
```

```
<opt_imp_adjtype> =
```

`0` : use LU-RELAX implicitation for adjoint equation ;
`1` : use LU-SSOR implicitation for adjoint equation.

▷ `opt_imp_coeffeuler` .opt_imp_coeffeuler

Flag to multiply all mean flow convective jacobian terms by 1.5 in $(\frac{\partial R}{\partial W})^{APP}$ matrix to take very roughly into account a second order upwind discretisation in the analysis computation.
default value : `inactive`

```
<shapeopt>.set('opt_imp_coeffeuler', '<opt_imp_coeffeuler>')
```

```
<opt_imp_coeffeuler> =
```

`active` : use coeffeuler $= 1.5$;
`inactive` : use coeffeuler $= 1.0$.

▷ `opt_imp_pctrad` .opt_imp_pctrad

Harten-type parameter for convective jacobian matrices inside $(\frac{\partial R}{\partial W})^{APP}$ LURELAX-type implicit matrices (mandatory choice for linearized equation, one of two possibilities for adjoint equations). High values enhance robustness but slow down convergence process.

```
<shapeopt>.set('opt_imp_pctrad', <float>)
```

▷ `opt_imp_usetimestep` .opt_imp_usetimestep

Flag to use the $\frac{1}{\Delta t}$ term in $(\frac{\partial R}{\partial W})^{APP}$ matrix of iterative resolution for adjoint or direct system.
default value : `inactive`

```
<shapeopt>.set('opt_imp_usetime', '<opt_imp_usetimestep>')
```

```
<opt_imp_usetimestep> =
```

`active` : include the $\frac{1}{\Delta t}$ term ;
`inactive` : do not include the $\frac{1}{\Delta t}$ term .

▷ `opt_imp_cflfactor` **.opt_imp_cflfactor**

If last key is set to "active", the time step of the $\frac{1}{\Delta t}$ term will be computed from following CFL number : $CFL_{opt} = CFL_{analysis} * opt\_imp\_cflfactor$.
default value : `1.0`

```
<shapeopt>.set('opt_imp_cflfactor', <float>)
```

▷ `opt_imp_useartdiss` **.opt_imp_useartdiss**

Flag to include an artificial dissipation term in $(\frac{\partial R}{\partial W})^{APP}$ matrix of iterative resolution (option is only available for LU-SSOR adjoint resolution).
default value : `active`

```
<shapeopt>.set('opt_imp_useartdiss', '<opt_mp_useartdiss>')
```

`<opt_imp_useartdiss> =`

`active` : include an artificial dissipation term ;
`inactive` : do not include an artificial dissipation term.

▷ `opt_inititer` **.opt_inititer**

First iteration of restart.

```
<shapeopt>.set('opt_inititer', <int>)
```

▷ `opt_refresidual` **.opt_refresidual**

Ordered list of reference residuals for a restart.

```
<shapeopt>.set('opt_refresidual', '<string>')
```

▷ `opt_restartfiles` **.opt_restartfiles**

Ordered list of restart files.

```
<shapeopt>.set('opt_restartfiles', '<string>')
```

▷ `opt_restartformat` **.opt_restartformat**

Restart files format.

```
<shapeopt>.set('opt_restartformat', '<string>')
```

▷ `opt_residualfile` .opt_residualfile

File name for residual values.
default value : `NewtonResidual.tp`.

Corresponding file is a Tecplot bloc formatted file.

```
<shapeopt>.set('opt_residualfile', '<string>')
```

▷ `opt_residualtype` .opt_residualtype

Type of residual extraction.
default value : `global`

```
<shapeopt>.set('opt_residualtype', '<opt_residualtype>')
```

`<opt_residualtype> =`

`global` : perform extraction for all blocks ;
`local` : perform per-block extraction.

▷ `opt_aoa_control` .opt_aoa_control

Control the angle of attack.
default value : `active`

```
<shapeopt>.set('opt_aoa_control', '<opt_aoa_control>')
```

`<opt_aoa_control> =`

`active` : activate attack angle control ;
`inactive` : do not activate attack angle control.

▷ `opt_compgradfile` .opt_compgradfile

Base name for direct gradient computation

```
<shapeopt>.set('opt_compgradfile', '<string>')
```

▷ `opt_expke_dest` .opt_expke_dest

Type of approximation in viscous destruction term derivation.
default value : `thin_layer`

```
<shapeopt>.set('opt_expke_dest', '<opt_expke_dest>')
```

`<opt_expke_dest> =`

dmudw : use the $\frac{d\mu}{dw}$ approximation ;
nothing : no approximation ;
thin_layer : use the thin-layer approximation.

▷ `opt_expke_fvis` .opt_expke_fvis

Type of approximation in viscous flux derivation.
default value : *thin_layer*

```
<shapeopt>.set('opt_expke_fvis', '<opt_expke_fvis>')

<opt_expke_fvis> =
```

dmudw : use the $\frac{d\mu}{dw}$ approximation ;
nothing : no approximation ;
thin_layer : use the thin-layer approximation.

▷ `opt_expke_lowcor` .opt_expke_lowcor

Type of approximation in low-Reynolds correctione term derivation.
default value : *thin_layer*

```
<shapeopt>.set('opt_expke_lowcor', '<opt_expke_lowcor>')

<opt_expke_lowcor> =
```

dmudw : use the $\frac{d\mu}{dw}$ approximation ;
nothing : no approximation ;
thin_layer : use the thin-layer approximation.

▷ `opt_expke_prod` .opt_expke_prod

Type of approximation in production term derivation.
default value : *thin_layer*

```
<shapeopt>.set('opt_expke_prod', '<opt_expke_prod>')

<opt_expke_prod> =
```

dmudw : use the $\frac{d\mu}{dw}$ approximation ;
nothing : no approximation ;
thin_layer : use the thin-layer approximation.

▷ `opt_mach_control` .opt_mach_control

Control the Mach number.
default value : *active*

```
<shapeopt>.set('opt_mach_control', '<opt_mach_control>')
```

`<opt_mach_control>` =

active : do control the Mach number ;
inactive : do not control the Mach number.

▷ `opt_updtnewtsol_switchstep` .opt_updtnewtsol_switchstep

Newton step for the update control

```
<shapeopt>.set('opt_updtnewtsol_switchstep', <int>)
```

▷ `opt_updtnewtsol_useswitch` .opt_updtnewtsol_useswitch

Control the update of the LHS of the Newton equation.
default value : *1*

▷ `opt_linmichel` .opt_linmichel

Linearization of Michel turbulence model.
default value : *inactive*

▷ `opt_par_inf_dmach` .opt_par_inf_dmach

Number of design param for change of inf state Mach

```
<shapeopt>.set('opt_par_inf_dmach', <int>)
```

▷ `opt_par_inf_rotx` .opt_par_inf_rotx

Number of design param for rotation of inf state around X

```
<shapeopt>.set('opt_par_inf_rotx', <int>)
```

▷ `opt_par_inf_roty` .opt_par_inf_roty

Number of design param for rotation of inf state around Y

```
<shapeopt>.set('opt_par_inf_roty', <int>)
```

▷ `opt_par_inf_rotz` .opt_par_inf_rotz

Number of design param for rotation of inf state around Z

```
<shapeopt>.set('opt_par_inf_rotz', <int>)
```

▷ `opt_par_rotox` .opt_par_rotox

Number of design param for rotation around OX

```
<shapeopt>.set('opt_par_rotox', <int>)
```

▷ `opt_par_rotoy` .opt_par_rotoy

Number of design param for rotation around OY

```
<shapeopt>.set('opt_par_rotoy', <int>)
```

▷ `opt_par_rotoz` .opt_par_rotoz

Number of design param for rotation around OZ

```
<shapeopt>.set('opt_par_rotoz', <int>)
```

1.9 Specific values of extractor.var for shapeopt

List of values to extract when a shapeopt object is defined, using an extract-like extractor object :

`<var>` =

adj1	: Adjoint vector (all components) of 1^{st} function ;
adj2	: Adjoint vector (all components) of 2^{nd} function ;
adj3	: Adjoint vector (all components) of 3^{rd} function ;
adj4	: Adjoint vector (all components) of 4^{th} function ;
adj5	: Adjoint vector (all components) of 5^{th} function ;
adj6	: Adjoint vector (all components) of 6^{th} function ;
adj7	: Adjoint vector (all components) of 7^{th} function ;
adj8	: Adjoint vector (all components) of 8^{th} function ;
adj9	: Adjoint vector (all components) of 9^{th} function ;
adj10	: Adjoint vector (all components) of 10^{th} function ;
dwda1	: Flow sensitivity with respect to 1^{st} design parameter ;
dwda2	: Flow sensitivity with respect to 2^{nd} design parameter ;
dwda3	: Flow sensitivity with respect to 3^{rd} design parameter ;
dwda4	: Flow sensitivity with respect to 4^{th} design parameter ;
dwda5	: Flow sensitivity with respect to 5^{th} design parameter ;
dwda6	: Flow sensitivity with respect to 6^{th} design parameter ;
dwda7	: Flow sensitivity with respect to 7^{th} design parameter ;
dwda8	: Flow sensitivity with respect to 8^{th} design parameter ;
dwda9	: Flow sensitivity with respect to 9^{th} design parameter ;
dwda10	: Flow sensitivity with respect to 10^{th} design parameter.