Politecnico di Torino

DIMEAS - Department of Mechanical and Aerospace Engineering

Master Degree in Aerospace Engineering

Master Thesis

A modular time-spectral implementation for aeroelastic analysis and optimization on structured meshes

Author
Ms. Francesca Basile

Academic Supervisor
Prof. Gaetano Iuso

ONERA Supervisors
Ing. Cédric Liauzun
Ing. Christophe Blondeu

March 27th, 2019

Acknowledgments

I would like to thank everybody who helped me get to this point.

A huge thanks goes to my French supervisors, Cédric Liauzun and Christophe Blondeau, who gave me the possibility to work in such an important research center as ONERA, and let me blend into a new context and enrich my knowledge, but above all, patiently supported me during my work.

Thanks to Fabio, Rocco, Javi, and all my young colleagues ONERA: their precious technical tips and friendship made my job and my life experience in Paris easier.

Moreover I thank my Italian supervisor prof. Gaetano Iuso, his advise and his presence during these years of study.

A special dedication goes to Federico Barra, Francesco, Martina and Federico Boni, and to Alessandra, Irene and Simona, my classmates from Politecnico and my flatmates in Turin for years. Their enthusiasm, fraternity and affection in sharing both happy and stressful times, and many experiences during my university years, helped me growing personally but also contributed to my academical achievements.

Finally I want to thank my family, and especially my mother, for her long-distance psychological support and extreme patience during the most critical moments of my studies (and why not, also for the economical support).

Thank you!

Contents

List of Figures

Nomenclature

a	speed of sound
α	angle of attack
C_{D_p}	pressure drag coefficient
C_L	lift coefficient
C_M	moment coefficient
$\underline{\underline{\mathbf{D}_t}}$	time - spectral matrix
f	frequence
E	total energy
$\underline{\underline{\boldsymbol{\mathcal{E}}}}$	matrix operator of Discrete Fourier Transform
J	objective function
$\underline{\underline{\mathbf{I}}}$	identity matrix
$\underline{\underline{\frac{\partial \mathbf{R}}{\partial \mathbf{W}}}}$	Jacobian matrix
p	pressure
S	source term
t	time
T	period $T = 1/f$
τ	pseudo time
u, v, w	components of speed in the axis x, y, z, respectively
U	norm of speed $U = \sqrt{u^2 + v^2 + w^2}$
U_e	entrainment speed
V	volume of the computational cell
W	vector of conservative variables $W = (\rho, \rho u, \rho v, \rho w, \rho E)$
ω	pulsation
Ω	angular speed of the mobile block
(x, y, z)	cartesian coordinates

Subscripts, superscript, symbols

a_k - *(TSM solver)*	quantity a at iteration k of Block Jacobi
a_k - *(Optimization)*	quantity a at iteration k of the optimization process
a_n	quantity a at instant n
a^q	quantity a at iteration q of Newton
$\underline{\underline{\mathbf{A}}}^T$	transpose of the $\underline{\underline{\mathbf{A}}}$ matrix
$\overline{\mathbf{W}}$	concatenation of instants $\mathbf{W} = (W_1, W_2, ..., W_{2N+1})$ (simplification in discussions dedicated to TSM)

Acronymes

CFD	*Computational Fluid Dynamics*
CFL	*Courant-Friedriech-Levy number*
DFT	*Discrete Fourier Transform*
DLM	*Doublet-Lattice Method*
elsA	*ensemble logiciel pour la simulation en Aérodinamique*
GMRES	*Generalized Minimal RESidual method*
HBT	*Harmonic Balance Technique*
IDFT	*Inverse Discrete Fourier Transform*
ILU	*Incomplete Lower-Upper*
LFD	*Linearized Frequency Domain*
LU-SSOR	*Lower-Upper-Symmetrical Successive Over Relaxation*
LU	*Lower-Upper*
NACA	*National Advisory Committee for Aeronautics*
NLFD	*Non-Linear Frequency Domain*
ONERA	*Office National d'Études et Recherches Aérospatiales*
SLSQP	*Sequential Least SQuares Programming*
TSM	*Time Spectral Method*
URANS	*Unsteady Reynolds-Averaged Navier-Stokes*

Abstract — Shape optimization of wings in periodic motion is a modern and highly valuable discipline, which involves strictly research and industry. The necessity of a trade-off between accuracy and computational time lead to different choices in terms of solver from the standard ones.

The classical unsteady simulation methods are not always effective in solving periodic motions, because an expensive transient needs to be met first.

The Time Spectral Method allows to simulate a periodic flow, coupling several steady computations with a satisfying accuracy and a consistent reduction of time.

This work has the aim of implementing a robust TSM solver for the solution of external flows. The solver has been developed externally in a python script, interacting with the ONERA CFD solver *elsA*, exploiting its modularity, but adopting a different approach: the exact Jacobian matrix is extracted and numerical strategies are investigated to help the convergence process. The code has been validated via test cases comparisons performed by the reference solver *elsA*, and moreover it showed faster convergence.

A first gradient-based steady optimization is implemented, creating the basis for the future implementation of the unsteady TSM optimizer.

Keywords : aeroelasticity, numerical methods, unsteady optimization, time spectral method, transonic unsteady flows, modular implementation

Abstract — L'ottimizzazione di forma di ali in movimento periodico è una disciplina moderna e altamente valida, che coinvolge strettamente ricerca e industria. La necessità di un compromesso tra accuratezza e tempi di calcolo portano a scelte diverse in termini di solutore, rispetto a quelle standard.

I classici metodi di simulazione instazionaria non sono sempre efficienti nel risolvere moti periodici, poichè prima che la soluzione converga a quella a regime, viene necessariamente incontrato un transitorio costoso e spesso non utile.

Il metodo Time-Spectral permette di simulare un flusso periodico, accoppiando diversi calcoli stazionari con un'accuratezza soddisfacente ed una riduzione di tempo consistente.

Il presente lavoro ha l'obiettivo di implementare un solutore TSM robusto per la risoluzione di flussi esterni. Il solutore è stato sviluppato esternamente in uno script python, interagendo con il solutore CFD ONERA *elsA*, sfruttando la sua modularità, ma adottando un approccio differente: è estratta la matrice Jacobiana esatta e sono investigate delle strategie numeriche per aiutare il processo di convergenza. Il codice è stato validato attraverso casi test, operando confronti tra i risultati del solutore esterno e quelli del solutore di riferimento *elsA*, riuscendo inoltre a dimostrare una convergenza più rapida.

Una prima ottimizzazione stazionaria gradient-based è stata implementata, gettando le basi per la futura implementazione dell'ottimizzatore TSM instazionario.

Parole chiave : aeroelasticità, metodi numerici, ottimizzazione instazionaria, metodo time-spectral, implementazione modulare

ONERA
29 avenue de la Division Leclerc
92322, Châtillon, France

Politecnico di Torino
Corso Duca degli Abruzzi, 24
10129, Torino, Italy

Introduction

Nowadays the development of high performances aircraft, helicopters, drones, engines or missiles is the main goal of aerospace industry. Performances are improved if the design is well adapted to the operative flight conditions and the desired forces: in order to achieve this goal a study on the shape optimization is needed.

Powerful tools aimed at steady shape optimization have already been developed. Thanks to CFD codes like elsA, provided with an optimization module able to extract the desired gradients and optimizers that follow an algorithm to find the right parameters to minimize an objective function, usually drag, it is possible to perform steady optimization, but also unsteady periodic optimization: it means finding the parameters which minimize the average drag over a period, for a pitching airfoil. This is the ultimate goal of the present work (and the following works).

In order to compute shape optimization problems externally from the ONERA CFD code, building an external solver of flows is first of all necessary. It is chosen to build a Time Spectral Method solver.

The reason of this choice stands in terms of speed and efficiency of calculations: in fact, an unsteady complete computation needs to encounter a transient before reaching a steady periodic regime. This transient is expensive in terms of number of iterations and subsequently in terms of CPU time, and above all it is irrelevant for most of practical applications when the main intent is modelling of physical phenomena. This is why new techniques have been developed recently.

The Time Spectral Method, dedicated to periodic flows and based on Fourier analysis, allows to bypass the transient, to compute only the established periodic motion, saving computational time for purely periodic problems when compared to standard time-implicit methods. The established motion is computed by several steady computations, coupled each other by a source term.

Hence non-linear systems resulting from time-spectral discretizations should be solved, that become larger and stiffer as more time instances are employed or the period of the flow becomes especially short.

After building a TSM solver, its behaviour is investigated, and new strategies are proposed.

Thesis organization

Chapter 1 presents the two test cases performed on the airfoil *NACA 64A010*, on which the solver is tested. The airfoil is in a sinusoidal pitching motion, and the two conditions studied consist in one with a low Mach number flow and low forcing frequency, and one with a high transonic Mach number flow with a higher frequency.

Chapter 2 introduces the Time Spectral Method in detail, deriving the final system to solve, and presenting also an example of how the time-spectral matrix, key feature of TSM, is exploited in simple cases.

Chapter 3 covers the numerical models adopted to simulate a flow around an airfoil in pitching motion, taking into account the continuous change of frame of reference in the motion equations.

In Chapter 4 direct and iterative solution methods of linear systems are discussed.

Chapter 5 gives a short overview of the CFD solver *elsA*, concerning the discretization models used and the solution process of a steady mean flow.

Chapter 6 deals with the core of the thesis, providing a detailed description of the steps carried out in the numerical development of the solver: first the implementation of the partially implicit TSM, its parallelization, then the fully implicit formulation and its adaptation to parallel computing. The flowchart of the algorithm is presented to show clearly how the solver works.

After finding in Chapter 7 the right reference data to validate the solver for the two test cases, in Chapters 8 and 9 the results of the steady and the TSM external solver are provided: the density field, the wall pressure and the aerodynamic coefficients are validated with respect to reference solutions, and the convergence behaviour is analyzed.

In Chapter 10 a steady optimization is performed to one of the test cases adopted for the solver, constituting a first step in the building of the final unsteady optimizer.

Chapter 11 gives a summary of this dissertation as well as a discussion on directions for future work.

Chapter 1

Test cases

The test cases to test and validate the implementation of the TSM solver involve a symmetric airfoil in a pitching motion. The prescribed motion is sinusoidal, that means that the angle of attack follows a periodic function:

$$\alpha(t) = \alpha_0 + \widehat{\alpha}\sin(\omega t) \tag{1.1}$$

The two test cases have been performed experimentally by the AGARD [1] on a *NACA64A010* airfoil: one implies a low Mach case and the other a higher one, respectively named *dynamic index 29* and *dynamic index 55*.

M	=	0.502
p_∞	=	171000 Pa
p°	=	203152 Pa
q	=	30199 Pa
f	=	10.8 Hz
α_0	=	-0.22°
$\widehat{\alpha}$	=	1.02°

Table 1.1: Dynamic index 29

M	=	0.796
p_∞	=	133912 Pa
p°	=	203321 Pa
q	=	59395 Pa
f	=	34.4 Hz
α_0	=	-0.21°
$\widehat{\alpha}$	=	1.01°

Table 1.2: Dynamic index 55

where α_0 is the mean angle of attack and $\widehat{\alpha}$ is the amplitude of the pitching motion.

So two Mach numbers, $M = 0.502$ and $M = 0.796$, and two frequencies of pitching will be studied, $f = 10.8Hz$ and $f = 34.4Hz$.

The studied flow is compressible and it is studied by the *Euler Equations*: it means that there are no viscosity, thermal effects and gravity which influence the state of the fluid, air in this case.

1.1 Mesh

The mesh adopted for the test cases corresponds to a *NACA 64A010* airfoil.

The NACA airfoils are airfoils shaped for aircraft wings developed by the National Advisory Committee for Aeronautics (NACA). The shape of NACA airfoils is described using a series of digits following the word "NACA", representing several numerical shape parameters, to enter into equations and generate precisely the desired shape.

The characteristics of the 64A - series studied airfoil *NACA 64A010* are the following:

- the position of the maximum thickness is located at 40 percent of the chord
- the airfoil is symmetric (due to the fourth digit - 0), or the lift coefficient can't exceed the 0 value for a 0 angle of attack
- the maximum thickness ratio is 10 percent of the chord

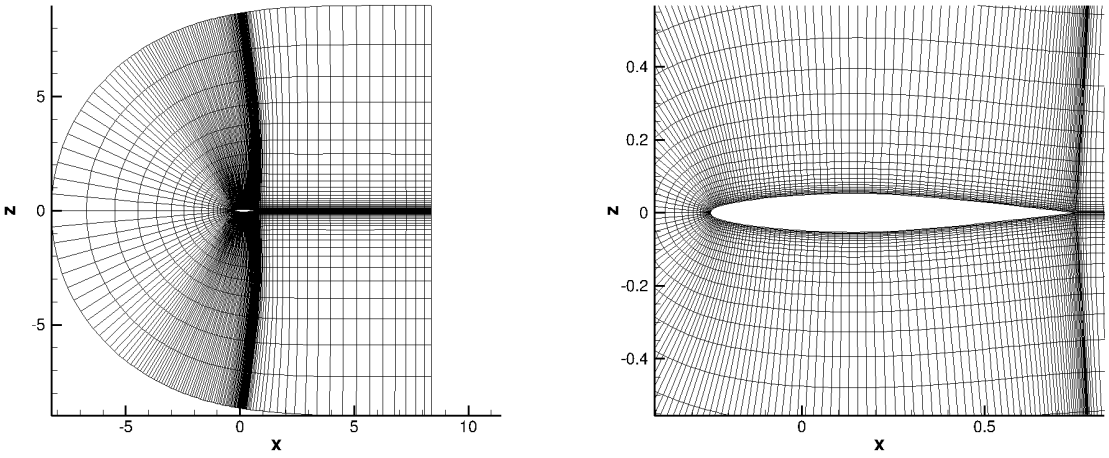


Figure 1.1: Structured 2D C-mesh of the *NACA64A010* airfoil

The mesh files contain the coordinates of the nodes in x, y, z direction, in form of block. The problem studied is a *2D problem* in x, z directions, and the adopted mesh is called *"false 3D problem"* because of the presence of only two planes in the y - *direction*, used to satisfy the condition of zero speed and zero fluxes in this direction.

The adopted mesh is of the *structured* type. In fact there are roughly two kinds of mesh to be used in CFD ([2]):

- *structured meshes*: the term "structured" refers to the way the grid information is addressed by the computer. In fact in a structured CFD grid, it is possible to build a function to turn the physical domain into a uniform Cartesian grid. In such a way it is easy to identify and access to a given point. It is used to speed CFD codes. On the other hand it is not allowed to set additional points into the grid if for any reason it is useful to have a more refined zone. The structured mesh utilized in this dissertation is a C-grid (fig. 1.1), named this way due to the shape similar to the letter "C" in which all the lines bend back to meet up with themselves at some point. In the analysed case the line that describes the airfoil

surface meets at the trailing edge, where the line leading from the lower surface of the airfoil coincides with the line from the upper surface. It is then clear that the mesh is finer around the airfoil and on its wake, where the highest gradients of variables are present. On the other hand in the far field from the airfoil wing effects on the flow can be neglected, and a strong refinement is not needed.

- *unstructured meshes*: Unstructured CFD grids don't have a direct mapping between the points stored in memory and their connections in the physical space. It means that the cell at a certain 'n' location in memory may not have physical relation to the cell next to it at location 'n+1'. This implies that the unstructured solver is more complex, in order to manage to correlate one point of the grid to another one. Nevertheless it allows to have more freedom to customize the grid with respect to the needed resolution in particular areas. An example of such a mesh is presented in fig. 1.2 for a *NACA 0012* airfoil.

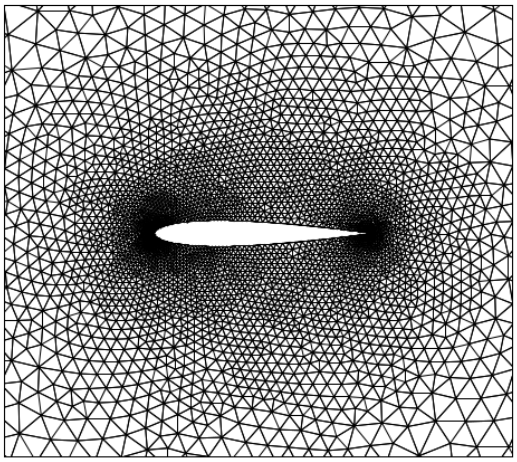


Figure 1.2: Example of unstructured mesh

The number of cells of the chosen mesh is 8192, 256 in the x - direction and 32 in the z - direction. As previously said, the y - direction is implemented with only 2 nodes to form 1 false $3D$ cell.

Chapter 2

Time Spectral Method

The Time Spectral Method has been introduced in CFD from turbomachinery developers trying to reduce the computational time needed for simulations of internal flows, starting from 2D and non viscous flows, to arrive to multi-stage 3D viscous flows and external fluid dynamics.

In fact it allows to skip the transient part of a standard unsteady simulation (time domain based, in which every time step depends on the previous one), and solves only the established motion, thanks to the passage back and forth to and from the frequency domain.

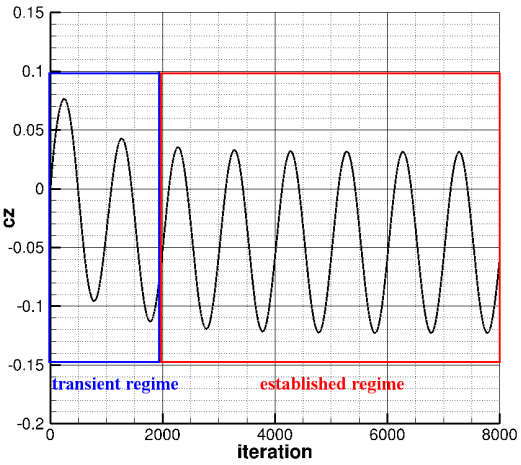
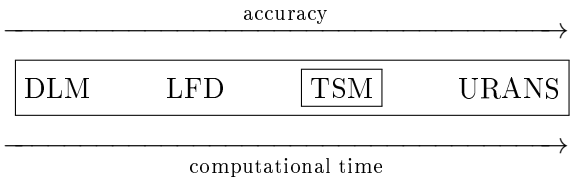


Figure 2.1: Example of unsteady computation performed by elsA

The Time Spectral Method is very useful to characterize and predict the response of non-linear dynamic systems subject to periodic oscillations, both auto-induced and due to a harmonic excitation.

In the context of aerodynamic analysis of the flow around moving surfaces, the Time Spectral Method is placed between LFD (*Linearized Frequency Domain*) Navier Stokes or Euler Equations ([3]) and the URANS (*Unsteady Reynolds - Averaged Navier Stokes*) concerning the realism of the physical non-linearities of the flow. The term URANS is improperly used in this case to indicate an Unsteady Euler simulation, without the contribution of viscosity and thermal conductivity.



In fact the basic state of linearized methods is a steady flow, which returns the same results as a RANS (steady) computation, while TSM's flow is averaged in time, and its results are different from a steady flow. In the LFD case, the flow is linearized around a fixed position of the geometry while the TSM preserves the non-linearities of the flow.

The calculations URANS correspond to a uniform sampling of the period and are integrated subsequently, while TSM can be considered a method parallel in time.

URANS can become too computationally expensive while increasing the size of the problem, but LFD cannot solve dynamic non-linearities, but only the steady ones: in this context TSM are introduced to save computational time with respect to URANS without losing the property to take into account all the non-linearities of the flow.

Lots of TSM approaches are given by the literature. Two of them are proposed in the following section, one that solves the equations in the frequency domain and one in the time domain. The resolution in the time domain will be then adopted.

2.1 Resolution in the frequency domain

The method *Non - Linear Frequency Domain* (NLFD) by McMullen *et al.* [4] turns the unsteady equations in the temporal domain into several steady problems in the frequency domain, extending the discussion of time averaged fluxes with the additional presence of the fundamental frequency of a perturbation.

In order to relax the notation, the problem will be treated as uni-dimensional (conservative variables W and R as scalar instead of vectors), and the concatenation of instants or frequencies that will soon be explained will be noted with bold characters (conservative variables $\mathbf{W}$ and residuals $\mathbf{R}$).

The unsteady equations, after being discretized with the finite volumes approach, can be written in the following form:

$$V\frac{\partial W}{\partial t} + R(W) = 0 \tag{2.1}$$

Under the hypothesis of time periodicity with pulsation ω of W and $R(W)$, the Fourier series of eq. 2.1 is:

$$\sum_{k=-\infty}^{\infty} \left(ik\omega V\widehat{W_k} + \widehat{R_k} \right) e^{ik\omega t} = 0 \tag{2.2}$$

where $\widehat{W_k}$ and $\widehat{R_k}$ represent the Fourier coefficients for W and $R(W)$ for a mode k. Since the complex exponential family forms an orthogonal basis, the only way for eq. 2.2 to be true is that the contribution of each mode k is zero.

$$ik\omega V\widehat{W_k} + \widehat{R_k} = 0 \qquad \forall k \in \mathbb{Z} \tag{2.3}$$

An infinite number of steady equations is then obtained (eq. 2.3) in the frequency domain.

Because of evident practical reasons, only a subset of modes, symmetric around 0, up to mode N, is considered , $-N \leq k \leq N$ in the frequency domain, and the Non-Linear Frequency Domain method can be used.

$$ik\omega V\widehat{W_k} + \widehat{R_k} = 0, \qquad k \in [-N, N] \tag{2.4}$$

In matrix form, eq. 2.4 can be written as:

$$i\underline{\underline{\mathbf{K}}}\omega V\widehat{\mathbf{W}} + \widehat{\mathbf{R}} = 0, \tag{2.5}$$

with $\underline{\underline{\mathbf{K}}}$, diagonal matrix with the $2N+1$ terms corresponding to harmonics and the bold vector corresponding to the concatenation of frequencies:

$$\underline{\underline{\mathbf{K}}} = \mathrm{diag}(-N, ..., N) \tag{2.6}$$

$$\widehat{\mathbf{W}} = (\widehat{W_{-N}}, \widehat{W_{1-N}}, ..., \widehat{W_N})^T, \quad \widehat{\mathbf{R}} = (\widehat{R_{-N}}, \widehat{R_{1-N}}, ..., \widehat{R_N})^T \tag{2.7}$$

To help integrate numerically the result, a pseudo-time derivative is added:

$$V\frac{\partial\widehat{W_k}}{\partial\tau_k} + ik\omega V\widehat{W_k} + \widehat{R_k} = 0, \qquad k \in [-N, N] \tag{2.8}$$

The term $ik\omega V\widehat{W_k}$ seems to be a supplementary source term to the transport equations. Therefore the method consists in the simultaneous resolution of the system of $2N+1$ steady equations.

These equations are not independent due to the non linearity of the operator $R(W)$: indeed $\widehat{R}_k$ cannot be computed directly from $\widehat{W_k}$, but only from the residuals in the time domain $R(W)$ for different time steps in the period.

A short overview of the NLFD method consists in:

- The Fourier coefficients of the conservative variables $\widehat{W_k}$ are known for the iteration q.
- The conservative variables $W(t)$ are calculated with the Inverse Discrete Fourier Transform, assuring the coupling of all the steady equations.
- The residuals $R(t)$ are calculated.
- The Discrete Fourier Transform is applied to the residuals and $\widehat{R}_k$ are obtained.
- At this point it is possible to make the convergence iteration, starting from the first item of the list, calculate the Fourier coefficient of the conservative variables at the iteration q, $\widehat{W}_k^{q+1}$

A continuous going back and forth to the frequency domain in needed here.

2.2 Resolution in the time domain

An alternative is the Time Spectral Algorithm, that proposes to solve the governing equations in the time-domain, considerably gaining on the computational cost required to transform back

and forth to the frequency domain. The method is also easy to implement in an existing solver in the time domain as *elsA* without having to deal with complex arithmetic.

For these purposes the *Time Spectral Method* has been introduced, or *Harmonic Balance Technique*, that in this discussion will denote the same method, to solve the steady equations in the time domain.

Hall *et al.* [5] proposed to apply a IDFT on the $2N+1$ equations 2.8, and the same number of equations in the temporal domain is obtained. Time is now discretized, and the new equations correspond to uniformly spaced instants into the period:

$$\mathbf{t} = (t_0, t_1, ..., t_{2N})^T \tag{2.9}$$

$$\tag{2.10}$$

hence $\mathbf{t}$ is the discrete, steady-state period, defined over $2N+1$ time instances, and can be conceptualized as well as a vector of length $2N+1$.

It is chosen to take the instants starting from zero, with regular intervals, making true the useful relationship:

$$\frac{t_j}{T} = \frac{j}{2N+1} \tag{2.11}$$

It is applied an Inverse Discrete Fourier Transform to the $2N+1$ equations 2.3, and the same number of equations in the time domain is obtained. The Fourier coefficients of R and W can be obtained by DFT this way:

$$\widehat{\mathbf{W}} \approx \underline{\underline{\varepsilon}}\mathbf{W} \tag{2.12}$$

$$\widehat{\mathbf{R}} \approx \underline{\underline{\boldsymbol{\varepsilon}}}\mathbf{R} \tag{2.13}$$

where the bold characters denote the concatenation operator of time instants and harmonics:

$$\mathbf{W} = (W_0, W_1, ..., W_{2N})^T, \quad \mathbf{R} = (R_0, R_1, ..., R_{2N})^T \tag{2.14}$$

and $\underline{\underline{\boldsymbol{\varepsilon}}}$ is the matrix corresponding to the discrete Fourier transform operator:

$$\varepsilon_{n,k} = \frac{1}{2N+1} e^{\frac{-2\pi ikn}{2N+1}}, \qquad \varepsilon_{n,k}^{-1} = e^{\frac{2\pi ikn}{2N+1}} \tag{2.15}$$

Substituting into the $2N+1$ equations 2.5 the approximation of complete amplitudes $\widehat{\mathbf{W}}$ and $\widehat{\mathbf{R}}$ in eq. 2.13, and coming back in the time domain via IDFT, $\widehat{R}$ becomes again the exact operator R for bijection of IDFT and the first term becomes a source term coupling all instants.

$$\underline{\underline{\boldsymbol{\varepsilon}}}\mathbf{R}(\mathbf{W}) + i\omega V \underline{\underline{\underline{\mathbf{K}}}}\underline{\underline{\boldsymbol{\varepsilon}}}\mathbf{W} = 0 \tag{2.16}$$

$$\mathbf{R}(\mathbf{W}) + i\omega V \underline{\underline{\varepsilon}}^{-1}\underline{\underline{\underline{\mathbf{K}}}}\underline{\underline{\varepsilon}}\mathbf{W} = 0, \tag{2.17}$$

$$\mathbf{R}(\mathbf{W}) + i\omega V \underline{\underline{\varepsilon}}^{-1}\underline{\underline{\underline{\mathbf{K}}}}\underline{\underline{\varepsilon}}\mathbf{W} = 0 \tag{2.18}$$

The term $i\omega\underline{\underline{\boldsymbol{\varepsilon}}}^{-1}\underline{\underline{\underline{\underline{\mathbf{K}}}}}\,\underline{\underline{\boldsymbol{\varepsilon}}}$ is identified as the time-spectral operator which, after multiplied, gives the source term coupling all the instants

$$\mathbf{S} = \underline{\underline{\underline{\underline{\mathbf{D_t}}}}}\mathbf{W} = i\omega\underline{\underline{\boldsymbol{\varepsilon}}}^{-1}\underline{\underline{\underline{\underline{\mathbf{K}}}}}\,\underline{\underline{\boldsymbol{\varepsilon}}}\mathbf{W} \approx \frac{\partial \mathbf{W}}{\partial t} \tag{2.19}$$

Gopinath et Jameson [6] explicit the source term developing the matrix formulation of Hall *et al.* in eq. 2.19 for each instant.

The final system to solve in the time domain is:

$$\mathbf{R}(\mathbf{W}) + V\underline{\underline{\underline{\underline{\mathbf{D_t}}}}}\mathbf{W} = 0 \tag{2.20}$$

that, after adding a pseudo-time derivative τ for the numerical convergence, becomes:

$$\boxed{V\frac{\partial \mathbf{W}}{\partial \tau} + \mathbf{R}(\mathbf{W}) + V\underline{\underline{\underline{\underline{\mathbf{D_t}}}}}\mathbf{W} = 0} \tag{2.21}$$

Being each of the $2N+1$ steady equations to solve, independent one from each other (for one iteration step), computational power can be saved by solving each instant separately (and at the end of the iteration coupling them updating the source term). For the $n-th$ instant the final equation is:

$$V\frac{\partial W_n}{\partial \tau_n} + R(W_n) + V\mathbf{S}_n = 0, \;\; 0 \leq n < 2N+1 \tag{2.22}$$

with W_n the conservative variables at the n^{th} instant and S_n the source term, $\mathbf{S} = \underline{\underline{\underline{\underline{\mathbf{D_t}}}}}\mathbf{W} = (S_0, S_1, ..., S_{2N})^T$, at the $n-th$ instant which couples all the instant, being $\mathbf{W} = (W_{n=1}, W_{n=2}, \ldots, W_{n=2N+1})$.

The operator $\underline{\underline{\underline{\underline{\mathbf{D}_t}}}}$ is a time discretization scheme, centered and high order.

The formulation of the time-spectral operator matrix, in an easy form to implement computationally, is derived in appendix A; it is an antisymmetric matrix, with the null main diagonal, and dimensions $(2N+1)\times(2N+1)$. All the work will be based on the time-spectral operator derived for an odd number of instants. In fact the time-spectral matrix for an even number of instants turns out to be unstable due its two zeros eigenvalues.

$$\underline{\underline{\underline{\underline{\mathbf{D}_t}}}}(i,j) = \begin{cases} \dfrac{2\pi}{T}\dfrac{1}{2}(-1)^{i-j}\csc\left[\dfrac{\pi}{2N+1}(i-j)\right] & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases} \tag{2.23}$$

Hence the TSM method consists in solving $2N+1$ steady relations in the time domain, with a source term that couples all the instants. According to the Nyquist-Shannon sampling criterion, $2N+1$ evenly-spaced time instants allow to capture the N^{th} harmonic of the fundamental frequency at most. Consequently, the TSM computations will be classified according to the number of instants $2N+1$, or to the number of harmonics potentially captured N. In the figure below the principle of TSM is described: starting from a null motion condition, a set of steady problems corresponding to the sampled instants is solved at each q iteration. When the whole process has converged, all the solutions (variables, fluxes and coefficients) reach the value of the corresponding instant in an unsteady computation.

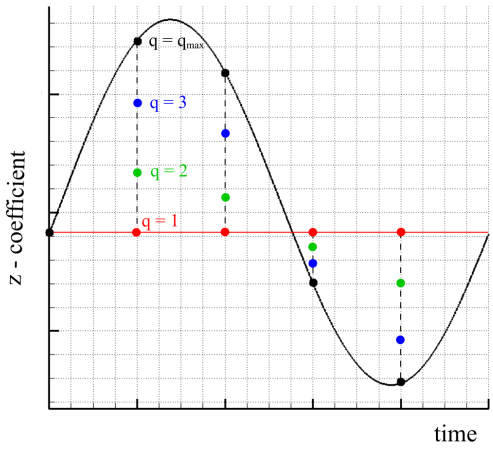


Figure 2.2: Principle of TSM - example with z-coefficient

2.2.1 Example of efficiency of the time spectral matrix

It is shown how powerful the time spectral matrix is in approximating the time derivative, depending on the complexity of the periodic function to approximate. Here the perfect approximation of the derivative of a sinusoidal signal (cosine) is shown, by only 3 instants, i.e. only one harmonic.

$$w_1 = \widehat{w_1} \sin(x) \tag{2.24}$$

$$\frac{\mathrm{d}w_1}{\mathrm{d}t} = \widehat{w_1} \cos(x) \tag{2.25}$$

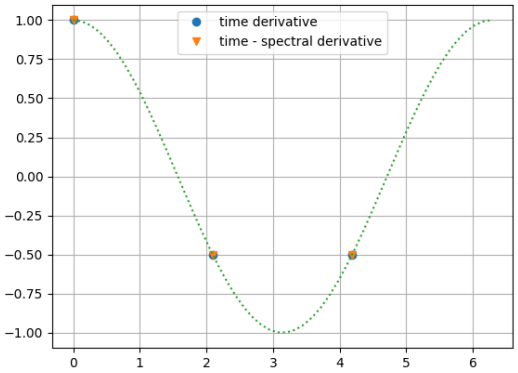


Figure 2.3: Approximation of derivatives by time-spectral operator of the function w_1

In order to be able to address more complex functions, it is sufficient to increase the number of harmonics (and instants sampled), to have the right compliance between time-spectral results and the analytical time derivative of the function.

To prove it, a more complex function (eq. 2.26) is chosen, which has eq. 2.27 as derivative.

$$w_2 = \widehat{w_2} \cdot \cos(2x) \cdot \sin(5x) \tag{2.26}$$

$$\frac{\mathrm{d}w_2}{\mathrm{d}t} = \widehat{w_2} \cdot (-2\sin(2x) \cdot \sin(5x) + 5\cos(2x) \cdot \cos(5x)) \tag{2.27}$$

In figure 2.4, the plot at the left shows how 9 instants (4 harmonics) are not enough to approximate in a satisfying way the derivative. While at the right a 15 instant sampling (7 harmonics) is proposed, showing perfect compliance between the time derivative and the time-spectral derivative.

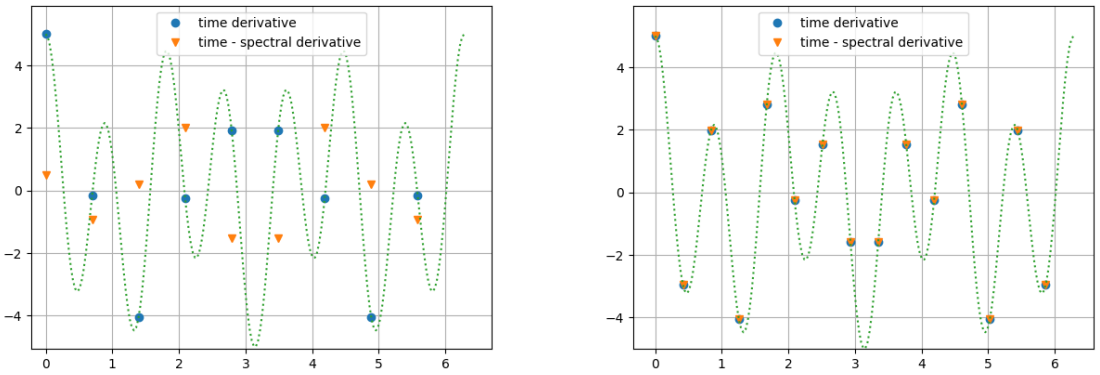


Figure 2.4: Approximation of derivatives by time-spectral operator of the function w_2 with 9 and 15 instants (4 and 7 harmonics)

Chapter 3

Fluid mechanic equations - Euler equations

In this chapter a more detailed explanation about the nature of the fluid mechanic equations will be investigated, especially to finally understand the nature of the conservative variables and fluxes (residuals) used in the TSM equations, referred to the specific problem of a moving airfoil with respect to the absolute frame of reference attached to the earth.

The case of Euler equations neglects the effects of the viscosity and thermal conductivity of the fluid, which are included in the Navier-Stokes equations. A solution of the Euler equations is therefore only an approximation of a real fluid problem. For some problems, like the lift of a thin airfoil at low angle of attack, a solution of the Euler equations provides a good model of reality. For other problems, like the growth of the boundary layer on a flat plate, the Euler equations do not properly model the problem. In these cases, other models should be implemented to take into account the viscous effects, as for examples the following:

- Direct Numerical Simulation - DNS, which solves all the scales of turbulence thanks to an extremely refined mesh.
- Large Eddy Simulation - LES, which solves the scales of turbulence up to a certain threshold below which several models can be exploited to shape turbulence in the unresolved scales.
- Reynolds Averaged Navier Stokes - RANS, based on a statistical approach to Navier-Stokes equations.

Being the study case a thin airfoil at low angle of attack, the hypothesis of no detachment of the boundary layer is realistically respected, and Euler equations are considered enough reliable to catch the reality of the phenomenon, making affordable the cost of simulations in terms of required memory and time.

Besides, the airfoil is in pitching motion, and the continuous change of frame of reference should be taken into account in the equations.

Different types of problem can be studied: from the *fixed block* formulation, in which no motion is considered, to arrive to *mobile block* formulation, in which the airfoil is fixed and the mesh is rigidly moving, and the *Arbitrary Lagrangian Eulerian* formulation, in which the airfoil is moving and the mesh is deforming accordingly to the motion of the airfoil.

The solver will use the mobile block approach, but other choices will need to be done before, as the frame of reference in which to express the equations of motion, absolute or relative, and the

frame of reference in which to express the velocities, absolute or relative.

In order to study a fluid dynamic phenomenon, the fluid equations can be expressed in three formulations:

- *absolute velocity* expressed in a *absolute frame of reference* $\mathcal{R}_\mathcal{A}$: this is well suited for example for the flow around a model placed in a wind tunnel, while in the case of a rotating body, the problem of the choice of both the frames of reference for the expression of the vector quantities and for the physical principles of the fluid equations is raised.
- *relative velocity* expressed in a *relative frame of reference* $\mathcal{R}_\mathcal{R}$: choosing a relative frame of reference leaves the choice of the frame of reference used to measure the speed of the fluid. The formulation of the laws of motion in a relative frame of reference can be advantageous for example for a body in uniform translation and/or rotation, because the volume of the fluid considered around the body is fixed in $\mathcal{R}_\mathcal{R}$ and it is independent of time, defining though a steady problem. Concerning the frame of reference in which to express the velocities, the most natural approach would be to use the same frame of reference as $\mathcal{R}_\mathcal{R}$, formulation usually used for turbomachinery internal flow problems. On the other hand this approach can become unsuccessful for large peripheral speeds, as in the case of helicopter rotors or propellers, in which the too large coarsened cells, positioned far from the wall, can reveal inappropriate to obtain a right flow balance.
- *absolute velocity* expressed in a *relative frame of reference* $\mathcal{R}_\mathcal{R}$: this is the most appropriate formulation to describe the flow field around an airfoil under a prescribed pitching motion. In this case, in which the body is experiencing a steady motion (uniform translation and/or rotation), the resulting problem is steady with respect to the moving frame.

A simulation aimed at such a computation, performed as a mobile block problem, uses an Eulerian approach to the fluid motion, that is a problem with a rigid and fixed mesh. The model (the airfoil in the case of this discussion) is fixed, while the computational domain, with its boundaries, rotate rigidly, leading to a rotation of the inlet boundary conditions instead of a rotation of the model itself. The general approach to perform such a computation is to attach a frame to the body (body frame) and express the fluid equations of motion with respect to that moving frame, called $\mathcal{R}_\mathcal{R}$, and the velocities with respect to the absolute, inertial, frame of reference $\mathcal{R}_\mathcal{A}$.

3.1 *Absolute frame of reference* formulation

The differential and integral forms of Euler equations are presented in this section.

The hypothesis of continuous medium allows us to consider an infinitesimal element of constant volume $\mathrm{d}V$ and of mass equal to $\delta m = \rho\,\mathrm{d}V$ in the frame of reference $\mathcal{R}_\mathcal{A}$.

The equations of conservation of mass, momentum and energy for the infinitesimal volume of the fluid, fixed into the space, permit to obtain the differential Euler equations Euler in the conservative form.

$$\begin{cases} \dfrac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{U}) = 0 \\ \dfrac{\partial \rho \mathbf{U}}{\partial t} + \nabla \cdot (\rho \mathbf{U} \otimes \mathbf{U}) = \nabla \cdot (-p\underline{\underline{\mathbf{I}}}) \\ \dfrac{\partial \rho E}{\partial t} + \nabla \cdot (\rho E \mathbf{U}) = \nabla \cdot (-p\mathbf{U}) \end{cases} \tag{3.1}$$

The passage to the integral form of the system of Euler equations is obtained by integration of each of the differential equations over the volume V independent of time.

Applying the Green - Ostrograski theorem it is possible to rewrite the equations, being S the interface surface of each computational volume V, transforming the volume integral of the second term into a surface integral. Using the transport theorem and the the independence of V with respect to time, the time derivative can be put out of the integral, due to the property of the volume taken constant with respect to time.

$$\begin{cases} \dfrac{\partial}{\partial t}\displaystyle\int_V \rho \,\mathrm{d}V + \int_S \rho \mathbf{U} \cdot \mathbf{n} \,\mathrm{d}S = 0 \\ \dfrac{\partial}{\partial t}\displaystyle\int_V \rho \mathbf{U} \,\mathrm{d}V + \int_S \left(\rho \mathbf{U} \otimes \mathbf{U} + p\underline{\underline{\mathbf{I}}}\right) \cdot \mathbf{n} \,\mathrm{d}S = 0 \\ \dfrac{\partial}{\partial t}\displaystyle\int_V \rho E \,\mathrm{d}V + \int_S \left(\rho E \mathbf{U} + p\underline{\underline{\mathbf{I}}}\mathbf{U}\right) \cdot \mathbf{n} \,\mathrm{d}S = 0 \end{cases} \tag{3.2}$$

The problem can be generalized this way. The generalized conservative form of the three conservative equations is:

$$\frac{\partial \mathbf{W}}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{W}) = 0 \tag{3.3}$$

and applying the Green - Ostrograski theorem:

$$\frac{\partial}{\partial t}\int_V \mathbf{W} \,\mathrm{d}V + \int_S \mathbf{F}(\mathbf{W}) \cdot \mathbf{n} \,\mathrm{d}S = 0 \tag{3.4}$$

with the conservative variables and the convective fluxes expressed as:

$$\mathbf{W} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{pmatrix}, \qquad \mathbf{F}_{c_x} = \begin{pmatrix} \rho u \\ \rho u u + p \\ \rho u v \\ \rho u w \\ (\rho E + p) u \end{pmatrix}, \qquad \mathbf{F}_{c_y} = \begin{pmatrix} \rho v \\ \rho u v \\ \rho v v + p \\ \rho w v \\ (\rho E + p) v \end{pmatrix}, \qquad \mathbf{F}_{c_z} = \begin{pmatrix} \rho w \\ \rho u w \\ \rho v w \\ \rho w w + p \\ (\rho E + p) w \end{pmatrix} \tag{3.5}$$

For a 2D problems defined in x and z directions, the equation 3.4 becomes:

$$\frac{\partial}{\partial t}\int_S \mathbf{W} \,\mathrm{d}S + \int_{\delta \mathcal{S}} \left(\mathbf{F}_{c_x} \,\mathrm{d}\mathbf{z} + \mathbf{F}_{c_z} \,\mathrm{d}\mathbf{x}\right) = 0 \tag{3.6}$$

where the volume has become a surface and the surface has become a length, and the conservative variables and fluxes are:

$$\mathbf{W} = \begin{pmatrix} \rho \\ \rho u \\ \rho w \\ \rho E \end{pmatrix}, \qquad \mathbf{F}_{c_x} = \begin{pmatrix} \rho u \\ \rho u u + p \\ \rho u w \\ (\rho E + p) u \end{pmatrix}, \qquad \mathbf{F}_{c_z} = \begin{pmatrix} \rho w \\ \rho u w \\ \rho w w + p \\ (\rho E + p) w \end{pmatrix} \tag{3.7}$$

Unfortunately these equations are valid for an absolute frame of reference, which if applied in the case of study would lead to an unsteady motion that would not allow some simplifications presented later.

It seems necessary to introduce a rotation of the frame of reference to analyse a pitching airfoil in a block mobile approach.

3.2 *Relative frame of reference* formulation

The general approach is to attach a frame to the moving body (often called the body frame) and to express the fluid equations of motion with respect to that frame. When the body is experiencing a steady motion (uniform translation and/or rotation) the resulting problem is steady with respect to the moving frame. The general kinematics of a rigid body with respect to a moving frame is here introduced.

Considered an absolute Galilean frame (also called absolute frame) of reference $\mathcal{R}_{\mathcal{A}}(O, \mathbf{e_1}, \mathbf{e_2}, \mathbf{e_3})$ (e = "earth") and a moving frame $\mathcal{R}_{\mathcal{R}}(O, \mathbf{b_1}, \mathbf{b_2}, \mathbf{b_3})$ (b = "body"), both equipped with an Euclidean Cartesian coordinate system, for any point M, the position vector can be expressed in any of these frames with different coordinates:

$$\mathbf{OM} = x_0 + x_i \mathbf{e}_i = x_0 + x'_i \mathbf{b}_i, \quad i = 1, 2, 3 \tag{3.8}$$

The transformation from $\mathbf{e}_i$ to $\mathbf{b}_i$ and vice versa is a rotation associated to the orthogonal matrix $\underline{\underline{\mathbf{R}}} = \mathbf{b}_k \otimes \mathbf{e}_k$ or equivalently $R_{ij} = \mathbf{e}_i^T \mathbf{b}_j$.

The rotated coordinate vector $\mathbf{x}'$, independent of time, is related to $\mathbf{x}$ though the relations $\mathbf{x}(t) = \underline{\underline{\mathbf{R}}}\mathbf{x}'$ and $\mathbf{x}' = \underline{\underline{\mathbf{R}}}^T \mathbf{x}(t)$.

It is supposed now that the moving block is animated with a uniform translation and is also rotating steadily with an angular velocity vector $\mathbf{\Omega}_{R_R/R_A}$. The location of a point M, with fixed coordinates $\mathbf{x}'$ in the moving frame $\mathcal{R}_{\mathcal{R}}$, is written as:

$$\mathbf{OM}(t) = \mathbf{OM_0}(t) + \mathbf{M_0M} = \mathbf{OM_0}(t) + x'_i \mathbf{b_i} \tag{3.9}$$

with $\mathbf{OM_0}$ the instantaneous position of the moving frame origin with respect to the absolute frame. By definition, the velocity is given by:

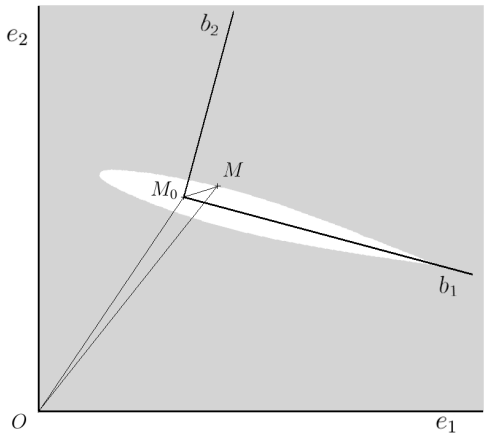


Figure 3.1: Frames of reference, absolute $(e_1 - e_2)$ and relative $(b_1 - b_2)$

$$\mathbf{U} = \frac{\mathrm{d}\mathbf{OM}(t)}{\mathrm{d}t} = \frac{\mathrm{d}\mathbf{OM_0}(t)}{\mathrm{d}t} + x_i' \frac{\mathrm{d}\mathbf{b}_i}{\mathrm{d}t} \tag{3.10}$$

$$\mathbf{U} = \mathbf{U}_e^T + \mathbf{\Omega}_{R_R/R_A} \times x_i' \mathbf{b}_i = \mathbf{U}_e^T + \mathbf{\Omega}_{R_R/R_A} \times \mathbf{M_0M} \tag{3.11}$$

$$\mathbf{U} = \mathbf{U}_e^T + \mathbf{U}_e^R \tag{3.12}$$

where $\mathbf{U}_e^T = \dfrac{\mathrm{d}\mathbf{OM_0}(t)}{\mathrm{d}t}$ and $\mathbf{U}_e^R = \mathbf{\Omega}_{R_R/R_A} \times x_i' \mathbf{b}_i$ represent respectively the translational and rotational speeds due to the motion of the frame.

In terms of absolute coordinates we can write the current position vector, in order to relate the coordinates of $\mathbf{\Omega_{R_R/R_A}}$ to the rotation matrix $\underline{\underline{\mathbf{R}}}$:

$$\mathbf{OM}(t) = \mathbf{x_0}(t) + \underline{\underline{\mathbf{R}}}(t)\mathbf{x}' \tag{3.13}$$

$$\frac{\mathrm{d}\mathbf{OM}(t)}{\mathrm{d}t} = \frac{\mathrm{d}\mathbf{x_0}(t)}{\mathrm{d}t} + \frac{\mathrm{d}\underline{\underline{\mathbf{R}}}(t)}{\mathrm{d}t}\mathbf{x}' = \frac{\mathrm{d}\mathbf{x_0}(t)}{\mathrm{d}t} + \frac{\mathrm{d}\underline{\underline{\mathbf{R}}}(t)}{\mathrm{d}t}\underline{\underline{\mathbf{R}}}^T\mathbf{x} = \frac{\mathrm{d}\mathbf{x_0}(t)}{\mathrm{d}t} + \left[\underline{\underline{\mathbf{W}}}_{R_A}\right]\mathbf{x} \tag{3.14}$$

in which the skew matrix $[\underline{\underline{\mathbf{W}}}]_{R_A} = \dfrac{\mathrm{d}\underline{\underline{\mathbf{R}}}(t)}{\mathrm{d}t}\underline{\underline{\mathbf{R}}}^T$ has been introduced.

Being $\underline{\underline{\mathbf{R}}}$ an orthogonal matrix, $\underline{\underline{\mathbf{R}}}\,\underline{\underline{\mathbf{R}}}^T = \underline{\mathbf{I}}$ and differentiating with respect to time gives

$$\frac{\mathrm{d}\underline{\underline{\mathbf{R}}}}{\mathrm{d}t}\underline{\underline{\mathbf{R}}}^T + \underline{\underline{\mathbf{R}}}\frac{\mathrm{d}\underline{\underline{\mathbf{R}}}^T}{\mathrm{d}t} = [\underline{\underline{\mathbf{W}}}] + [\underline{\underline{\mathbf{W}}}^T] = 0 \tag{3.15}$$

which shows that $[\underline{\underline{\mathbf{W}}}]$ is skew. Defining $\mathbf{\Omega}_{R_R/R_A} = \omega_i \mathbf{e}_i$, then the skew tensor $[\underline{\underline{\mathbf{W}}}]_{R_A}$ can be written as:

$$[\underline{\underline{\mathbf{W}}}]_{R_A} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}_{R_A} \tag{3.16}$$

Thus it is equivalent for the computation of the entrainment speed in rotation $\mathbf{U_e^R}(t)$ to write one of the following formulations:

$$\mathbf{U}_e^R = \mathbf{\Omega}_{R_R/R_A} \times \mathbf{M_0M} = [\underline{\underline{\mathbf{W}}}]\mathbf{x} \tag{3.17}$$

However in practice it is easier to use the angular velocity coordinates in the relative reference frame $\mathcal{R}_\mathcal{R}$. In order to do this it is necessary to rewrite $\mathbf{U}_e^R(t)$ using the components of the tensor $[\underline{\underline{\mathbf{W}}}]$ expressed in the moving coordinate frame $\mathcal{R}_\mathcal{R}$. Recalling the expression of the rotation velocity components in $\mathcal{R}_\mathcal{A}$,

$$\mathbf{U}_e^R|_{R_A} = \frac{\mathrm{d}\underline{\underline{\mathbf{R}}}(t)}{\mathrm{d}t}\underline{\underline{\mathbf{R}}}^T\mathbf{x} = [\underline{\underline{\mathbf{W}}}]_{R_A}\mathbf{x} \tag{3.18}$$

the components in the moving frame are obtained by right-multiplying by $\underline{\underline{\mathbf{R}}}^T$:

$$\mathbf{U}_e^R|_{R_R} = \underline{\underline{\mathbf{R}}}^T\mathbf{v}_e^R|_{R_A} = (\underline{\underline{\mathbf{R}}}^T[\underline{\underline{\mathbf{W}}}]_{R_A}\underline{\underline{\mathbf{R}}})\underline{\underline{\mathbf{R}}}^T\mathbf{x} = [\underline{\underline{\mathbf{W}}}]_{R_R}\mathbf{x}' \tag{3.19}$$

where the tensor coordinate transformation operation from basis vectors $\mathbf{e}_i$ to basis vectors $\mathbf{b}_i$:

$$[\underline{\underline{\mathbf{W}}}]_{R_R} = \underline{\underline{\mathbf{R}}}[\underline{\underline{\mathbf{W}}}]_{R_A}\underline{\underline{\mathbf{R}}}^T \tag{3.20}$$

The components of the skew matrix $[\underline{\underline{\mathbf{W}}}]_{R_R}$ then define the components of the pseudo angular velocity vector $\mathbf{\Omega_{R_R/R_A}}$ in $\mathcal{R}_\mathcal{R}$.

Equations

The conservation equations of motions are now modified with respect to the equations expressed in the absolute frame of reference. We need to take into account the entrainment speed of the domain, that is the speed to add due to the rotation of the computational domain, measured with respect to the reference $\mathcal{R}_\mathcal{A}$ but expressed in $\mathcal{R}_\mathcal{R}$, called $\mathbf{U}_{e_{|RR}}$. The absolute speed considered is $\mathbf{U}_{a_{|RR}}$, that is the velocity of the fluid measured with respect to the frame of reference $\mathcal{R}_\mathcal{A}$ but expressed in the relative frame of reference $\mathcal{R}_\mathcal{R}$.

A constant infinitesimal Cartesian element of volume $\mathrm{d}V_{|RR}$ is considered fixed in the reference R_R, with mass equal to $\delta m = \rho(\mathbf{OM}(t), t)\,\mathrm{d}V_{|RR}$ in $\mathcal{R}_\mathcal{R}$ for which the conservation of mass, momentum and energy are written in eq. 3.23.

The new convective fluxes for a 2D problem in eq. 3.6, become:

$$F_{c_x} = \begin{pmatrix} \rho(U_{a_{|RRx}} - U_{e_{|RRx}}) \\ \rho U_{a_{|RRx}}(U_{a_{|RRx}} - U_{e_{|RRx}}) + p \\ \rho U_{a_{|RRz}}(U_{a_{|RRx}} - U_{e_{|RRx}}) \\ (\rho E + p)(U_{a_{|RRx}} - U_{e_{|RRx}}) \end{pmatrix}, \qquad F_{c_z} = \begin{pmatrix} \rho(U_{a_{|RRz}} - U_{e_{|RRz}}) \\ \rho U_{a_{|RRx}}(U_{a_{|RRz}} - U_{e_{|RRz}}) \\ \rho U_{a_{|RRz}}(U_{a_{|RRz}} - U_{e_{|RRz}}) + p \\ (\rho E + p)(U_{a_{|RRz}} - U_{e_{|RRz}}) \end{pmatrix} \tag{3.21}$$

Moreover the total derivative of ρ with respect to time ($\frac{\mathrm{d}\rho}{\mathrm{d}t}$) introduces a term due to the absolute displacement of the point M' with respect to $\mathcal{R}_\mathcal{A}$ and a term representing the unsteady nature of the density field. This temporal derivation is also called *convective derivative* because it takes into account the movement of the vector $\mathbf{OM}(t)$ even if it is considered fixed in the frame of reference $\mathcal{R}_\mathcal{R}$, it is denoted:

$$\left(\frac{\delta\rho}{\delta t}\right)_{|RR} = \boldsymbol{\nabla}\rho \cdot \mathbf{U}_{e_{|RR}} + \frac{\partial\rho}{\partial t} \tag{3.22}$$

The use of the convective derivative is used equivalently for the momentum and the energy.

For the complete derivation of the equations of motion expressed in the relative frame of reference the reader is redirected to [7].

The system of Euler equations expressed in the relative fame of reference $\mathcal{R}_\mathcal{R}$ with the quantities measured with respect to the absolute frame of reference $\mathcal{R}_\mathcal{A}$, in the differential formulation, is:

$$\begin{cases} \left(\dfrac{\delta\rho}{\delta t}\right)_{|RR} + \boldsymbol{\nabla}\cdot\Big(\rho(\mathbf{U}_{a_{|RR}} - \mathbf{U}_{e_{|RR}})\Big) = 0 \\ \left(\dfrac{\delta\rho\mathbf{U}_{a_{|RR}}}{\delta t}\right)_{|RR} + \boldsymbol{\nabla}\cdot\Big(\rho\mathbf{U}_{a_{|RR}}(\mathbf{U}_{a_{|RR}} - \mathbf{U}_{e_{|RR}})\Big) = \boldsymbol{\nabla}\cdot\left(-p\underline{\underline{\mathbf{I}}}\right) - \mathbf{\Omega}_{|RR}\times\Big(\rho\mathbf{U}_{a_{|RR}}\Big) \\ \left(\dfrac{\delta\rho E}{\delta t}\right)_{|RR} + \nabla\cdot\Big(\rho E(\mathbf{U}_{a_{|RR}} - \mathbf{U}_{e_{|RR}})\Big) = \boldsymbol{\nabla}\cdot\Big(-p\underline{\underline{\mathbf{I}}}\cdot\mathbf{U}_{a_{|RR}}\Big) \end{cases} \tag{3.23}$$

To pass to the integral formulation, each of the differential equations is integrated on a volume V fixed in $\mathcal{R}_{\mathcal{R}}$ and undeformable in time, using the transport theorem together with the divergence theorem, and also exploiting the property of the null divergence of the entrainment speed characterizing a solid rotation.

The system expressed in the integral conservative form is hence:

$$\begin{cases} \dfrac{\partial}{\partial t}\displaystyle\int_V \rho \,\mathrm{d}V_{|RR} + \int_S \rho(\mathbf{U}_{a_{|RR}} - \mathbf{U}_{e_{|RR}}) \cdot \mathbf{n}\,\mathrm{d}S_{|RR} = 0 \\ \dfrac{\partial}{\partial t}\displaystyle\int_V \rho\mathbf{U}_{a_{|RR}}\,\mathrm{d}V_{|RR} + \int_S \rho\mathbf{U}_{a_{|RR}} \otimes \left(\mathbf{U}_{a_{|RR}} - \mathbf{U}_{e_{|RR}}\right) \cdot \mathbf{n}\,\mathrm{d}S_{|RR} = \\ \qquad\qquad = -\displaystyle\int_S p\underline{\underline{\mathbf{I}}} \cdot \mathbf{n}\,\mathrm{d}S - \int_V \boldsymbol{\Omega}_{|RR} \times \left(\rho\mathbf{U}_{a_{|RR}}\right) \cdot \mathrm{d}V \\ \dfrac{\partial}{\partial t}\displaystyle\int_{\mathcal{V}} \rho E\,\mathrm{d}V_{|RR} + \int_S \rho E \cdot \left(\mathbf{U}_{a_{|RR}} - \mathbf{U}_{e_{|RR}}\right) \cdot \mathbf{n}\,\mathrm{d}S_{|RR} = -\displaystyle\int_S p\underline{\underline{\mathbf{I}}} \cdot \mathbf{U}_{a_{|RR}} \cdot \mathbf{n}\,\mathrm{d}S_{|RR} \end{cases} \tag{3.24}$$

As can be easily noticed, an additional source term of Coriolis acceleration appears on the right-hand side of the momentum equations.

Inquimbert in [8] derives also the formulations of the problem expressed in the relative frame of reference with relative velocity.

Another formulation, developed combining the two classical algorithms of continuum mechanics used to describe motion, i.e. the Lagrangian description and the Euler one, is the *Arbitrary Lagrangian-Eulerian*. This formulation was developed in order to make the most of advantages, minimizing the drawbacks of both the methods. Here the mesh is not fixed any more, but the airfoil moves and subsequently the mesh is deformed. A detailed description of *ALE* formulation in given by Donea *et al.* [9].

Chapter 4

Solution methods of linear systems

Each TSM steady coupled equation constitutes a non-linear system to solve. Using implicit methods we have to finally solve linear systems, and several ways to do it are listed in the following. The easy LU decomposition is used in the main code that has been implemented, but in following updates of the code a more robust GMRES solver will be implemented (the reader is readdressed to the book by Saad [10] for a detailed explanation of the method).

4.1 Direct solvers

4.1.1 LU decomposition

The problem to solve is the following, [11] : given an $n \times n$ matrix A, called *coefficient matrix* and a real n-vector $\mathbf{b}$, called *right hand side* vector, the aim is to find the vector $\mathbf{x}$ belonging to $\mathbb{R}^{\kappa}$ called *vector of unknowns*, such that:

$$\underline{\underline{\mathbf{A}}}\mathbf{x} = \mathbf{b} \tag{4.1}$$

The LU factorization decomposes the input matrix A into the product LU, with proper row and/or column orderings and permutations, where $\underline{\underline{\mathbf{L}}}$ is a lower triangular matrix with diagonal elements equal to one and $\underline{\underline{\mathbf{U}}}$ is an upper triangular matrix.

The LU factorization is performed transforming the starting system $\underline{\underline{\mathbf{A}}}\mathbf{x} = \mathbf{b}$ to $\underline{\underline{\mathbf{L}}}\,\underline{\underline{\mathbf{U}}}\mathbf{x} = \mathbf{b}$, and subsequently the problem is split in two problems:

$$\begin{cases} \underline{\underline{\mathbf{L}}}\mathbf{y} = \mathbf{b} \\ \underline{\underline{\mathbf{U}}}\mathbf{x} = \mathbf{y} \end{cases} \tag{4.2}$$

In this case we are dealing with triangular matrices, $\underline{\underline{\mathbf{L}}}$ and $\underline{\underline{\mathbf{U}}}$), which can be solved directly by forward and backward substitution without using the Gaussian elimination process (however we do need this process or equivalent to compute the LU decomposition itself).

With this method described previously, the factorization will fail when a 0 is found on the diagonal of the matrix, leading to a division by zero. Also if elements of small magnitude are on the diagonal, entries on the triangular factors will grow significantly and the system may diverge. Because of these problems, in order to make the factorization numerically more stable, the pivoting method, i.e. the reordering of rows and/or columns are introduced.

4.1.2 Incomplete LU decomposition

It consists in the use in the solution process of an approximate matrix $\widetilde{\underline{\underline{\mathbf{A}}}}$ in place of the exact matrix $\underline{\underline{\mathbf{A}}}$, obtained with an *LU* factorization that takes into account two approximate matrices $\widetilde{\underline{\underline{\mathbf{L}}}}$ and $\widetilde{\underline{\underline{\mathbf{U}}}}$ which, multiplied, give an approximation of the input matrix $\underline{\underline{\mathbf{A}}}$ ([12]).

$$\widetilde{\underline{\underline{\mathbf{A}}}} = \widetilde{\underline{\underline{\mathbf{L}}}}\widetilde{\underline{\underline{\mathbf{U}}}} \sim \underline{\underline{\mathbf{A}}} \tag{4.3}$$

$$\tag{4.4}$$

where in fact $\underline{\underline{\mathbf{A}}}$ is:

$$\underline{\underline{\mathbf{A}}} = \widetilde{\underline{\underline{\mathbf{L}}}}\widetilde{\underline{\underline{\mathbf{U}}}} + \underline{\underline{\mathbf{E}}} \tag{4.5}$$

$$\tag{4.6}$$

and $\underline{\underline{\mathbf{E}}}$ is an error matrix.

The new problem to solve is:

$$\widetilde{\underline{\underline{\mathbf{A}}}}\mathbf{x} = \mathbf{b} \tag{4.7}$$

$$\begin{cases} \widetilde{\underline{\underline{\mathbf{L}}}}\mathbf{y} = \mathbf{b} \\ \widetilde{\underline{\underline{\mathbf{U}}}}\mathbf{x} = \mathbf{y} \end{cases} \tag{4.8}$$

This method is rather easy and inexpensive to compute, but on the other hand it could lead to an approximation that is often too crude and may require too many iterations or may even diverge.

In general, the more accurate the ILU factorization, the fewer the iterations required to converge, but at the same time the computational cost of the whole solving process is increased, since the factorization is closer to the exact LU.

In order to truncate $\underline{\underline{\mathbf{L}}}$ and $\underline{\underline{\mathbf{U}}}$, first of all it is necessary to define a fill-in factor which is a measure of the desired degree of the density of the matrices $\underline{\underline{\mathbf{L}}}$ and $\underline{\underline{\mathbf{U}}}$ and will damp to zero the elements that satisfies a certain criterion of smallness (obviously the more terms are dropped to zero in order to make the matrix sparser the more approximated is the matrix $\widetilde{\underline{\underline{\mathbf{A}}}}$ and the solution $\mathbf{x}$.)

4.2 Iterative solvers

The basic iterative methods used to solve large linear systems are based on relaxation of the coordinates ([10]). Starting with a given initial guess of the solution, these methods modify the components of the approximate solution, one or a few at a time in a certain order, depending on the method, until convergence is reached. Each of these modifications, called relaxation steps, is aimed at decreasing to zero one or a few components of the residual vector. These techniques are rarely used separately but combined with more efficient methods as *Krylov Subspace Method* (as the GMRES method - Generalized Minimal RESidual method), helped by preconditioning techniques to speed up the convergence and improve the robustness.

Given the same problem as the one explained for direct solvers,

$$\underline{\underline{\mathbf{A}}}\mathbf{x} = \mathbf{b} \tag{4.9}$$

the aim is to find the final vector solution x in a way involving passing from one iterate to the next by modifying one or a few components of the approximate vector solution at a time. The most used criterion to achieve improvement over the iterations is the annihilation of some components of the residual vector $\mathbf{b} - \underline{\underline{\mathbf{A}}}\mathbf{x}$.

The matrix A can be decomposed as:

$$\underline{\underline{\mathbf{A}}} = \underline{\underline{\mathbf{D}}} - \underline{\underline{\mathbf{E}}} - \underline{\underline{\mathbf{F}}} \tag{4.10}$$

in which $\underline{\underline{\mathbf{D}}}$ is the diagonal of $\underline{\underline{\mathbf{A}}}$ (assumed to have all the diagonal entries non-zero), $-\underline{\underline{\mathbf{E}}}$ is the strict lower part, and $-\underline{\underline{\mathbf{F}}}$ is the strict upper part.

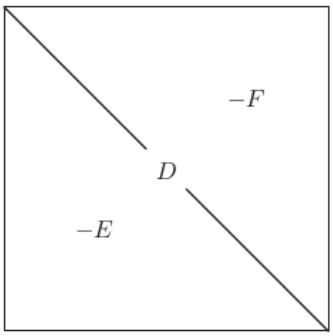


Figure 4.1: Initial partitioning of matrix $\underline{\underline{\mathbf{A}}}$

4.2.1 Jacobi method

One Jacobi iteration is aimed at the annihilation of the i-th component of the residual vector, giving as a result the i-th component of the next approximation.

Thus starting from the problem:

$$(\mathbf{b} - \underline{\underline{\mathbf{A}}}\mathbf{x}_{k+1})_i = 0 \tag{4.11}$$

and applying the proper splitting, the vectorial form of Jacobi iterative method is obtained:

$$\mathbf{x}_{k+1} = \underline{\underline{\mathbf{D}}}^{-1}(\underline{\underline{\mathbf{E}}} + \underline{\underline{\mathbf{F}}})\mathbf{x}_k + \underline{\underline{\mathbf{D}}}^{-1}\mathbf{b} \tag{4.12}$$

4.2.2 Gauss - Seidel method

In a similar way, the aim of a Gauss-Seidel iteration consists in annihilate the i-th component of residual by correcting the i-th component of the residual, but this time by updating the approximate solution immediately after the new component is determined.

Hence, while in Jacobi method it was sufficient to invert the diagonal matrix, simple to invert, now not-strict upper and lower more complex matrices need to be inverted, giving more complexity to the method but better convergence.

In the forward version of the method the order followed to update the solution is $i = 1, 2, ..., n$:

$$\mathbf{x}_{k+1} = (\underline{\underline{\mathbf{D}}} - \underline{\underline{\mathbf{E}}})^{-1}(\underline{\underline{\mathbf{F}}})\mathbf{x}_k + (\underline{\underline{\mathbf{D}}} - \underline{\underline{\mathbf{E}}})^{-1}\mathbf{b} \tag{4.13}$$

while the backward Gauss-Seidel corresponds to making the coordinate corrections in the order $i = n, n-1, ..., 1$:

$$\mathbf{x}_{k+1} = (\underline{\underline{\mathbf{D}}} - \underline{\underline{\mathbf{F}}})^{-1}(\underline{\underline{\mathbf{E}}})\mathbf{x}_k + (\underline{\underline{\mathbf{D}}} - \underline{\underline{\mathbf{F}}})^{-1}\mathbf{b} \tag{4.14}$$

Performing a Symmetric Gauss - Seidel Iteration (SGS) consists in a forward sweep followed by a backward sweep.

It is evident that the Jacobi and the Gauss-Seidel iterations are both of the form:

$$\underline{\underline{\mathbf{M}}}\mathbf{x}_{k+1} = \underline{\underline{\mathbf{N}}}\mathbf{x}_k + \mathbf{b} = (\underline{\underline{\mathbf{M}}} - \underline{\underline{\mathbf{A}}})\mathbf{x}_k + \mathbf{b} \tag{4.15}$$

in which

$$\underline{\underline{\mathbf{A}}} = \underline{\underline{\mathbf{M}}} - \underline{\underline{\mathbf{N}}} \tag{4.16}$$

is a splitting of $\underline{\underline{\mathbf{A}}}$, with $\underline{\underline{\mathbf{M}}} = \underline{\underline{\mathbf{D}}}$ for Jacobi, $\underline{\underline{\mathbf{M}}} = \underline{\underline{\mathbf{D}}} - \underline{\underline{\mathbf{E}}}$ for forward Gauss-Seidel and $\underline{\underline{\mathbf{M}}} = \underline{\underline{\mathbf{D}}} - \underline{\underline{\mathbf{F}}}$ for backward Gauss - Seidel.

4.2.3 Successive Over Relaxation method

The over relaxation method is based on a weight parameter ω, which can be chosen by the operator depending on his needs (speed up the convergence or make more robust the system) which is multiplied by the initial problem to solve:

$$\omega\underline{\underline{\mathbf{A}}}\mathbf{x} = \omega\mathbf{b} \tag{4.17}$$

The adopted decomposition of the matrix is:

$$\omega\underline{\underline{\mathbf{A}}} = (\underline{\underline{\mathbf{D}}} - \omega\underline{\underline{\mathbf{E}}}) - (\omega\underline{\underline{\mathbf{F}}} + (1-\omega)\underline{\underline{\mathbf{D}}}) \tag{4.18}$$

which is used in the recursion

$$(\underline{\underline{\mathbf{D}}} - \omega\underline{\underline{\mathbf{E}}})\mathbf{x}_{k+1} = [\omega\underline{\underline{\mathbf{F}}} + (1-\omega)\underline{\underline{\mathbf{D}}}]\mathbf{x}_k + \omega\mathbf{b} \tag{4.19}$$

As for the SGS, also in this method performing a Symmetric Successive Over Relaxation (SSOR) method consists in a forward sweep followed by a backward sweep, leading to the two steps:

$$(\underline{\underline{\mathbf{D}}} - \omega\underline{\underline{\mathbf{E}}})\mathbf{x}_{k+1/2} = [\omega\underline{\underline{\mathbf{F}}} + (1-\omega)\underline{\underline{\mathbf{D}}}]\mathbf{x}_k + \omega\mathbf{b} \tag{4.20}$$

$$(\underline{\underline{\mathbf{D}}} - \omega\underline{\underline{\mathbf{F}}})\mathbf{x}_{k+1} = [\omega\underline{\underline{\mathbf{E}}} + (1-\omega)\underline{\underline{\mathbf{D}}}]x_{k+1/2} + \omega\mathbf{b} \tag{4.21}$$

4.3 Block Relaxation Schemes

Block relaxation schemes update a whole set of components at each time: instead of only one component they update a whole set of components at each time, typically a sub-vector of the solution vector, being indeed generalizations of the "point" relaxation schemes described above.

It is shown below the partition of the matrix $\underline{\underline{\mathbf{A}}}$, the right-hand side and the solution vector:

$$\underline{\underline{\mathbf{A}}} = \begin{pmatrix} \underline{\underline{\mathbf{A}}}_{11} & \underline{\underline{\mathbf{A}}}_{12} & \underline{\underline{\mathbf{A}}}_{13} & \cdots & \underline{\underline{\mathbf{A}}}_{1p} \\ \underline{\underline{\mathbf{A}}}_{21} & \underline{\underline{\mathbf{A}}}_{22} & \underline{\underline{\mathbf{A}}}_{23} & \cdots & \underline{\underline{\mathbf{A}}}_{2p} \\ \underline{\underline{\mathbf{A}}}_{31} & \underline{\underline{\mathbf{A}}}_{32} & \underline{\underline{\mathbf{A}}}_{33} & \cdots & \underline{\underline{\mathbf{A}}}_{3p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \underline{\underline{\mathbf{A}}}_{p1} & \underline{\underline{\mathbf{A}}}_{p2} & \cdots & \cdots & \underline{\underline{\mathbf{A}}}_{pp} \end{pmatrix}, \qquad x = \begin{pmatrix} \boldsymbol{\xi}_1 \\ \boldsymbol{\xi}_2 \\ \boldsymbol{\xi}_3 \\ \vdots \\ \boldsymbol{\xi}_p \end{pmatrix}, \qquad b = \begin{pmatrix} \boldsymbol{\beta}_1 \\ \boldsymbol{\beta}_2 \\ \boldsymbol{\beta}_3 \\ \vdots \\ \boldsymbol{\beta}_p \end{pmatrix} \tag{4.22}$$

where $\mathbf{b}$ and $\mathbf{x}$ are partitioned into sub-vectors $\boldsymbol{\beta}_i$ and $\boldsymbol{\xi}_i$ compatible with the partitioning of $\underline{\underline{\mathbf{A}}}$. Similar to the scalar case, the block matrix $\underline{\underline{\mathbf{A}}}$ can be split as:

$$\underline{\underline{\mathbf{A}}} = \underline{\underline{\mathbf{D}}} - \underline{\underline{\mathbf{E}}} - \underline{\underline{\mathbf{F}}} \tag{4.23}$$

with:

$$\underline{\underline{\mathbf{D}}} = \begin{pmatrix} \underline{\underline{\mathbf{A}}}_{11} & & & & \\ & \underline{\underline{\mathbf{A}}}_{22} & & & \\ & & \underline{\underline{\mathbf{A}}}_{33} & & \\ & & & \ddots & \\ & & & & \underline{\underline{\mathbf{A}}}_{pp} \end{pmatrix}, \; \underline{\underline{\mathbf{E}}} = -\begin{pmatrix} 0 & & & & \\ \underline{\underline{\mathbf{A}}}_{21} & 0 & & & \\ \underline{\underline{\mathbf{A}}}_{31} & \underline{\underline{\mathbf{A}}}_{32} & 0 & & \\ \vdots & \vdots & \vdots & \ddots & \\ \underline{\underline{\mathbf{A}}}_{p1} & \underline{\underline{\mathbf{A}}}_{p2} & \underline{\underline{\mathbf{A}}}_{p3} & \cdots & 0 \end{pmatrix}, \; \underline{\underline{\mathbf{F}}} = -\begin{pmatrix} 0 & \underline{\underline{\mathbf{A}}}_{12} & \underline{\underline{\mathbf{A}}}_{13} & \cdots & \underline{\underline{\mathbf{A}}}_{1p} \\ & 0 & \underline{\underline{\mathbf{A}}}_{23} & \cdots & \underline{\underline{\mathbf{A}}}_{2p} \\ & & 0 & \cdots & \underline{\underline{\mathbf{A}}}_{3p} \\ & & & \ddots & \vdots \\ & & & & 0 \end{pmatrix} \tag{4.24}$$

The only difference with respect to the cases previously studied is that the meanings of $\underline{\underline{\mathbf{D}}}$, $\underline{\underline{\mathbf{E}}}$ and $\underline{\underline{\mathbf{F}}}$ have changed to their block analogues: thanks to these definitions all the three iterative methods mentioned before can be generalized by substituting vectors to sub-vectors and matrices to sub-block matrices.

Now the three generalized iterative procedures become:

- Block Jacobi

$$\boldsymbol{\xi}_i^{(k+1)} = \underline{\underline{\mathbf{A}}}_{ii}^{-1}((\underline{\underline{\mathbf{E}}} + \underline{\underline{\mathbf{F}}})\mathbf{x}_k)_i + \underline{\underline{\mathbf{A}}}_{ii}^{-1}\boldsymbol{\beta}_i \tag{4.25}$$

- Block Gauss Seidel - forward and backward

$$\boldsymbol{\xi}_i^{k+1} = ((\underline{\underline{\mathbf{D}}} - \underline{\underline{\mathbf{E}}})^{-1}(\underline{\underline{\mathbf{F}}})\mathbf{x}_k)_i + (\underline{\underline{\mathbf{D}}} - \underline{\underline{\mathbf{E}}})^{-1}\boldsymbol{\beta}_i \tag{4.26}$$

$$\boldsymbol{\xi}_i^{k+1} = ((\underline{\underline{\mathbf{D}}} - \underline{\underline{\mathbf{F}}})^{-1}(\underline{\underline{\mathbf{E}}})\mathbf{x}_k)_i + (\underline{\underline{\mathbf{D}}} - \underline{\underline{\mathbf{F}}})^{-1}\boldsymbol{\beta}_i \tag{4.27}$$

- Block SOR

$$(\underline{\underline{\mathbf{D}}} - \omega\underline{\underline{\mathbf{E}}})\boldsymbol{\xi}_i^{k+1/2} = ([\omega\underline{\underline{\mathbf{F}}} + (1-\omega)\underline{\underline{\mathbf{D}}}]\mathbf{x}_k)_i + \omega\boldsymbol{\beta}_i \tag{4.28}$$

$$(\underline{\underline{\mathbf{D}}} - \omega\underline{\underline{\mathbf{F}}})\boldsymbol{\xi}_i^{k+1} = ([\omega\underline{\underline{\mathbf{E}}} + (1-\omega)\underline{\underline{\mathbf{D}}}]x_{k+1/2})_i + \omega\boldsymbol{\beta}_i \tag{4.29}$$

Chapter 5

Overview of *elsA* CFD code

The TSM external solver built in this work is based on the interaction with the elsA software, for the extraction of residuals and Jacobian matrices.

Hence a short and general overview of the CFD code *elsA*, property of Onera, Airbus, Safran, is presented.

elsA stands for *ensemble logiciel de simulation en Aérodynamique*, which means *software platform for Aerodynamic simulation.*

5.1 Discretization models

The main space discretization models used in fluid mechanic numerical implementations are:

- Finite differences: based on the differential form of the motion equations, these schemes are based on Taylor series development and the conservative variables are known on the points of the mesh. These methods are not conservative, so particular treatments are needed to transport the conservative variables.
- Finite volumes: based on the integral form of the motion equations, they average the variables over a control volume. Surface integrals are obtained on the faces of each mesh cell, and the fluxes through these surfaces allows the information to be transmitted from a cell to another one. This method is intrinsically conservative.
- Finite elements: based on the integral form of the motion equations, they consist in finding an approximate solution of the exact one with a field defined over sub-domains chosen among an arbitrary family of fields (generally polynomials).

elsA uses a space discretization of the type *finite volumes*, [13].

5.1.1 Finite volumes

Generalizing the problem to a 3D problem, in the finite volumes approach the values of conservative variables are computed at each cell center of volume V. The integral form of conservative laws on a cell is:

$$\frac{\partial}{\partial t}\int_V \mathbf{W}\,\mathrm{d}V + \int_V \nabla\cdot\mathbf{F}(\mathbf{W})\,\mathrm{d}V = 0 \tag{5.1}$$

Applying the Green - Ostrograski theorem it is possible to rewrite the equation, being $\mathcal{S}$ the interface surface of each cell:

$$\frac{\partial}{\partial t}\int_V \mathbf{W}\,\mathrm{d}V + \int_{\mathcal{S}} \mathbf{F}(\mathbf{W})\cdot\mathbf{n}\,\mathrm{d}\mathcal{S} = 0 \tag{5.2}$$

Defining $\mathbf{W}$ as the average of the conservative variables on a cell $\mathbf{W} = \frac{1}{V}\int_V \mathbf{W}\,\mathrm{d}V$ and introducing the residual $\mathbf{R}(\mathbf{W}) = \int_{\mathcal{S}} \mathbf{F}(\mathbf{W})\cdot\mathbf{n}\,\mathrm{d}\mathcal{S}$ on a cell, the equation becomes:

$$\frac{\partial V\mathbf{W}}{\partial t} + \mathbf{R}(\mathbf{W}) = 0 \tag{5.3}$$

Each cell has several interfaces, in the case of a 3D cell, there are 6 interfaces, that can appear in the equation:

$$\frac{\partial V\mathbf{W}}{\partial t} + \sum_{k=1}^{6} \mathbf{F}_{face_k}(\mathbf{W},\mathbf{W}_i) = 0 \tag{5.4}$$

where $\mathbf{W}$ is the vector of conservative variables in the cell of interest and $\mathbf{W}_i$ is the vector of conservative variables in the neighbouring cells, chosen depending on the used stencil to discretize the space.

5.1.2 Space discretization - ROE scheme

In order to explicit the fluxes on interfaces, an upwind Roe scheme is chosen in the elsA solver.

Considering the initial-value problem for a hyperbolic system of conservation laws, we seek $\mathbf{W}(x,t)$ (1D formulation):

$$\frac{\partial \mathbf{W}}{\partial t} + \frac{\partial \mathbf{F}}{\partial x} = 0 \tag{5.5}$$

Roe describes a mechanism by which any algorithm developed for numerical solution of the linear advection equation

$$\frac{\partial \mathbf{W}}{\partial t} + a\frac{\partial \mathbf{W}}{\partial x} = 0 \tag{5.6}$$

can be generalized to the case of non-linear systems. He considers approximate solutions which are exact solutions to an approximate problem

$$\frac{\partial \mathbf{W}}{\partial t} + \underline{\underline{\widetilde{\mathbf{A}}}}\frac{\partial \mathbf{W}}{\partial x} = 0 \tag{5.7}$$

where $\underline{\underline{\widetilde{\mathbf{A}}}}$ is a constant matrix that has to be chosen so that it is representative of local conditions.

Defining two states W_L and W_R of the space of the states, the numerical flux F_{Roe} is traditionally defined by:

$$\mathbf{F}_{Roe}(\mathbf{W}_L,\mathbf{W}_R) = \frac{\mathbf{F}_c(\mathbf{W}_L)+\mathbf{F}_c(\mathbf{W}_R)}{2} - \frac{1}{2}|\underline{\underline{\widetilde{\mathbf{A}}}}|(\mathbf{W}_R-\mathbf{W}_L) \tag{5.8}$$

The matrix $\underline{\underline{\widetilde{\mathbf{A}}}} = \underline{\underline{\widetilde{\mathbf{A}}}}(\mathbf{W}_L, \mathbf{W}_R)$ satisfies the following list of properties:

$$\begin{cases} \text{It constitutes a linear mapping from the vector space } \mathbf{W} \text{ to the vector space } \mathbf{F}_c \\ \text{As } \mathbf{W}_L \to \mathbf{W}_R \to \mathbf{W},\ \underline{\underline{\widetilde{\mathbf{A}}}}(\mathbf{W}_L, \mathbf{W}_R) \to \underline{\underline{\mathbf{A}}}(\mathbf{W}), \text{ where } \underline{\underline{\mathbf{A}}}(\mathbf{W}) = \dfrac{\partial \mathbf{F}}{\partial \mathbf{W}} \\ \text{For any } \mathbf{W}_L, \mathbf{W}_R,\ \underline{\underline{\widetilde{\mathbf{A}}}}(\mathbf{W}_L, \mathbf{W}_R)(\mathbf{W}_L - \mathbf{W}_R) = \mathbf{F}_c(\mathbf{W}_L) - \mathbf{F}_c(\mathbf{W}_R) \\ \text{The eigenvectors of } \underline{\underline{\widetilde{\mathbf{A}}}} \text{ are linearly independent} \end{cases} \tag{5.9}$$

To discuss the construction of the $\underline{\underline{\widetilde{\mathbf{A}}}}$ matrix in detail, the reader is referred to [14] and [15].

In our false 3D problem (that is actually 2D), the conservative variables $\mathbf{W}$ and the convective fluxes $\mathbf{F}_c$ are as follows:

$$\mathbf{W} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{pmatrix}, \qquad \mathbf{F}_{cx} = \begin{pmatrix} \rho u \\ \rho uu + p \\ \rho uv \\ \rho uw \\ (\rho E + p)u \end{pmatrix}, \qquad \mathbf{F}_{cy} = \begin{pmatrix} \rho v \\ \rho vu \\ \rho vv + p \\ \rho vw \\ (\rho E + p)v \end{pmatrix}, \qquad \mathbf{F}_{cz} = \begin{pmatrix} \rho w \\ \rho wu \\ \rho wv \\ \rho ww + p \\ (\rho E + p)w \end{pmatrix} \tag{5.10}$$

Expliciting the problem, the $\underline{\underline{\mathbf{A}}}(\mathbf{X}_{av})$ matrix is the Jacobian of convective fluxes with conservative variables which are an average of left and right cell.

$\mathbf{X}_{av}$ indicates the traditional Roe average, for $\mathbf{X} = \left(u, v, w, \frac{E+p}{\rho}\right)$

$$\mathbf{X}_{av} = \frac{\sqrt{\rho_L}\mathbf{X}_L + \sqrt{\rho_R}\mathbf{X}_R}{\sqrt{\rho_L} + \sqrt{\rho_R}} \tag{5.11}$$

and ρ is calculated simply as:

$$\rho = \sqrt{\rho_L \rho_R} \tag{5.12}$$

5.2 Solution process - steady mean flow solution

elsA has been built as a powerful CFD code, able to be accurate or robust depending on the user selection of the numerical parameters. Choosing the robustness, so being able to perform every simulation without encountering too many convergence problems, a lack in accuracy can be faced, leading to slower convergence.

Here vectors and matrices are indicated as they were 1D, to lighten the notation.

As usual the steady problem to solve is:

$$R(W) = 0$$

The approach elsA uses for solving this problem is an inexact Newton method, with a first order implicit stage:

$$\left(\frac{V}{\Delta\tau} + \frac{\partial R}{\partial W}^{APP}\right)\Delta W_q = -R(W_q) \tag{5.13}$$

The Jacobian matrix is an approximation of the exact Jacobian matrix, which ensures diagonal dominance and hence higher robustness, but at the same time slows the convergence because the matrix is not the exact one.

The solution method adopted is incomplete LU-SSOR (only 2 or 4 iterations are performed).

On the other hand, the external solver implements an exact Newton method, with a first order implicit stage. This time the Jacobian matrix to invert is the exact one, that can be first or second order in space discretization.

$$\left(\frac{V}{\Delta\tau} + \frac{\partial R}{\partial W}^{EXA}\right)\Delta W = -R(W^q) \tag{5.14}$$

The LU-SSOR solution method fails with the stiffer $\frac{\partial R}{\partial W}^{EXA}$ because it consists in a too ill conditioned matrix or it has bad properties. The failure of the pseudo Newton/Backward-Euler method is maybe because the solution of the linear system is not converged enough, or too converged. This is why only two approaches are feasible:

- direct solver LU
- iterative Krylov solver like GMRES with an ILU preconditioner

It is fundamental to remark that $\frac{\partial R}{\partial W}^{EXA}$ never appears in a standard elsA steady mean flow solution. In order to get access to the exact Jacobian matrix the *optimization module* by elsA is then used.

5.3 Solution process - optimization module

The objective in the optimization module is to solve the linearized system of discretized equations around the steady configuration. The steady problem solution satisfies:

$$R(W, X) = 0 \tag{5.15}$$

where X is the fluid mesh.

Linearizing this residual with respect to a design parameter p (direct differentiation method) gives the sensitivity equation, for each of the parameters:

$$\frac{\partial R}{\partial W}\frac{\mathrm{d}W}{\mathrm{d}p} = -\frac{\partial R}{\partial X}\frac{\mathrm{d}X}{\mathrm{d}p} \tag{5.16}$$

In elsA it is solved by implementing:

$$\left(\frac{V}{\Delta\tau} + \frac{\partial R}{\partial W}^{APP}\right)\frac{\mathrm{d}\Delta W}{\mathrm{d}p} = -\frac{\partial R}{\partial W}^{EXA}\frac{\mathrm{d}W}{\mathrm{d}p}^{q} - \frac{\partial R}{\partial X}\frac{\mathrm{d}X}{\mathrm{d}p} \tag{5.17}$$

which is the linearization of the steady mean flow solution.

While the equation for the adjoint method (read section 10.1 in the chapter about optimization) is, one for each constraint or objective function:

$$\left(\frac{\partial R}{\partial W}\right)^T \lambda = -\left(\frac{\partial J}{\partial W_b}\frac{\mathrm{d}W_b}{\mathrm{d}W} + \frac{\partial J}{\partial W}\right)^T \tag{5.18}$$

which in elsA is implemented as:

$$\left(\frac{V}{\Delta\tau} + \left(\frac{\partial R}{\partial W}^{APP}\right)^T\right)\Delta\lambda = -\left(\frac{\partial R}{\partial W}^{EXA}\right)^T \lambda^q - \left(\frac{\partial J}{\partial W_b}\frac{\mathrm{d}W_b}{\mathrm{d}W} + \frac{\partial J}{\partial W}\right)^T \tag{5.19}$$

being J the objective or the constraint function (read chapter 10).

The Jacobian matrix ($\frac{\partial R}{\partial W}$) 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 kind of iterative strategy 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 exact Jacobian $\frac{\partial R}{\partial W}^{EXA}$ appears in that right hand side of the equation and the extraction of it is now available.

Chapter 6

Numerical implementation

In this chapter all the aspects concerning the numerical implementation of all the features are clarified, and all the steps done in the building of the code are presented.

6.1 Numerical solution of the TSM system - Backward Euler

Here the bold notation will be used for the sake of clarity to indicate the concatenation of instants (as in chapter 2), so in steady equations the notations for vectors and matrices is as they were scalars (the actual dimensions are given by the mesh size, multiplied by 5 that is the number of conservative variables).

Considering a non-deformable mesh, the dependence of the residual is only on the conservative variables and not on the mesh, that is fixed, the non-linear system to solve for an unsteady case is:

$$V\frac{\partial W}{\partial t} + R(W) = 0 \tag{6.1}$$

that in a steady case is reduced to:

$$R(W) = 0 \tag{6.2}$$

Adding a pseudo time derivative to help the convergence process, the differential equation become a pseudo time marching equation of a physical steady problem.:

$$V\frac{\partial W}{\partial \tau} + R(W) = 0 \tag{6.3}$$

The Backward Euler scheme consists in solving the previous system, introducing a dependency on the current iteration of the residuals:

$$VW^{q+1} = VW^{q} + \Delta\tau R(W^{q+1}) \tag{6.4}$$

The residuals expression at the iteration $q+1$ is obtained via Taylor's series:

$$R(W^{q+1}) = R(W^q) + \left.\frac{\partial R}{\partial W}\right|^q (W^{q+1} - W^q) \tag{6.5}$$

Hence the steady system to solve (without the source term that keeps into account the time derivative) is the following:

$$V\frac{\partial W}{\partial \tau} = -R(W^{q+1}) \tag{6.6}$$

$$VW^{q+1} = VW^q - \Delta\tau R(W^q) - \Delta\tau \left.\frac{\partial R}{\partial W}\right|^q (W^{q+1} - W^q) \tag{6.7}$$

$$\frac{V}{\Delta\tau}(W^{q+1} - W^q) + \left.\frac{\partial R}{\partial W}\right|^q (W^{q+1} - W^q) = -R(W^q) \tag{6.8}$$

$$\left(\frac{V}{\Delta\tau} + \left.\frac{\partial R}{\partial W}\right|^q\right)\Delta W = -R(W^q) \tag{6.9}$$

where q is the index related to the number of Newton iterations.

The solution is the vector $\Delta W = W^{q+1} - W^q$. At each iteration the solution W^q is updated with W^{q+1}.

The final steady system is then:

$$\left(\frac{V}{\Delta\tau} + \left.\frac{\partial R}{\partial W}\right|^q\right)\Delta W = -R(W^q) \tag{6.10}$$

where $\Delta\tau_{cell} = \dfrac{CFL \cdot \sqrt[3]{V_{cell}}}{||U_{cell}|| + a_{cell}}$ in the steady case.

For the case of a TSM computation, the equation to solve is:

$$\mathbf{R}(\mathbf{W}) + \mathbf{V}\underline{\underline{\mathbf{D_t}}}\mathbf{W} = 0 \tag{6.11}$$

Applying again Backward Euler in the steady problem and adding the new source term the system becomes:

$$\left(\frac{V}{\Delta\tau}\underline{\underline{\mathbf{I}}} + \left.\frac{\partial \mathbf{R}}{\partial \underline{\underline{\mathbf{W}}}}\right|^q\right)\Delta\mathbf{W} = -\mathbf{R}(\mathbf{W}^q) - \mathbf{V}\underline{\underline{\mathbf{D_t}}}\mathbf{W} \tag{6.12}$$

In the TSM case the vector $\mathbf{W}$ gathers the conservative variables of all the $2N+1$ instants, and the left hand side matrix to invert in order to solve the system is block diagonal.

As already written in equation 2.22, instead of solving the entire system in equation 6.12, it is possible to solve it separately for each instant. By doing that, the matrix to invert will remain the same dimensions of the Jacobian matrix of the single steady problem, instead of $2N+1$ times the dimensions of the initial matrix.

For each iteration q eq. 6.13 will be solved $2N+1$ times.

$$\left(\frac{V}{\Delta\tau_n} + \left.\frac{\partial R(W_n)}{\partial W}\right|^q\right)\Delta W_n = -R(W_n^q) - VS_n, \quad 1 \leq n \leq 2N+1 \tag{6.13}$$

At the following q, the source term is updated and the process continues until a certain convergence is reached.

In equation 6.12 a numerical choice can be made concerning the source term:

- dependence on the previous iteration q, $\mathbf{S} = \mathbf{V}\underline{\underline{\mathbf{D_t}}}\mathbf{W}^q$
- dependence on the previous iteration q, $\mathbf{S} = \mathbf{V}\underline{\underline{\mathbf{D_t}}}\mathbf{W}^{q+1}$

The first version of the code is implemented with the dependence on the previous iteration.

6.2 Explicitation of the source term - partially implicit TSM

The unsteady problem, expressed in the explicit form of the source term, i.e. in the case it depends on the conservative variables calculated at the previous iteration, consists in:

$$\left(\frac{\mathbf{V}}{\mathbf{\Delta}\tau}\underline{\underline{\mathbf{I}}} + \left.\frac{\partial \mathbf{R}}{\underline{\underline{\partial \mathbf{W}}}}\right|^q\right)\Delta\mathbf{W} = -\mathbf{R}(\mathbf{W}^q) - \mathbf{V}\underline{\underline{\mathbf{D_t}}}\mathbf{W}^q \tag{6.14}$$

This is a *partially implicit TSM* implementation, because despite the implicitation of the numerical method, the source term has not been implicited.

Since the beginning of the TSM chapter, the quantities conservative variables $\mathbf{W}^q$ and $\mathbf{R}^q$ have been considered for simplicity as vectors of scalars, and the Jacobian matrix $\left.\frac{\partial \mathbf{R}}{\underline{\underline{\partial \mathbf{W}}}}\right|^q$ a simple "mono-block" matrix.

But in fact $\mathbf{W}^q$ and $\mathbf{R}^q$ are vectors of vectors corresponding to each instant (one element per mesh cell and conservative variable), and the Jacobian matrix is a block diagonal matrix with the Jacobian matrix corresponding to each instant concatenated on the diagonal.

Naming the dimensions of the mesh $I_m = 256$, $J_m = 32$ and $K_m = 1$, and reminding that the number of conservative variables is 5, the sizes of the sub-vectors for the n^{th} instant are:

$$\text{size}(\mathbf{W}_n^q) = (I_m \cdot J_m \cdot K_m \cdot 5) \times 1 \tag{6.15}$$

$$\text{size}(\mathbf{R}_n^q) = (I_m \cdot J_m \cdot K_m \cdot 5) \times 1 \tag{6.16}$$

$$\text{size}\left(\left.\frac{\partial \mathbf{R}}{\underline{\underline{\partial \mathbf{W}}}}\right|_n^q\right) = (I_m \cdot J_m \cdot K_m \cdot 5) \times (I_m \cdot J_m \cdot K_m \cdot 5) \tag{6.17}$$

Also the time spectral matrix has a different shape, because it must have the same dimensions as the Jacobian matrix.

Analyzing the problem from a strictly numerical point of view, we call $\underline{\underline{\mathbf{d}_t}}$ the coefficient matrix that is a $(2N+1) \times (2N+1)$ matrix of the type:

$$\underline{\underline{\mathbf{d}_t}} = \begin{pmatrix} 0 & \dots & d_{t\,1,2N+1} \\ \vdots & \ddots & \vdots \\ d_{t\,2N+1,1} & \dots & 0 \end{pmatrix} \tag{6.18}$$

and the block - time spectral matrix is:

$$\underline{\underline{\mathbf{D_t}}} = \begin{pmatrix} 0 & \dots & \underline{\underline{\mathbf{D_t}}}_{1,2N+1} \\ \vdots & \ddots & \vdots \\ \underline{\underline{\mathbf{D_t}}}_{2N+1,1} & \dots & 0 \end{pmatrix} \tag{6.19}$$

and the $\underline{\underline{\mathbf{D_t}}}_{i,j}$ matrix is an identity matrix multiplied by the coefficient $d_{t\,i,j}$

$$\underline{\underline{\mathbf{D_t}}}_{i,j} = \begin{pmatrix} d_{t\,i,j} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & d_{t\,i,j} \end{pmatrix} \tag{6.20}$$

The block system to solve is shown below:

$$\begin{pmatrix} \frac{\mathbf{V}}{\mathbf{\Delta\tau_1}}\underline{\underline{\mathbf{I}}} + \frac{\partial \mathbf{R}}{\partial \mathbf{W}}\Big|_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \frac{\mathbf{V}}{\mathbf{\Delta\tau_{2N+1}}}\underline{\underline{\mathbf{I}}} + \frac{\partial \mathbf{R}}{\partial \mathbf{W}}\Big|_{2N+1} \end{pmatrix}^q \begin{pmatrix} \Delta\mathbf{W}_1 \\ \vdots \\ \Delta\mathbf{W}_{2N+1} \end{pmatrix} = \\ - \begin{pmatrix} \mathbf{R}_1 \\ \vdots \\ \mathbf{R}_{2N+1} \end{pmatrix}^q - \begin{pmatrix} 0 & \dots & \mathbf{V}\underline{\underline{\mathbf{D_t}}}_{1,2N+1} \\ \vdots & \ddots & \vdots \\ \mathbf{V}\underline{\underline{\mathbf{D_t}}}_{2N+1,1} & \dots & 0 \end{pmatrix}^q \begin{pmatrix} \mathbf{W}_1 \\ \vdots \\ \mathbf{W}_{2N+1} \end{pmatrix}^q \tag{6.21}$$

The nature of this problem allows to solve separately the $2N+1$ rows of the system above: in a Newton q-iteration an independent set of $2N+1$ equations is solved independently, and at the end of the iteration, the source term coupling the instants is updated, exchanging data among the instants.

Besides a dependence on the pitching frequency (ω) and the number of harmonics (N) that the case intends to study is added to the pseudo time step, turning it, after a stability analysis in the frequency domain, into:

$$\Delta\tau_{cell} = \frac{CFL \cdot \sqrt[3]{V_{cell}}}{||U_{cell}|| + a_{cell} + N\omega\sqrt[3]{V_{cell}}} \tag{6.22}$$

In this case the time step is restricted for stability reasons. This formulation of the CFL implies restrictions on the CFL number, since high frequency and/or a high number of sampled instants. In fact it has been demonstrated in the following that the convergence of the method slows down for increasing N and f.

This is the case of explicit methods, whose stability criteria on CFL number are very restrictive.

To improve stability and relax the constraints on the CFL number fully implicit schemes will be introduced.

6.3 Algorithm of the external Python TSM solver

Once examined all the peculiar aspects of the Time Spectral Method:

- TSM backward Euler numerical scheme

- Absolute velocity in the relative frame of reference
- Lower - Upper decomposition for the inversion of the Jacobian matrix

the algorithm expressing how the external python TSM solver is supposed to work is finally presented.

In order to solve the system, an interaction between the external python solver, in which the conservative variables $(\rho, \rho u, \rho v, \rho w, \rho E)$ are computed, and the elsA code, from which the residuals and the Jacobian matrix are extracted for each iteration, is needed.

The steps the solver performs, combined with elsA, are the following:

- The prescribed conditions are set:
  - atmospheric infinite state conditions p_∞^0, p_∞, q_∞, T_∞^0
  - Mach number M
  - amplitude of the motion $\widehat{\alpha}$
  - pitching frequency f
  - number of instants $2N+1$
  - threshold of convergence tol
- The initial uniform field is computed by elsA, with the prescribed conditions, for a zero angle of attack.
- The python script reads the files written by elsA, and stores the initial conservative variables, the volumes of each cell (constant for the whole computation), and the mesh coordinates and distances from the center of the mesh, to allow after the computation of the entrainment speed.
- For each q-iteration a directory *elsA_iter_q* is created, in which $2N+1$ *inst_n* subdirectories are created, to keep trace of all the steps. In each of these subdirectories every needed extraction will be available (upper and lower pressure, fluxes, conservative variables with coordinates for visualization, residuals, ...), it is sufficient to ask elsA for that.
- elsA is launched for 1 iteration with a very low CFL in order not to modify the flow field, but just to extract Jacobian and residuals for that particular condition. The launched simulations will be a *steady computation* taking into account the entrainment speed (used as a "trick" to simulate a forced motion) in *absolute velocity - relative frame of reference* formulation. Depending on the instant to compute, the elsA simulations will have not only different angles of attack picked in the period (fig. 6.1), but also different entrainment speeds. The angular speed doesn't correspond to the only pitching frequency, but is indeed the derivative of the angle of attack.

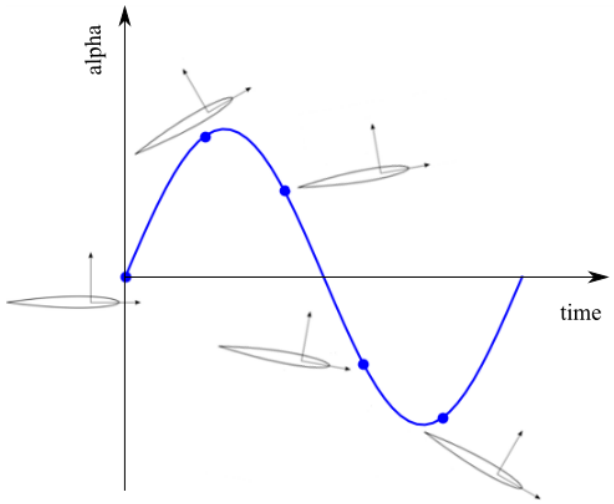


Figure 6.1: Variation of the angle of attack during the period

Being the rotation applied to the y-axis, the normal axis to the plane of the airfoil, the vector of the angular velocity $\mathbf{\Omega}$ is constituted by only one component in the y-direction, $\mathbf{\Omega} = (\Omega_x, \Omega_y, \Omega_z) = (0, \Omega_y, 0)$

$$\alpha(t) = \alpha_0 + \widehat{\alpha}\sin(\omega t) \tag{6.23}$$

$$\Omega_y(t) = \dot{\alpha}(t) = \omega\widehat{\alpha}\cos(\omega t) \tag{6.24}$$

Writing the pitching angular speed as $\omega = 2\pi f$ and the dependence of the n-th instant on the period of oscillation T as $t = \dfrac{n}{2N+1}T = \dfrac{n}{2N+1}\dfrac{1}{f}$ (knowing that $T = \dfrac{1}{f}$), it is possible to write the eqs. 6.24 in the following more operative way:

$$\alpha(n) = \alpha_0 + \widehat{\alpha}\sin\left(2\pi\frac{n}{2N+1}\right) \tag{6.25}$$

$$\Omega_y(n) = \dot{\alpha}(n) = \omega\widehat{\alpha}\cos\left(2\pi\frac{n}{2N+1}\right) \tag{6.26}$$

At the end of the iteration all the extractions will be available, but above all the residuals file and the Jacobian matrix file.

- These files are read and stored by the python script (R^q and $\frac{\partial R}{\partial W}\big|^q$), for each instant.
- The source term is computed from the vector of conservative variables at the previous iteration q (for the first iteration the initial field is used) by multiplying the time-spectral operator matrix of coefficients with the conservative variables, for each instant n, where $d_{i,j} = \dfrac{2\pi}{T}\dfrac{1}{2}(-1)^{i-j}\csc\left[\dfrac{\pi}{2N+1}(i-j)\right]$:

$$S_n^q = \begin{pmatrix} S_1 \\ S_2 \\ \vdots \\ S_{2N+1} \end{pmatrix}_n^q = V \begin{pmatrix} 0 & d_{t\,1,2} & \dots & d_{t\,1,2N+1} \\ d_{t\,2,1} & 0 & \dots & d_{t\,2,2N+1} \\ \vdots & \vdots & \ddots & \vdots \\ d_{t\,2N+1,1} & d_{t\,2N+1,2} & \dots & 0 \end{pmatrix} \begin{pmatrix} W_1 \\ W_2 \\ \vdots \\ W_{2N+1} \end{pmatrix}_n^q \tag{6.27}$$

- The norm of the TSM right hand side, i.e. what is supposed to converge to zero, is computed from the residuals extracted by elsA R, the source term S, and the number of cells of the mesh n_{cells} as:

$$||R_{TSM}||_n = \sqrt{\frac{\sum_{i=1}^{n_{cells}} (R_i + S_i)_n^2}{n_{cells}}} \tag{6.28}$$

  It is calculated for each of the instants, and the simulation stops only once all the $2N+1$ norms of residuals have dropped below the asked tolerance.

- Entrainment local speeds are computed:

$$u_e = \mathbf{\Omega} \times \mathbf{r}|_x = \Omega_y z \tag{6.29}$$
$$v_e = 0 \tag{6.30}$$
$$w_e = \mathbf{\Omega} \times \mathbf{r}|_z = -\Omega_y x \tag{6.31}$$

  taking into account the nature of the studied problem, i.e. an airfoil in pitching motion around the aerodynamic center in the y- direction. The components of angular velocity in directions x and z are null: $\Omega = (\Omega_x, \Omega_y, \Omega_z) = (0, \Omega_y, 0)$

- Local speeds are computed starting from the conservative variables, together with the local speed of sound as:

$$u = \frac{\rho u_{abs}}{\rho} - u_e \tag{6.32}$$
$$v = \frac{\rho v_{abs}}{\rho} - v_e \tag{6.33}$$
$$w = \frac{\rho w_{abs}}{\rho} - w_e \tag{6.34}$$
$$a = \sqrt{\frac{\gamma \cdot (\gamma - 1)}{\rho} \left(\rho E_{abs} - \rho \frac{u_{abs}^2 + v_{abs}^2 + w_{abs}^2}{2} \right)} \tag{6.35}$$

  (remembering that $E_{abs} = e + \frac{1}{2}||U_{abs}^2||$ is the total energy dependent on the frame of reference and e the internal one, independent of the frame of reference.)

- The local pseudo-time steps are computed from global CFL number, the dimension of the cells, the speeds, the number of harmonics N and the pitching angular speed ω.

$$\Delta\tau = \frac{CFL \cdot \sqrt[3]{V}}{||U|| + a + N\omega\sqrt[3]{V}} \tag{6.36}$$

- The term $\frac{V}{\Delta\tau}$ is added to the diagonal of the Jacobian matrix to help the convergence.

- The linear system is solved for each instant, to obtain as a result the increment of the conservative variables ΔW_n:

$$\left(\frac{V}{\Delta\tau} + \left. \frac{\partial R}{\partial W} \right|_n^q \right) \cdot \Delta W_n = -R_n^q - S_n^q \tag{6.37}$$

- The increment vector is added to the previous vector of the conservative variables:

$$W_n^{q+1} = W_n^q + \Delta W_n \tag{6.38}$$

- A file is written to gather all the variables at the new iteration step $q+1$.
- The file is copied to the directory of the new iteration $q+1$ and the process is repeated with a restart from the new computed field (except of course for the calculations of the initial field and of the constant quantities) until the desired degree of convergence is achieved.

The algorithm can be summed up in the main steps in the following map:

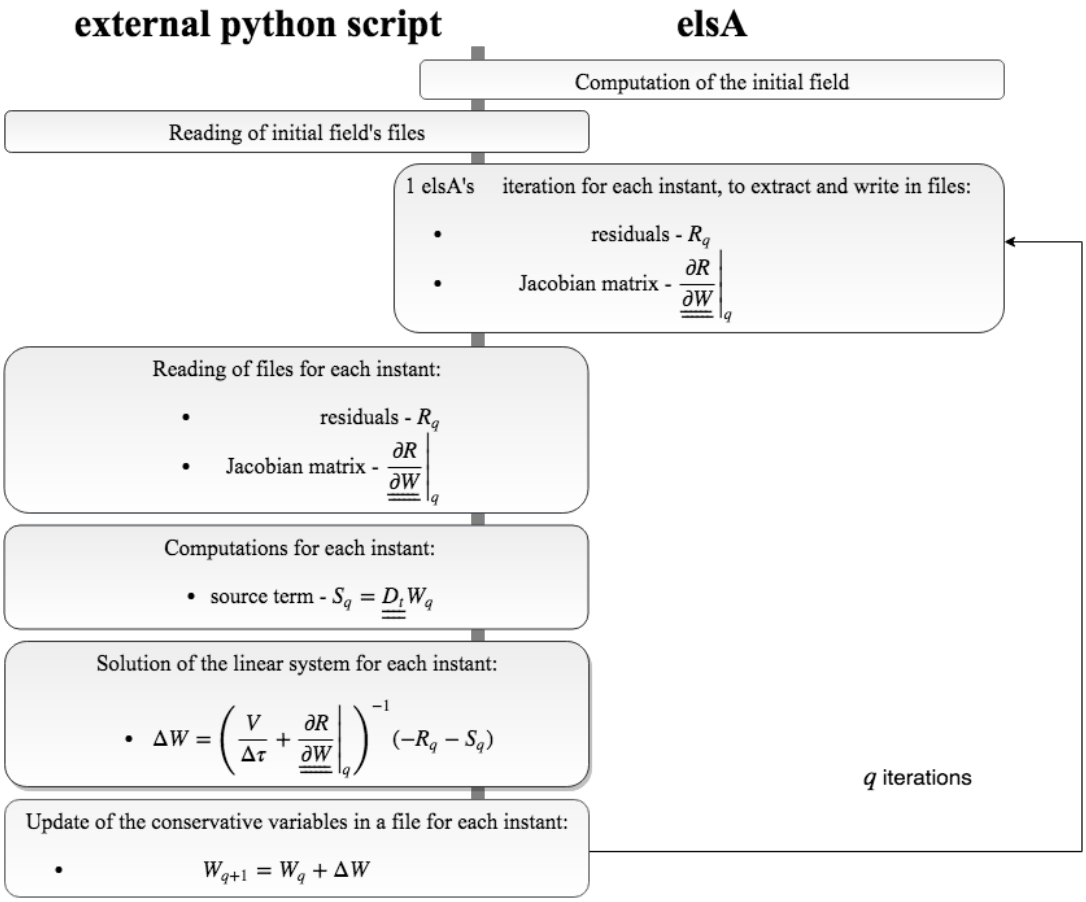


Figure 6.2: Algorithm of the external TSM solver, partially implicit method

6.4 Parallel computing

The serial version of the code, which consists in solving the linear system sequentially for $2N+1$ times, i.e. for each instant, is evidently slow in terms of CPU time. And the higher is the chosen number of sampled instants, the higher is the CPU time of more or less a factor $2N+1$.

Nevertheless, the intrinsic nature of TSM, which presents $2N+1$ uncoupled systems to solve and a final update at the end of each iteration, allows to split the problem in $2N+1$ problems, also in terms of CPU computational effort.

This can be achieved thanks to the *multiprocessing* technique, and in particular to the *multiprocessing module* present in python [16].

The multiprocessing module allows the programmer to fully leverage multiple processors on a given machine to take full advantage of a multi-core system. In the case of TSM the choice to

split the problem in $2N + 1$ cores is straightforward, in such a way that each processor solves only one instant simultaneously.

Multiprocess works simply by creating a *process* object and then calling its *start()* method, passing arguments to the functions to run in parallel. After using *put()* and *get()* to exchange variables between the function and the main source code, the process is finally complete with the *join()* method.

In figure 6.3 it is shown an example of the logics of multiprocessing for a simple function aimed at building a list of $n = 4$ strings in a 4-core machine, showing the serial case and the parallel one.

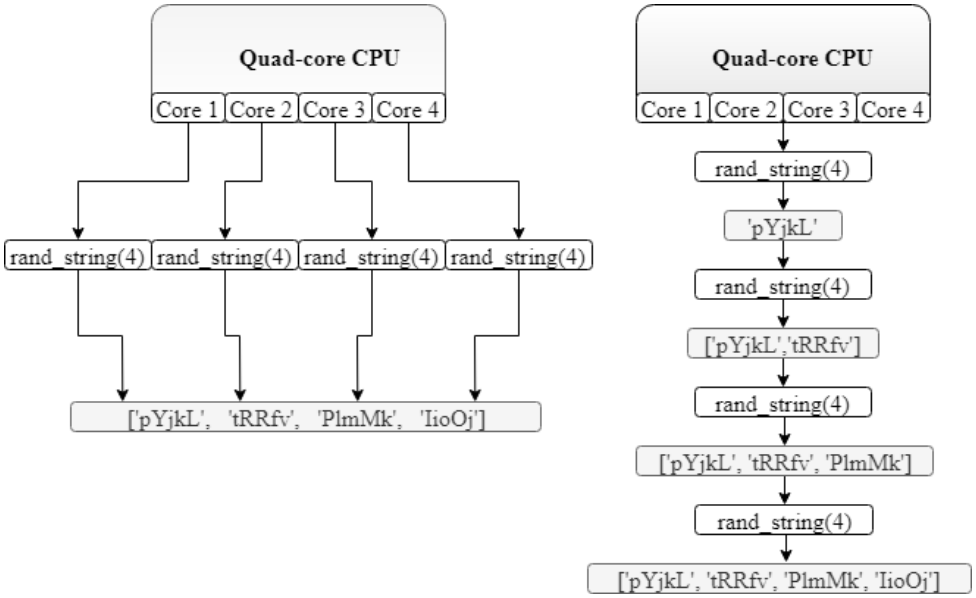


Figure 6.3: Exemple of multiprocessing computation

Analyzing the serial program from the CPU time point of view, two bottlenecks have been isolated: the elsA iteration and the solution of the linear system that in a first release of the code is achieved by a direct solver with LU decomposition, a very time spending procedure.

Hence it has been decided to compute in parallel the two bottleneck - blocks in order to dispatch each instant computation to a single core.

- First bottleneck: the $2N + 1$ 1 iteration - computations by elsA are computed on $2N + 1$ cores at the same time, giving in input to elsA the right instant, i.e. the right angle of attack and rotation speed.
- Second bottleneck: the solution of $2N + 1$ linear systems is computed simultaneously on $2N + 1$ cores, using the residuals, the source term, and the Jacobian matrix related to that instant.

After the second parallel computing step, the increment of the conservative variables is given to the main script, since the TSM system is coupled by the source term and the instants are not independent.

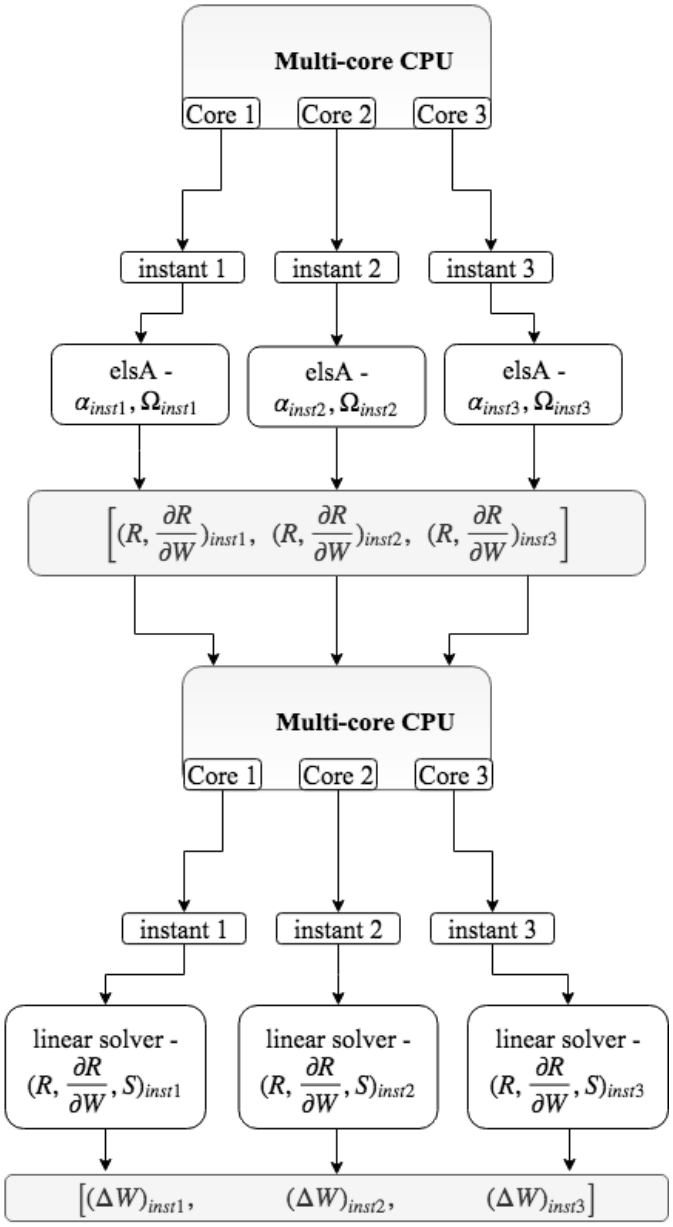


Figure 6.4: Multiprocessing steps for the external python TSM solver for 3 instants

Unfortunately the CPU time cannot decrease by a factor $2N+1$ due to internal multiprocessing of *numpy* and *scipy* python modules, that lead to interaction among the machine processors and slow the process. Also other system processes, such as the operating system, are running in the background. Thus, using for example 8 cores on a 8-cores machine, the last core doesn't have enough capacity left to further increase the performance of the eighth process to a large extend.

By the way, a consistent amount of time can be saved by implementing the code in a multiprocessing window, estimated of 100% for a number of sampled instants of 3 (the expected 150% is not possible to achieve due to the reasons expressed above); the performance increase is less significant when moving to 5 and 7 instants (and 5 and 7 processors). An increase of number of instants cannot be performed in a local 8-core machine, but needs a cluster computation.

6.5 Implicitation of the source term - fully implicit TSM

As previously mentioned, despite the straightforwardness of the partially implicit TSM formulation, they present strong constraints in the rate of convergence, and the development of an implicitation of the source term seemed necessary to relax convergence issues. Therefore in order to improve the performances, it is useful to express the dependence of the source term on the current iteration $q+1$, as Sicot in [17], and the equation 6.14 is written differently as:

$$\left(\frac{\mathbf{V}}{\mathbf{\Delta}\tau}\underline{\underline{\mathbf{I}}}+\left.\frac{\partial \mathbf{R}}{\partial \mathbf{W}}\right|^{q}\right)\Delta\mathbf{W}=-\mathbf{R}(\mathbf{W}^{q})-\mathbf{V}\underline{\underline{\mathbf{D_t}}}\mathbf{W}^{q+1} \tag{6.39}$$

The operator $\underline{\underline{\mathbf{D_t}}}$ is linear, and can be written in the following way:

$$\underline{\underline{\mathbf{D_t}}}(\mathbf{W}^{q+1})=\underline{\underline{\mathbf{D_t}}}(\mathbf{W}^{q})+\underline{\underline{\mathbf{D_t}}}(\Delta\mathbf{W}) \tag{6.40}$$

leading to a coupling of the increments $\Delta\mathbf{W}$ at all instants.

The new equation to solve is:

$$\left(\frac{\mathbf{V}}{\mathbf{\Delta}\tau}\underline{\underline{\mathbf{I}}}+\left.\frac{\partial \mathbf{R}}{\partial \mathbf{W}}\right|^{q}\right)\Delta\mathbf{W}+\mathbf{V}\underline{\underline{\mathbf{D_t}}}\mathbf{\Delta W}=-\mathbf{R_{TSM}}(\mathbf{W}^{q}) \tag{6.41}$$

where $\mathbf{R_{TSM}}(\mathbf{W}^{q})=\mathbf{R}(\mathbf{W}^{q})+\underline{\underline{\mathbf{D_t}}}(\mathbf{W}^{q})$ is the same TSM residual as in the explicit formulation.

The left hand side matrix to invert is now non-diagonal and not block-sparse any more:

$$\begin{pmatrix} \frac{\mathbf{V}}{\mathbf{\Delta}\tau_1}\underline{\underline{\mathbf{I}}}+\left.\frac{\partial \mathbf{R}}{\partial \mathbf{W}}\right|_1 & \cdots & \mathbf{V}\underline{\underline{\mathbf{D_t}}}_{1,2N+1} \\ \vdots & \ddots & \vdots \\ \mathbf{V}\underline{\underline{\mathbf{D_t}}}_{2N+1,1} & \cdots & \frac{\mathbf{V}}{\mathbf{\Delta}\tau_{\mathbf{2N+1}}}\underline{\underline{\mathbf{I}}}+\left.\frac{\partial \mathbf{R}}{\partial \mathbf{W}}\right|_{2N+1} \end{pmatrix}^{q} \begin{pmatrix} \Delta\mathbf{W}_1 \\ \vdots \\ \Delta\mathbf{W}_{2N+1} \end{pmatrix} = -\begin{pmatrix} \mathbf{R}_1 \\ \vdots \\ \mathbf{R}_{2N+1} \end{pmatrix}^{q} - \begin{pmatrix} 0 & \cdots & \mathbf{V}\underline{\underline{\mathbf{D_t}}}_{1,2N+1} \\ \vdots & \ddots & \vdots \\ \mathbf{V}\underline{\underline{\mathbf{D_t}}}_{2N+1,1} & \cdots & 0 \end{pmatrix}^{q} \begin{pmatrix} \mathbf{W}_1 \\ \vdots \\ \mathbf{W}_{2N+1} \end{pmatrix}^{q} \tag{6.42}$$

The new shape of the left hand side matrix prevents us from treating separately each instant because the coupling between instants is not relegated only to the right hand side, but is present also on the left.

The first idea could be the treatment of the complete system, that would involve the inversion of a really big and not block-sparse matrix.

This requires evidently a massive storage of data and memory consumption, which is not the best strategy because it would limit the treatable number of instants and the speed of computation.

Another way can be found: a useful property applies the $\underline{\underline{\mathbf{d}_t}}$ matrix (the coefficient matrix, reminded in the eq. 6.43): the $(2N+1)\times(2N+1)$ coefficient matrix has the null main diagonal. This allows the use of an iterative Block-Jacobi method, for the implicitation of the phase of the source term.

$$\underline{\underline{\mathbf{d}_t}}=\begin{pmatrix} 0 & \cdots & d_{t\,1,2N+1} \\ \vdots & \ddots & \vdots \\ d_{t\,2N+1,1} & \cdots & 0 \end{pmatrix} \tag{6.43}$$

It allows the implicit coupling term $\underline{\underline{\mathbf{D_t}}}\mathbf{\Delta W}$ to be moved to the right hand side, leading to $2N+1$ independent linear systems.

Recalling the block Jacobi presentation in chapter 4 for a linear system $\underline{\underline{\mathbf{A}}}\mathbf{x} = \mathbf{b}$, the left hand side matrix is split and the following solution $\mathbf{x}_{k+1}$ is obtained as:

$$\underline{\underline{\mathbf{A}}} = \underline{\underline{\mathbf{D}}} - \underline{\underline{\mathbf{E}}} - \underline{\underline{\mathbf{F}}} \tag{6.44}$$

$$\mathbf{x}_{k+1} = \underline{\underline{\mathbf{D}}}^{-1}(\underline{\underline{\mathbf{E}}} + \underline{\underline{\mathbf{F}}})\mathbf{x}_k + \underline{\underline{\mathbf{D}}}^{-1}\mathbf{b} \tag{6.45}$$

The TSM problem can be split in an analogue way:

$$\underline{\underline{\mathbf{A}}}_{\mathbf{TSM}} = \overbrace{\left(\frac{\mathbf{V}}{\mathbf{\Delta}\tau}\underline{\underline{\mathbf{I}}} + \left.\frac{\partial \mathbf{R}}{\partial \underline{\underline{\mathbf{W}}}}\right|\right)}^{\underline{\underline{\mathbf{D}}}} + \overbrace{\mathbf{V}\underline{\underline{\mathbf{D}}}_t}^{-\underline{\underline{\mathbf{E}}}-\underline{\underline{\mathbf{F}}}} \tag{6.46}$$

$$\mathbf{b}_{\mathbf{TSM}} = -\mathbf{R}_{\mathbf{TSM}} \tag{6.47}$$

$$\mathbf{x}_{\mathbf{TSM}_{k+1}} = \mathbf{\Delta W}_{k+1} = \left(\frac{\mathbf{V}}{\mathbf{\Delta}\tau}\underline{\underline{\mathbf{I}}} + \left.\frac{\partial \mathbf{R}}{\partial \underline{\underline{\mathbf{W}}}}\right|\right)^{-1}(-\mathbf{V}\underline{\underline{\mathbf{D}}}_t)\mathbf{\Delta W}_k + \left(\frac{\mathbf{V}}{\mathbf{\Delta}\tau}\underline{\underline{\mathbf{I}}} + \left.\frac{\partial \mathbf{R}}{\partial \underline{\underline{\mathbf{W}}}}\right|\right)^{-1}(-\mathbf{R}_{\mathbf{TSM}}) \tag{6.48}$$

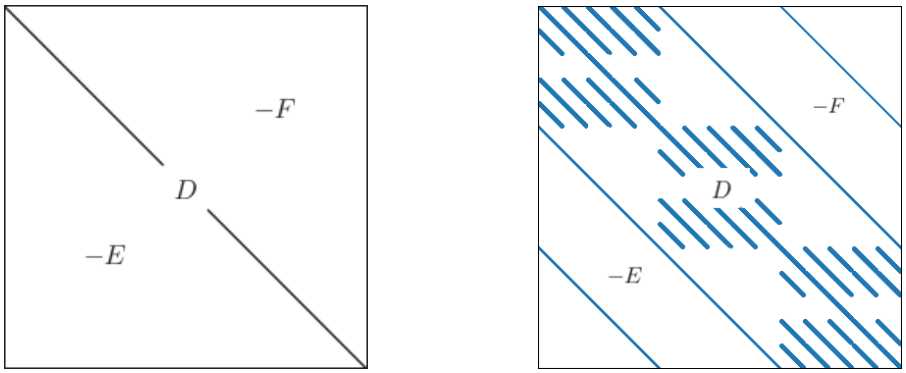


Figure 6.5: $\underline{\underline{\mathbf{A}}}$ - general system, at the left, and $\underline{\underline{\mathbf{A}}}_{\mathbf{TSM}}$ - TSM system, 3 instants, at the right

And an internal Jacobi step finally consists in :

$$\left(\frac{\mathbf{V}}{\mathbf{\Delta}\tau}\underline{\underline{\mathbf{I}}} + \left.\frac{\partial \mathbf{R}}{\partial \underline{\underline{\mathbf{W}}}}\right|_q\right)\mathbf{\Delta W}^q_{k+1} = -\mathbf{R}_{\mathbf{TSM}}(\mathbf{W}^q) - \mathbf{V}\underline{\underline{\mathbf{D_t}}}\mathbf{\Delta W}^q_k \tag{6.49}$$

For every block - Jacobi step, a linear system has to be solved for k_{max} times with the usual methods, that can be direct or iterative solvers (LU direct solver is currently used, but it will be replaced by GMRES in the prosecution of the work).

At the end of the Jacobi iterations, when k_{max} is reached, the vector of solution of the conservative variables for the external iteration $q+1$ is computed as

$$\mathbf{W}^{q+1} = \mathbf{W}^q + \mathbf{\Delta W}^q_{k_{max}} \tag{6.50}$$

The initial vector of the increments for $k = 0$ is initialized at 0 for every q-iteration, $\Delta W^0 = 0$. It is evident that the minimum number of block-Jacobi iterations must be greater or equal to 2, because the first block-Jacobi iteration corresponds to an explicit step.

Hence the solution algorithm, for each of the sampled instants, becomes:

---

inputs: $\mathbf{W}^q$ known from previous iteration q, $k_{max} \geq 2$, $\mathbf{\Delta W}_0^q = 0$
for $k = 0$ to $k_{max} - 1$:
  solve $\left(\dfrac{\mathbf{V}}{\mathbf{\Delta}\tau} \underline{\underline{\mathbf{I}}} + \left. \dfrac{\partial \mathbf{R}}{\partial \underline{\underline{\mathbf{W}}}} \right|^q \right) \mathbf{\Delta W}_{k+1}^q = -\mathbf{R_{TSM}}(\mathbf{W}^q) - \mathbf{V}\underline{\underline{\mathbf{D_t}}}\mathbf{\Delta W}_k^q$
  obtain $\mathbf{\Delta W}_{k+1}^q$
  compute the coupling term $\mathbf{V}\underline{\underline{\mathbf{D_t}}}\mathbf{\Delta W}_{k+1}^q$
  update the increment $\mathbf{\Delta W}_k^q = \mathbf{\Delta W}_{k+1}^q$
end for
output: $\mathbf{\Delta W}_{k_{max}}^q$
update with the last increment $\mathbf{W}^{q+1} = \mathbf{W}^q + \mathbf{\Delta W}_{k_{max}}^q$

In the parallel computing no consistent change is made: the parallel solution of the linear system for each instants still remains, but the external Jacobi iterations are performed sequentially, due to the impossibility to exchange data in a *Multiprocessing* approach. To speed up this implementation, the use of *Message Passing Interface* to compute in parallel the instants will be necessary.

6.6 Jacobian Matrix

A short overview of the structure of the Jacobian matrix extracted is presented, being it a key factor in our equations. Plots of the stencil of the Jacobian matrix for one instant (or the steady simulation) are presented in fig. 6.6 and 6.7.

At the left data are classified under all ρ, after ρu and so on. $(\rho_1, \rho_2, ..., \rho_{Ncells})$, $(\rho u_1, \rho u_2, ..., \rho u_{Ncells})$, $(\rho v_1, \rho v_2, ..., \rho v_{Ncells})$, $(\rho w_1, \rho w_2, ..., \rho w_{Ncells})$, $(\rho E_1, \rho E_2, ..., \rho E_{Ncells})$ (*block of variable* ordering).

In order to facilitate the inversion of the matrix with a direct approach, it is recommended to ensure a diagonal dominance. To do that it is sufficient to reorder the matrix per *cell block*, in which data corresponds to the five variables for the first cell $(\rho, \rho u, \rho v, \rho w, \rho E)_1$, then for the second $(\rho, \rho u, \rho v, \rho w, \rho E)_2$ along the i-curvilinear axis, then along the j-curvilinear axis up to $(\rho, \rho u, \rho v, \rho w, \rho E)_{Ncells}$ (this reordered matrix is at the right in fig. 6.6 and 6.7.).

By the way no evident improvement in terms of computational time for the inversion of the matrix was noticed. So the two orderings are basically equivalent for a direct solver. This would't be true for iterative solvers.

The plots are referred to both the first order Jacobian matrix and the second order Jacobian matrix. It is evident in the reordered plot the contribution of the neighbouring cells in the stencil, which is just one for the first order matrix and two for the second order.

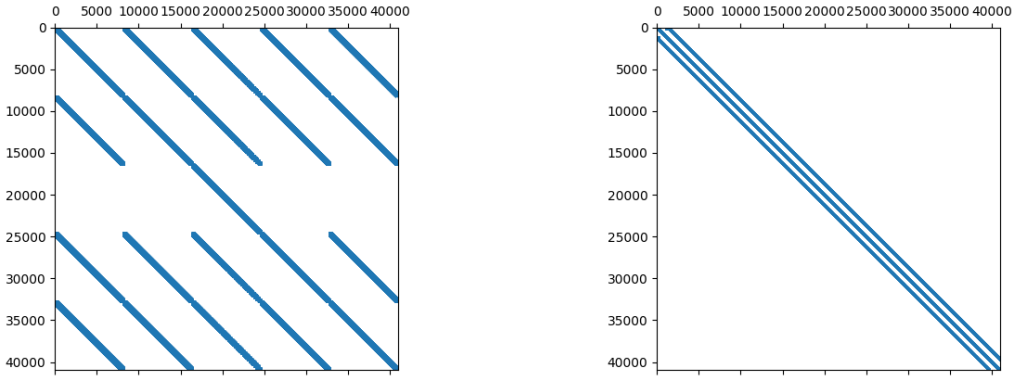


Figure 6.6: Not reordered and reordered Jacobian matrix order 1

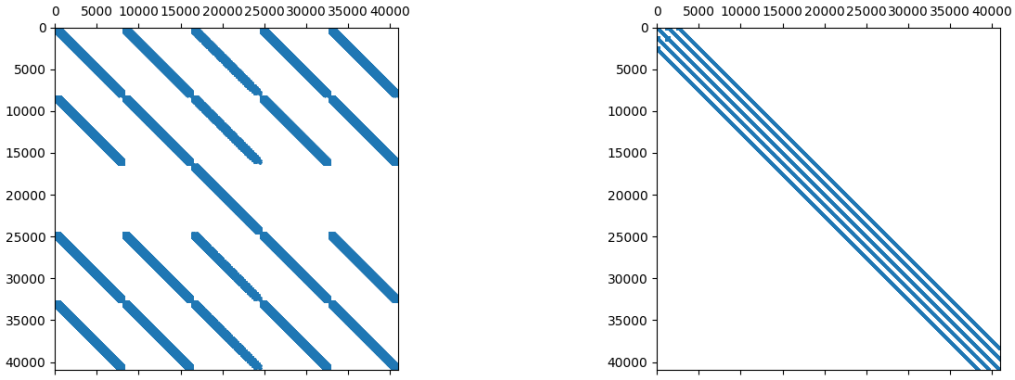


Figure 6.7: Not reordered and reordered Jacobian matrix order 2

Moving to the TSM problem with the coupling of several $2N+1$ instants, the matrix to invert is composed of $2N+1$ diagonal blocks of the initial matrix, with the addition of the extra diagonal terms due to the source term.

The shapes of the two matrices (only for second order matrix), not reordered and reordered, are shown below for a 3 instants case.

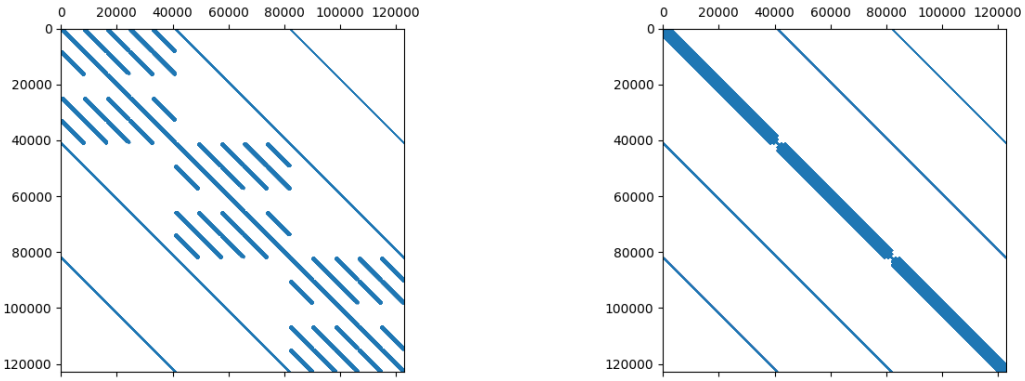


Figure 6.8: Not reordered and reordered left hand side matrix for a 3 instants case - fully implicit

It is straightforward to notice that, increasing the number of sampled instants the matrix becomes larger and larger, and this leads to memory problems already for a case with 5 instants. The aim of the Block Jacobi method is indeed avoiding to invert such huge matrices.

Also, increasing the number of instants, the system loses the diagonal dominance property, and this makes the matrix increasingly harder to invert.

Thanks to Block Jacobi is then possible to solve each step of the sub-iterations k separately for each instant, that will be after coupled by the time-spectral matrix in order to perform the following $k+1$ iteration, and so on.

Chapter 7

Reference data

The validation of the model is subject to the comparison of results from the new external solver with some reference data, that need to be chosen accurately.

Thus a preliminary study of test cases needs to be carried out. Several numerical computations referred to the two chosen test cases will be performed by the CFD solver *elsA* using both the formulations *Block Mobile* and *ALE*, and both the *unsteady* computation and the *TSM* (also called here *HBT*) computation. The results are compared also with the experimental data, and these cross checks will allow to chose the reliable reference data to use to validate the model.

The NASA papers provide the distribution of the pressure coefficient c_p around the upper and lower surface of the airfoil for the steady case, and the *unsteady* pressure coefficient for the unsteady case, consisting in a mean value, a real part and an imaginary part. In order to be able to compare the results of the unsteady simulations performed using *elsA*, which returns instantaneous values of the pressure $p(t)$ at each position of the chord as output, computing the unsteady pressure distribution over the wall is needed. In the following section a demonstration of what has been implemented in a Fortran post-processing tool is provided.

7.1 Unsteady pressure calculation - Fourier Analysis

The *elsA* code is provided with an internal post-processing tool, which extracts the harmonics of the pressure signal, only for the *ALE* formulation and not for the mobile *block formulation*. Therefore it is necessary to build an external post-processing tool, able to compute the unsteady pressure for both the cases, especially for the mobile block formulation, that will be validated with the elsA's reconstruction of the unsteady signal available for ALE formulation.

7.1.1 *Mobile block* formulation

The simulation returns the instantaneous values of the pressure $p(t)$ at each position of the chord as output, and it has the function $q(t)$ as input.

$$q(t) = \alpha \sin(\omega t) \tag{7.1}$$

with α and ω the amplitude and the frequency (pulsation) of the forced pitching motion. In this case the initial phase is set to 0 in the *elsA* simulation).

It is necessary to specify that $\hat{p}$ is not straightly pressure, but pressure transfer function (*input* divided by *output*) at each x-coordinate, while $p(t)$ is the local pressure at each x-coordinate.

$$\hat{p} = \frac{\frac{1}{T}\int_{t_0}^{t_0+T} p(t)e^{-i\omega t}\,\mathrm{d}t}{\frac{1}{T}\int_{t_0}^{t_0+T} \alpha \sin\omega t e^{-i\omega t}\,\mathrm{d}t} = \tag{7.2}$$

$$= \frac{\int_{t_0}^{t_0+T} p(t)e^{-i\omega t}\,\mathrm{d}t}{\int_{t_0}^{t_0+T} \alpha \sin\omega t(\cos\omega t - i\sin\omega t)\,\mathrm{d}t} = \tag{7.3}$$

$$= \frac{\int_{t_0}^{t_0+T} p(t)e^{-i\omega t}\,\mathrm{d}t}{\int_{t_0}^{t_0+T} \alpha \left(\sin\omega t\cos\omega t - i\sin^2\omega t\right)\,\mathrm{d}t} = \tag{7.4}$$

$$= \frac{\int_{t_0}^{t_0+T} p(t)e^{-i\omega t}\,\mathrm{d}t}{-i\alpha\int_{t_0}^{t_0+T} \frac{1-\cos 2\omega t}{2}\,\mathrm{d}t} = \tag{7.5}$$

$$= \frac{\int_{t_0}^{t_0+T} p(t)e^{-i\omega t}\,\mathrm{d}t}{-i\alpha\frac{T}{2}} = \tag{7.6}$$

$$= i\frac{2}{\alpha}\frac{1}{T}\int_{t_0}^{t_0+T} p(t)e^{-i\omega t}\,\mathrm{d}t \tag{7.7}$$

$$= i\frac{2}{\alpha}\frac{1}{T}\int_{t_0}^{t_0+T} p(t)(\cos\omega t - i\sin\omega t)\,\mathrm{d}t \tag{7.8}$$

Passing to the discrete formulation, one can replace the integral with the sum, by choosing a time step Δt and adding the terms of the sum for the j index from the initial instant $t_0 = n_0\Delta t$ to the final one $t_0 + T = (n_0 + (2N+1))\Delta t$, with $(2N+1)\Delta t = T$.

$$\hat{p} = i\frac{2}{\alpha}\frac{1}{(2N+1)}\sum_{j=n_0}^{n_0+(2N+1)} p(j)\left(\cos\frac{2\pi j}{2N+1} - i\sin\frac{2\pi j}{2N+1}\right)\Delta t = \tag{7.9}$$

$$= Re(\hat{p}) + iIm(\hat{p}) \tag{7.10}$$

with:

$$Re(\hat{p}) = \frac{2}{\alpha}\frac{1}{2N+1}\sum_{j=n_0}^{n_0+(2N+1)} p(j)\sin\frac{2\pi j}{2N+1} \tag{7.11}$$

$$Im(\hat{p}) = \frac{2}{\alpha}\frac{1}{2N+1}\sum_{j=n_0}^{n_0+(2N+1)} p(j)\cos\frac{2\pi j}{2N+1} \tag{7.12}$$

7.1.2 Simulation using *elsA* with *ALE* implementation

The simulation returns the instantaneous values of the pressure $p(t)$ at each position of the chord as output, and it has the function $q(t)$ as input:

$$q(t) = \alpha\cos(\omega t + \phi) \tag{7.13}$$

with α and ω the amplitude and the frequency (pulsation) of the forced pitching motion, and ϕ the initial phase (set to $\frac{\pi}{2}$ in the *elsA* simulation).

$$\hat{p} = \frac{\frac{1}{T}\int_{t_0}^{t_0+T} p(t)e^{-i\omega t}\,\mathrm{d}t}{\frac{1}{T}\int_{t_0}^{t_0+T} \alpha\cos(\omega t+\phi)e^{-i\omega t}\,\mathrm{d}t} = \tag{7.14}$$

$$= \frac{\int_{t_0}^{t_0+T} p(t)e^{-i\omega t}\,\mathrm{d}t}{\int_{t_0}^{t_0+T} \alpha\cos(\omega t+\phi)(\cos\omega t - i\sin\omega t)\,\mathrm{d}t} = \tag{7.15}$$

$$= \frac{\int_{t_0}^{t_0+T} p(t)e^{-i\omega t}\,\mathrm{d}t}{\alpha\int_{t_0}^{t_0+T}(\cos\phi\cos^2\omega t - \sin\phi\sin\omega t\cos\omega t - i\cos\phi\sin\omega t\cos\omega t + i\sin\phi\sin^2\omega t)\,\mathrm{d}t} = \tag{7.16}$$

$$= \frac{\int_{t_0}^{t_0+T} p(t)e^{-i\omega t}\,\mathrm{d}t}{\alpha\left(\cos\phi\int_{t_0}^{t_0+T}\frac{1+\cos 2\omega t}{2}\,\mathrm{d}t + i\sin\phi\int_{t_0}^{t_0+T}\frac{1-\cos 2\omega t}{2}\,\mathrm{d}t\right)} = \tag{7.17}$$

$$= \frac{\int_{t_0}^{t_0+T} p(t)e^{-i\omega t}\,\mathrm{d}t}{\alpha\frac{T}{2}e^{i\phi}} = \tag{7.18}$$

$$= \frac{2}{\alpha}e^{-i\phi}\frac{1}{T}\int_{t_0}^{t_0+T} p(t)e^{-i\omega t}\,\mathrm{d}t = \tag{7.19}$$

$$= \frac{2}{\alpha}e^{-i\phi}\frac{1}{T}\int_{t_0}^{t_0+T} p(t)(\cos\omega t - i\sin\omega t)\,\mathrm{d}t \tag{7.20}$$

Passing to the discrete formulation, one can replace the integral with the sum, by choosing a time step Δt and adding the terms of the sum for the j index from the initial instant $t_0 = n_0\Delta t$ to the final one $t_0 + T = (n_0 + N)\Delta t$, with $N\Delta t = T$.

$$\hat{p} = \frac{2}{\alpha}e^{-i\phi}\frac{1}{T}\sum_{j=n_0}^{n_0+(2N+1)} p(j)\left(\cos\frac{2\pi j}{2N+1} - i\sin\frac{2\pi j}{2N+1}\right)\Delta t = \tag{7.21}$$

$$= \frac{2}{\alpha}e^{-i\phi}\frac{1}{2N+1}\sum_{j=n_0}^{n_0+N} p(j)\left(\cos\frac{2\pi j}{2N+1} - i\sin\frac{2\pi j}{2N+1}\right) = \tag{7.22}$$

$$= \frac{2}{\alpha}(\cos\phi - i\sin\phi)\frac{1}{2N+1}\sum_{j=n_0}^{n_0+(2N+1)} p(j)\left(\cos\frac{2\pi j}{2N+1} - i\sin\frac{2\pi j}{2N+1}\right) = \tag{7.23}$$

$$= Re(\hat{p}) + iIm(\hat{p}) \tag{7.24}$$

with:

$$Re(\hat{p}) = \frac{2}{\alpha}\frac{1}{2N+1}\left(\cos\phi\sum_{j=n_0}^{n_0+(2N+1)} p(j)\cos\frac{2\pi j}{2N+1} - \sin\phi\sum_{j=n_0}^{n_0+(2N+1)} p(j)\sin\frac{2\pi j}{2N+1}\right) \tag{7.25}$$

$$Im(\hat{p}) = -\frac{2}{\alpha}\frac{1}{2N+1}\left(\cos\phi\sum_{j=n_0}^{n_0+(2N+1)} p(j)\sin\frac{2\pi j}{2N+1} + \sin\phi\sum_{j=n_0}^{n_0+(2N+1)} p(j)\cos\frac{2\pi j}{2N+1}\right) \tag{7.26}$$

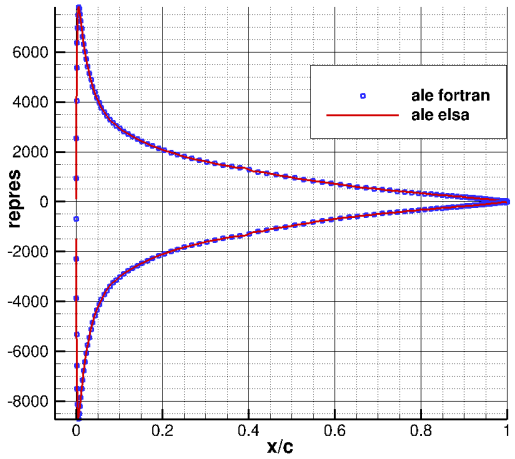


Figure 7.1: Comparison between internal and external post-processing tools - case low Mach number

7.1.3 Validation of the external post-processing tool

In order to validate the external tool aimed to the post-processing of data using the Fourier analysis, a comparison between the one internal to *elsA* and the external one was made in fig. 7.1.

The two curves, relative to the real part of the unsteady pressure on the upper and the lower surface of the airfoil, obtained by two different tools, are completely superposed. Once the post-processing tool has been validated, it is possible to use it, particularly in the *Mobile block* formulation both in the *URANS* and in the *HBT* simulations.

7.2 Case 29 - Low Mach number

In this section the low Mach number simulations are post processed and the results are shown.

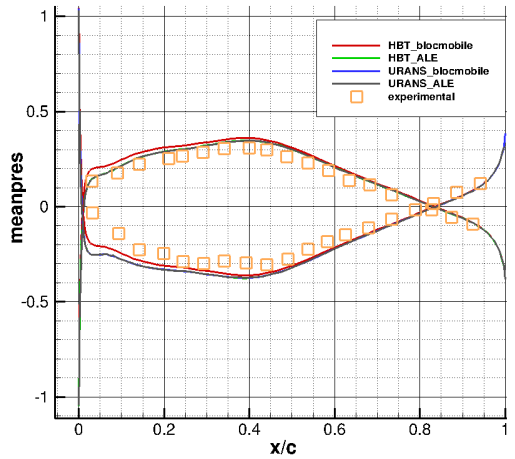


Figure 7.2: Comparison between *Mobile Block* and *ALE* formulation - *HBT* and *URANS* - low Mach number case - mean pressure

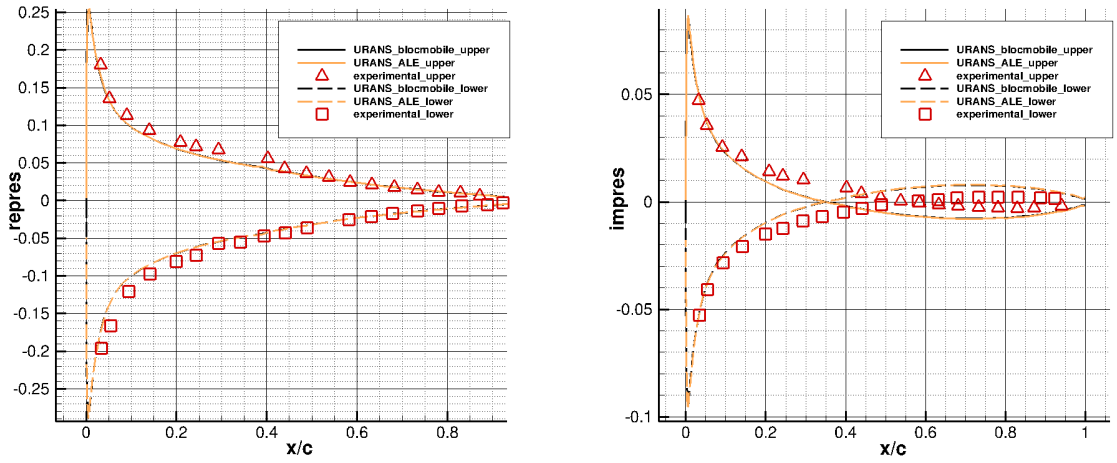


Figure 7.3: Comparison between *Mobile Block* and *ALE* formulation - *URANS* - low Mach number case - real and imaginary part

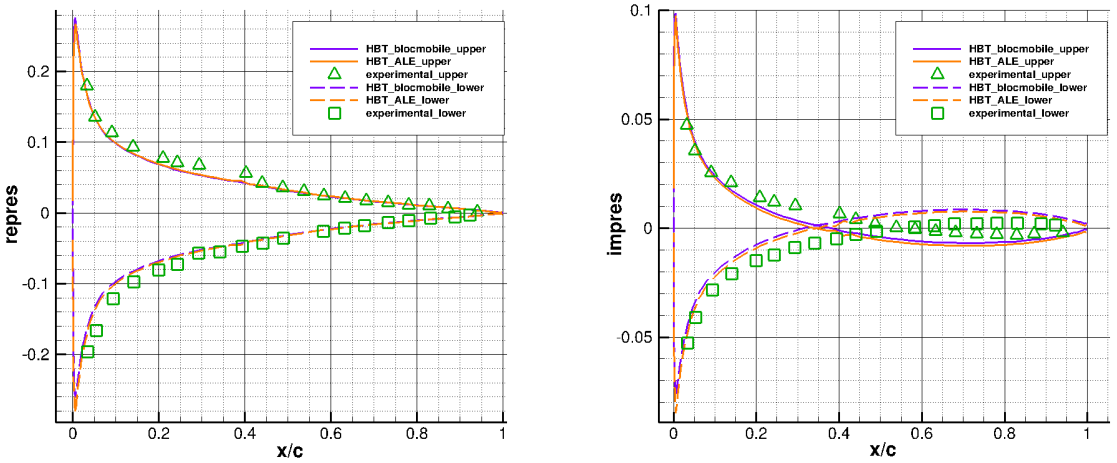


Figure 7.4: Comparison between *Mobile Block* and *ALE* formulation - *HBT* - low Mach number case - real and imaginary part

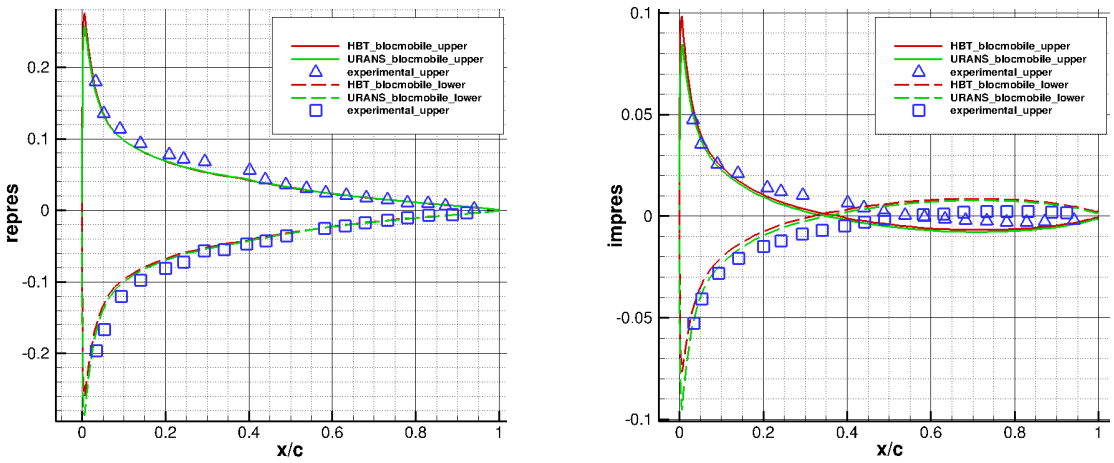


Figure 7.5: Comparison between *URANS* and *HBT* formulation - mobile block - low Mach number case - real and imaginary part

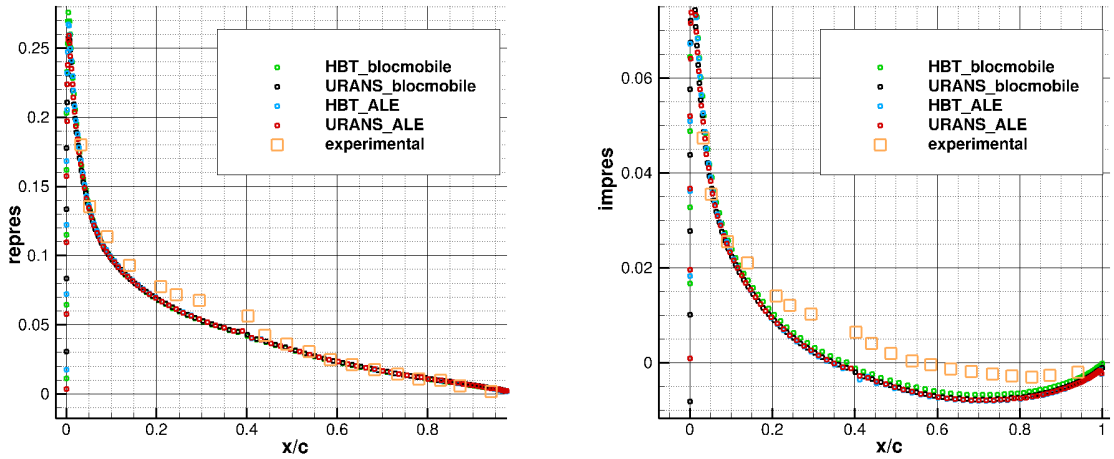


Figure 7.6: Comparison between *Mobile Block* and *ALE* formulation - *HBT* and *URANS* - upper surface - low Mach number case - real and imaginary part

All the results seem in good accordance with the experimental data. In fact, the Mach number being low in this case, the results are not so far from the experimental results in both the URANS and HBT computations performed by elsA.

Anyway it is possible to notice some discrepancies of the HBT - mobile block formulation with respect to the three other methods: this is evident in fig. 7.2 where all the mean pressure distributions are superposed except for the HBT - mobile block. Before performing the high Mach number simulations, we already know that the block mobile formulation in the HBT solution by elsA has not been validated. Only HBT implementation with ALE formulation has been.

7.3 Case 55 - High Mach number

Before analyzing the results from the simulations referred to the high Mach number case, a worse superposition with respect to the low Mach number, is expected, between the experimental data and the numerical ones. The reason is evidently the impossibility to neglect in the Euler fluid equations viscosity and conductibility. In fact in some cases Euler simulations provide rather good results even for transonic regimes. Nevertheless the shock is most of time not located at the same place as experiments. But in any case in the transonic regime the risk of non-negligible effects of fluid viscosity is higher. The possibility to neglect viscosity and conductibility strongly depends also on the incidence: high incidence can yield flow separation even for subsonic cases. Navier-Stokes simulations are then mandatory.

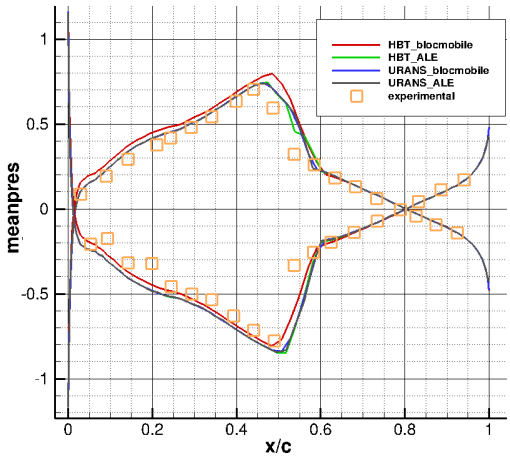


Figure 7.7: Comparison between *Mobile Block* and *ALE* formulation - *HBT* and *URANS* - high Mach number case - mean pressure

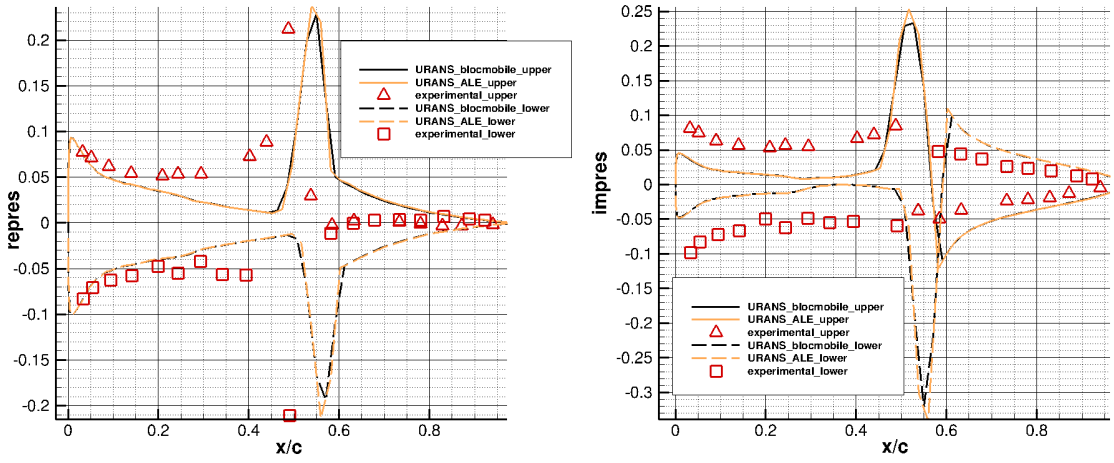


Figure 7.8: Comparison between *Mobile Block* and *ALE* formulation - *URANS* - high Mach number case - real and imaginary part

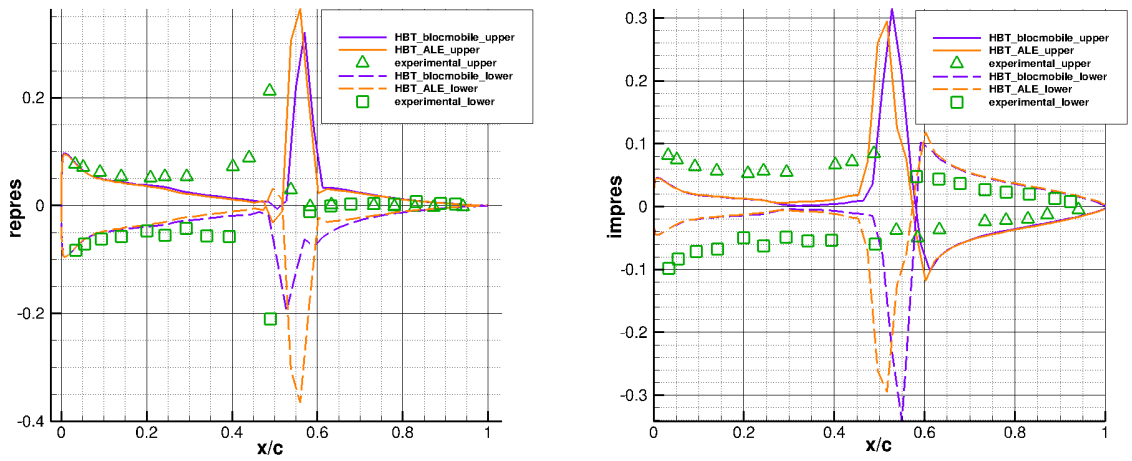


Figure 7.9: Comparison between *Mobile Block* and *ALE* formulation - *HBT* - high Mach number case - real and imaginary part

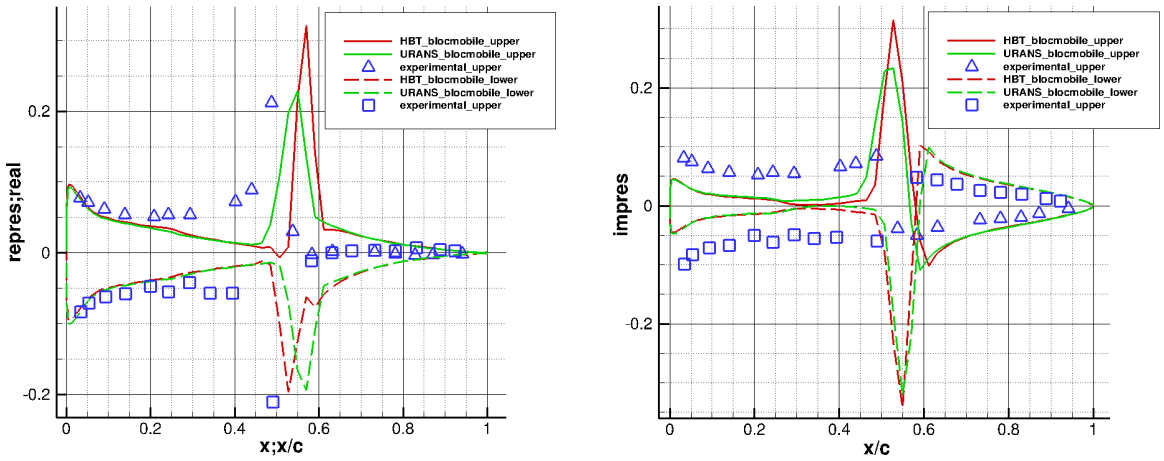


Figure 7.10: Comparison between *URANS* and *HBT* formulation - mobile block - high Mach number case - real and imaginary part

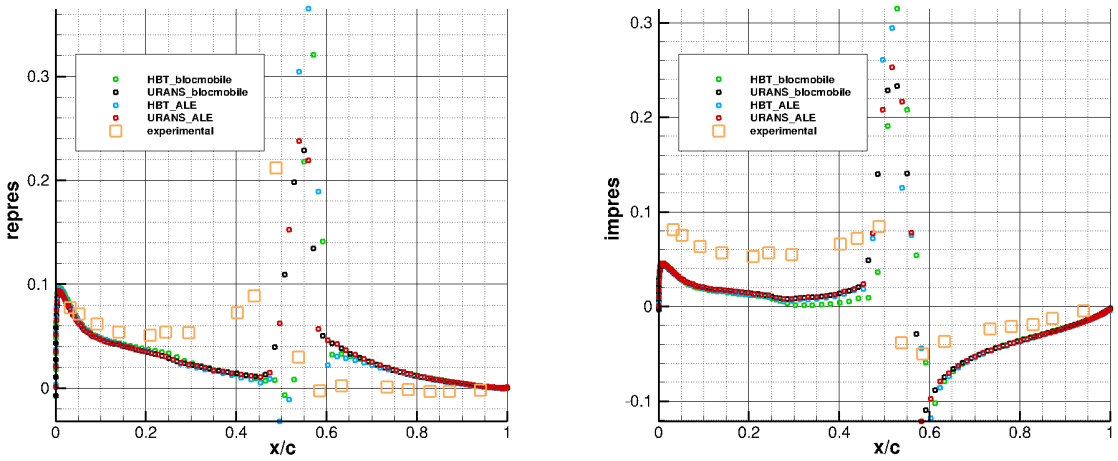


Figure 7.11: Comparison between *Mobile Block* and *ALE* formulation - *HBT* and *URANS* - high Mach number case - real and imaginary parts

It is evident how the differences between numerical results and experimental data are stronger for the real and imaginary part, than for the mean pressure.

Again, only the HBT - mobile block computation shows discrepancies also in the mean pressure field (fig: 7.7). This is a signal of not reliability, because at least a perfect accordance between the ALE and the mobile block HBT computation would be expected, treating them the same problem.

Moreover the effects of neglecting the viscous and heat conduction effects are severe in the shock area, i.e. the zone with strong discontinuities in the pressure field, capturing differently the position and the entity of the discontinuity.

It is to be noticed also that, analyzing a more complex case, presenting a shock wave, the HBT simulations differ consistently from the unsteady ones. In fact the computations have been conducted with the minimum number of instants 3 (so only one harmonic) and these are not sufficient to reconstruct accurately the field of a complex flow.

At the end of the reference data investigation, the unsteady mobile block is chosen as reference, for several reasons:

- It is the standard reference computation used for numerical validations.
- In order to have completely consistent comparisons, a mobile block approach should be used as reference. But the HBT elsA simulations in the mobile block formulation have not been validated yet, then cannot be used as reference.
- The unsteady computation provides results for every instant in the last period, so it's not necessary to perform different reference simulations when the number of sampled instants changes.

Since the TSM in its nature doesn't guarantee perfect accuracy for high Mach number cases with a low number of harmonics, the validations will be done also with the HBT-ALE formulation.

Chapter 8

Results of the external steady solver

The first step in building the solver consisted in the development of a steady solver, that means basically performing only one instant, without the computation of the source term.

The results in terms of rate of convergence and field with respect to elsA unsteady simulation are shown in the following sections.

In the process of validation of the external solver, difficulties of convergence have been encountered for the high Mach number case.

8.1 Low Mach number - case 29

Different CFL strategies must be used to compare elsA and the external solver, because of the different nature of the system to solve: while elsA solves the system using an approximation of the Jacobian matrix of first order, the external solver uses the exact Jacobian matrix of second order. (The order is referred to the order of the approximation due to the space discretization scheme, which says that the error that is introduced by the approximation is proportional to the grid spacing to the power of the order, first, second or above.)

Hence, even though the maximum CFL number is much lower when the external solver is used. This is due to the fact that the second order matrix is harder to invert than the approximation of the first order one. Nevertheless the convergence is improved because an exact matrix is used in the implicit stage.

The best adaptive CFL strategy is shown in the following, that adopts a variable CFL number from 50 to 100, which makes the residuals reach a decrease of almost 10 orders of magnitude in 31 iterations against the 3996 performed by elsA.

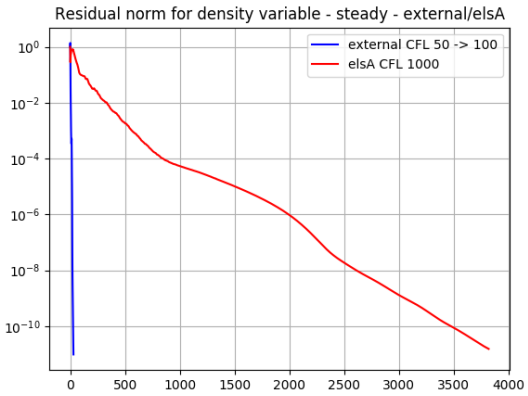


Figure 8.1: Rate of convergence - elsA unsteady, CFL = 1000 - external solver, CFL from 50 to 100

Comparisons of the density field and vertical speed are shown: the resulting fields are completely superposed.

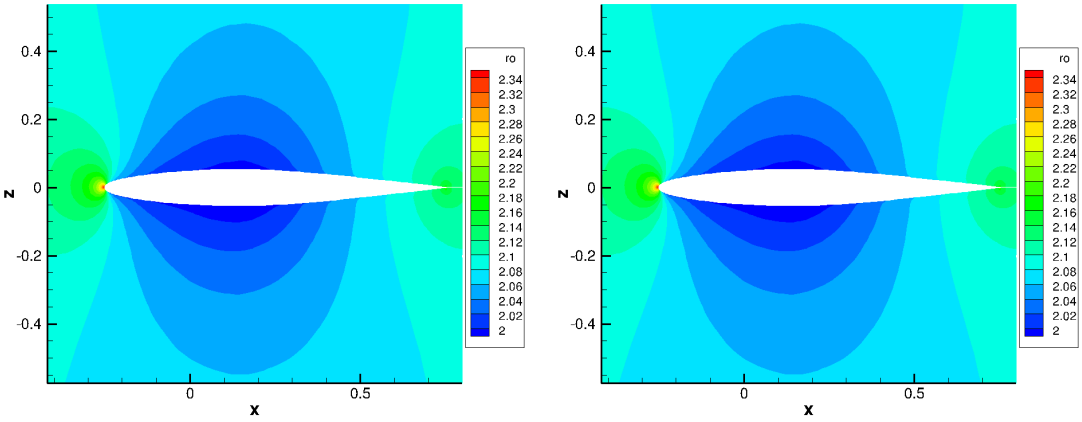


Figure 8.2: ρ field - comparison between the elsA steady solver and the steady external solver

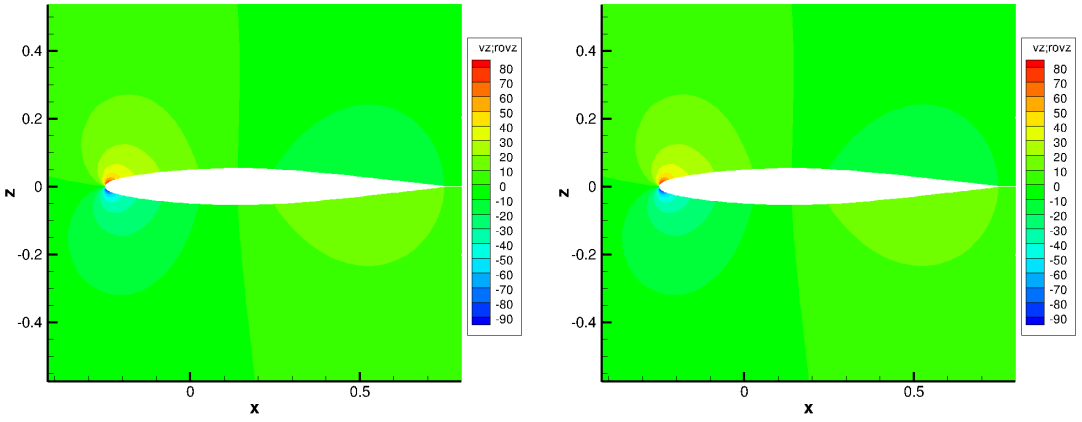


Figure 8.3: u field - comparison between the elsA steady solver and the steady external solver

The same perfect coincidence of the numerical results by elsA and by the external solver are shown also concerning the pressure distribution over the surface of the airfoil.

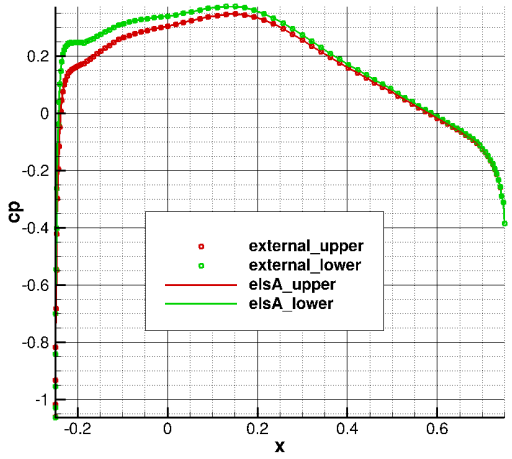


Figure 8.4: Upper and lower pressure coefficients - comparison between the elsA steady solver and the steady external solver

8.2 High Mach number - case 55

The first thing that has been noticed launching the first simulation with the high Mach number is that steady external solver fails: even with a low CFL as 5, already tested for the low Mach number case, after the 1st iteration, the computed field is not physical, with negative density in some cells. It means that the Jacobian matrix in this case is stiffer and needs a very lower CFL to be able to converge, not affordable in terms of practical computational time, especially for the simple steady solver.

Therefore it is presented an owerview of the possible strategies adoptable in order to help the convergence problem, also for the transonic case, that introduces non- linearities in the field (data of the analyzed flow field are in tab. 1.2).

8.2.1 Introduction of a relaxation factor

A first strategy, tried in order to relax the solution, has been the introduction of a relaxation factor in the resolution process:

$$w_{k+1} = w_k + \alpha \Delta w_k \tag{8.1}$$
$$= w_k + \alpha(w_{k+1} - w_k) \tag{8.2}$$
$$= w_k(1-\alpha) + \alpha w_{k+1} \tag{8.3}$$
$$\tag{8.4}$$

instead of the standard Newton process resolution:

$$w_{k+1} = w_k + \Delta w_k \tag{8.5}$$

with very low CFL number $CFL = 0.1$ and relaxation factor $\alpha = 0.3$, to test the effectiveness of the method.

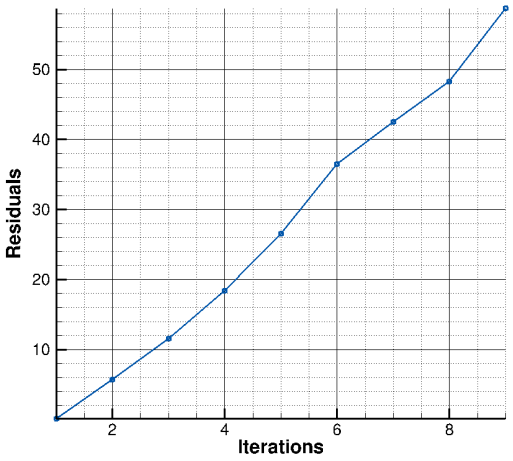


Figure 8.5: Residuals with CFL = 0.1 and $\alpha = 0.3$ - relaxation strategy

Despite the low parameters, the simulation isn't able to converge.

8.2.2 Standard and directional CFL formulation

Another strategy has been tried, the *directional CFL* formulation for the computation of the local pseudo time step.

The standard CFL formulation is the following:

$$\Delta\tau = \frac{CFL \cdot \Delta h}{||U|| + a} \tag{8.6}$$

where the reference length Δh is chosen as the cubic root of the volume of the cell

$$\Delta h = \sqrt[3]{V} \tag{8.7}$$

$||U||$ is the norm of the local speed, computed as

$$||U|| = \sqrt{u^2 + v^2 + w^2} \tag{8.8}$$

and a is the speed of sound.

The directional CFL formulation used as possible strategy is:

$$\Delta t = \frac{CFL \cdot \Delta h}{\rho(\mathbf{n}^{(I)}) + \rho(\mathbf{n}^{(J)}) + \rho(\mathbf{n}^{(K)})} \tag{8.9}$$

where $\rho(\mathbf{n}^{(k)})$ is the spectral radius of the Jacobian matrix associated with the convective flux in the direction $\mathbf{n}^{(k)}$, and it is defined as:

$$\rho(\mathbf{n}^{(k)}) = |n_1^{(k)}u_1 + n_2^{(k)}u_2 + n_3^{(k)}u_3| + a\sqrt{(n_1^{(k)})^2 + (n_2^{(k)})^2 + (n_3^{(k)})^2} \tag{8.10}$$

The reference length used is still the cubic root of the volume and a is still the speed of sound.

Using this new formulation of CFL it is possible to relax the solution, because it takes into account the direction of signal propagation. Indeed, using a CFL=1 the simulation with a standard CFL fails after 5 iterations, while using the directional formulation of CFL the solution process is relaxed.

In fig. 8.6 the behaviour of residuals using a standard CFL=1 and CFL=0.1, and a directional CFL=1 is presented.

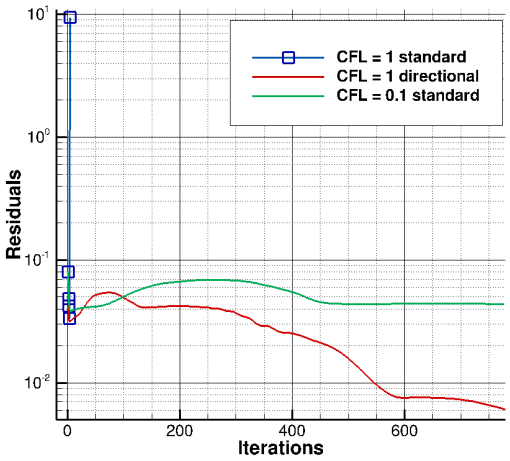


Figure 8.6: Residuals with standard CFL formulation, CFL = 0.1 and CFL =1, and directional CFL formulation, CFL =1

The comparison with the standard CFL=0.1 has been made because in fact the new formulation consists in decreasing the local CFL in cells. It can be noticed that, despite an improvement of the solution with respect to the standard CFL cases, the strategy is anyway insufficient to reach an acceptable convergence (too many iterations with too low decrease of residuals).

Iteration n.1

The behaviour of the pseudo $\Delta\tau$ has been investigated for the three cases after one iteration and only the standard CFL=0.1 and the directional CFL=1 after 700 iterations, in order to see if the shock has altered the $\Delta\tau$ field, in terms of decrease in the proximity of the shock, where the gradients are higher than in other zones.

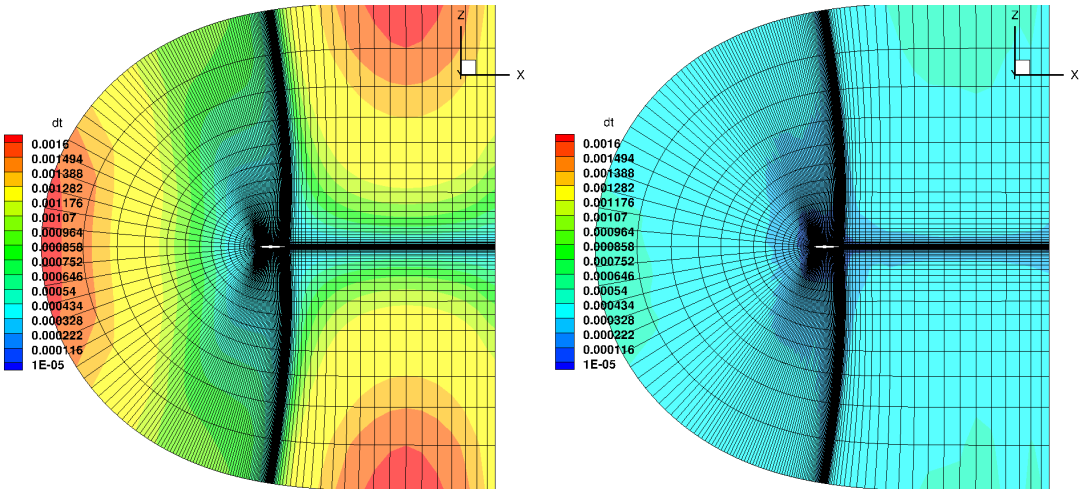


Figure 8.7: $\Delta\tau$ distribution, comparison between standard and directional CFL formulation - CFL = 1

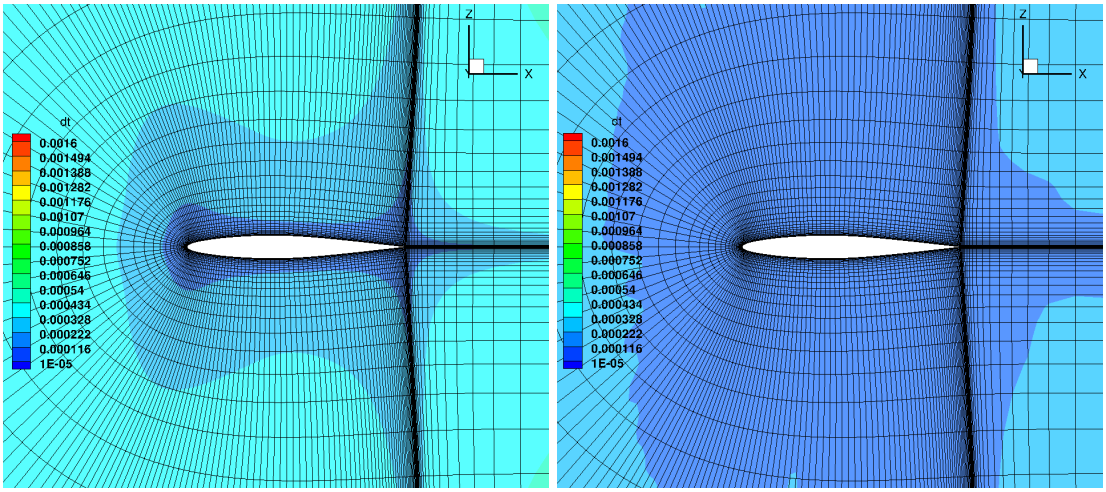


Figure 8.8: $\Delta\tau$ distribution, comparison between standard and directional CFL formulation - CFL = 1 - zoom

The local time steps (CFL is constant everywhere) are evidently lower in the directional case (the colours scale set is the same), but at the first iteration the shock hasn't been caught yet, as expected.

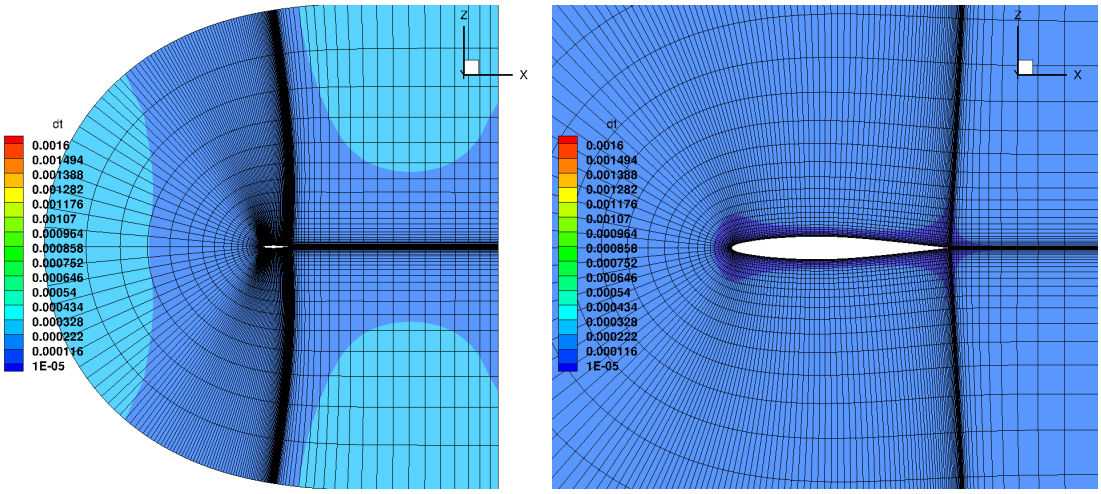


Figure 8.9: $\Delta\tau$ distribution, CFL = 0.1, standard formulation, with zoom

Iteration n.700

The Δt field at iteration n. 700 can be computed, to check if the shock has affected the Δt distribution.

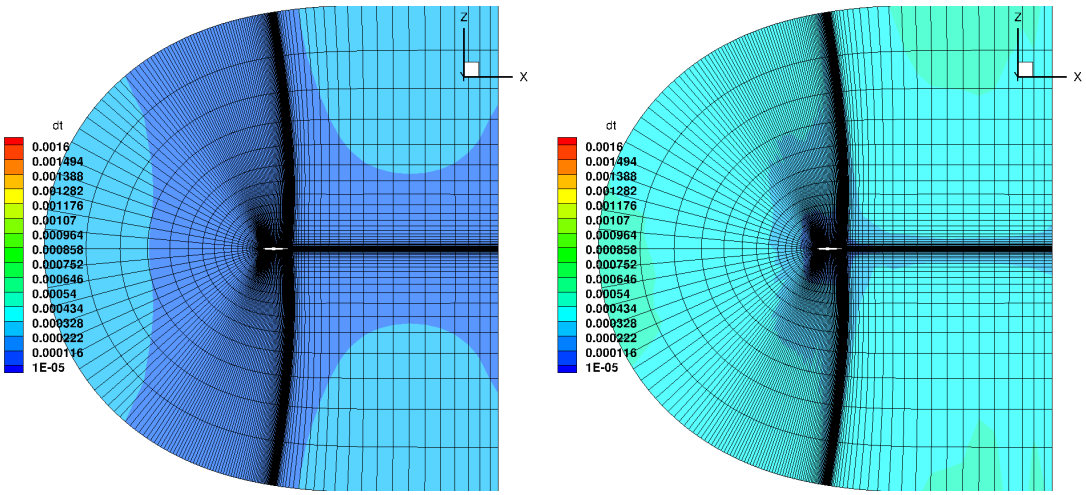


Figure 8.10: $\Delta\tau$ distribution, comparison between standard and directional CFL formulation - standard CFL = 0.1 in the left and directional CFL = 1 in the right at the iteration n.700

The pseudo time step field is not adapted to the shock for any of the simulations, neither the standard formulation strategy with a really low CFL number, nor the directional one.

Therefore we decided to keep the standard formulation, since the results with the directional one simply consist in a decrease of the local time step, achievable also by decreasing the global CFL, without introducing further complications.

8.2.3 Introduction of the first order Jacobian matrix

One main issue with Newton-like algorithms is the determination of the initial field ($\mathbf{W}_0$). This algorithm would be more robust with respect to initialization if the first order Jacobian matrix is used to start, and then the Newton algorithm using the second order one can be launched with the solution of the first order one.

Thus another possible strategy to relax the solution, in order to avoid the quick divergence of the high Mach case, is to use for some iterations the exact Jacobian of spatial first order discretization, instead of the second order matrix, harder to invert than the first order one, being more dense.

First of all, it is found the maximum achievable CFL number to ensure convergence with the first order Jacobian.

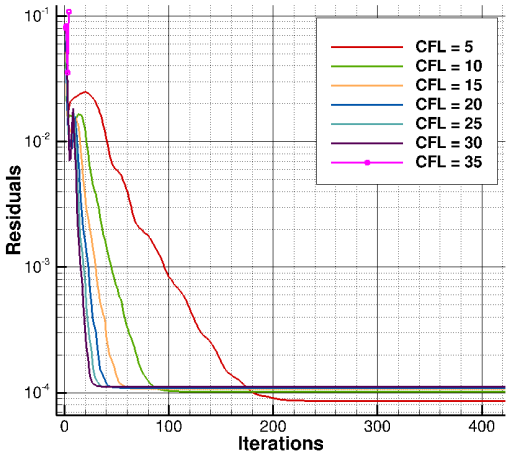


Figure 8.11: Residuals behaviour - comparison among different CFL numbers with Jacobian order 1

From CFL=35 and more, the solution diverges. Therefore as maximum CFL for the first part of the simulation (with Jacobian of first order) CFL=30 is chosen.

Besides it is possible to see a strong plateau for a residual of $\simeq 10^{-4}$, which prevents the residual from decreasing below this value.

In the validation process of the two matrices, consisting in performing the same simulation inverting the two different matrices, in the simulation with first order Jacobian a problem in the wake arose. This behaviour is an explanation of the final stagnation of residuals.

Thus performing a complete simulation with the first order Jacobian matrix (called below JO1) is not possible because its extraction has not been validated yet and possible bugs can remain, but it is useful to give the solution the right direction for convergence, by relaxing the first steps' solution and providing a closer initial guess for the Newton process with second order Jacobian (called below JO2).

At this point it is necessary to restart the simulation with the second order Jacobian. The restart is performed with a CFL=10, for 4 cases:

- 1 iteration with JO1 - switch to JO2,
- 3 iteration with JO1 - switch to JO2,
- 10 iteration with JO1 - switch to JO2, in a central point of the residual ramp
- 25 iteration with JO1 - switch to JO2, when the plateau is almost established

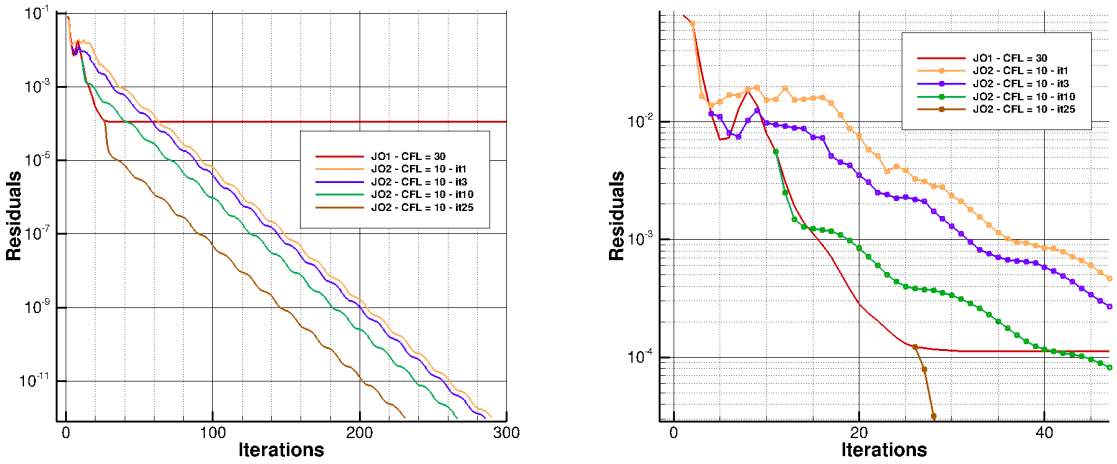


Figure 8.12: Restart with Jacobian order 2 - different restarting points (with zoom)

The main cause of the divergence problem experienced is evidently in the first step of the simulation: indeed applying the switch of the Jacobian matrix even after the first iteration the simulation is able to reach convergence.

In order to understand the physics that are the basis of the numerical behaviour, the density fields after 1st, 3rd, 10th and 25th iteration are presented below:

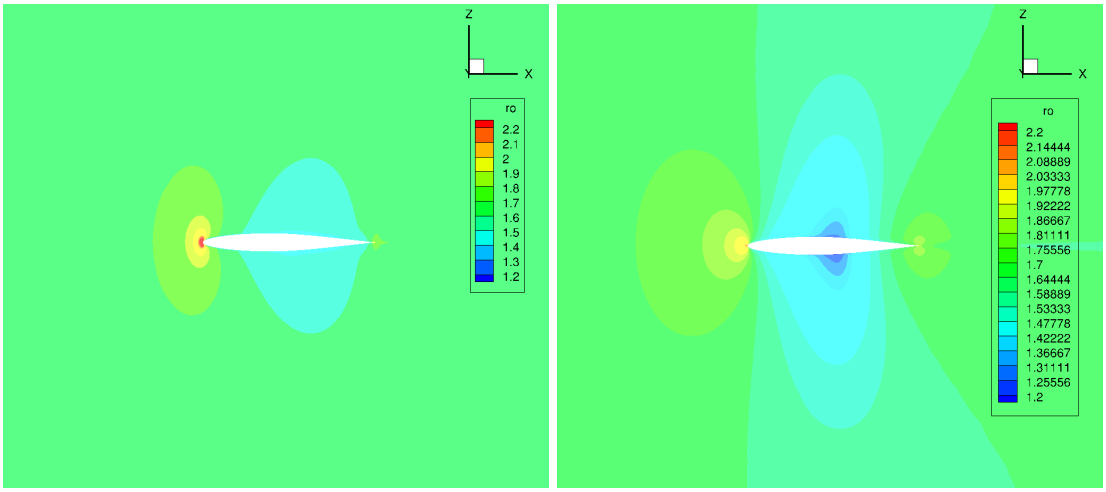


Figure 8.13: Density field at iteration n.1 and n.3

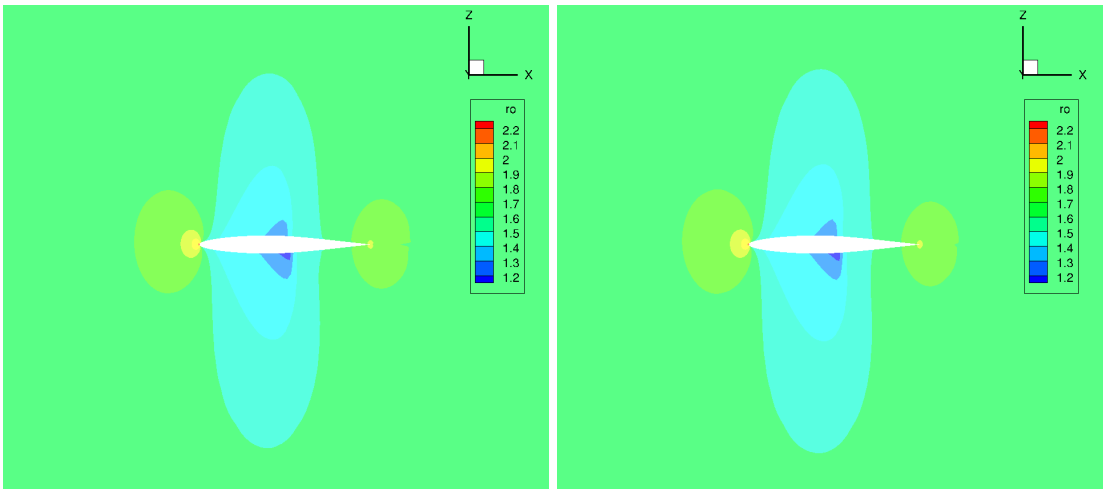


Figure 8.14: Density field at iteration n.10 and n.25

Neither after the first iteration nor after the third the shock has been properly captured, but the field is already changing in both cases and the physics can be captured in a better way. Because of the physical field, that is not completely resolved yet, these two cases present an initial numerical plateau in the residuals after the restart (fig. 8.12), being the restart performed from a field still very far from the solution.

On the other hand, in the third and last case the shock is present in the field, and the beginning of the descent is straight for both the residuals.

It is decided to set the switch around the 10th iteration, since the field is already established (it could be possible also to switch at the first iteration but this strategy wouldn't be conservative enough, while switching too late would mean too many iterations using the first order Jacobian, with the possibility to experience numerical divergence in the wake).

In the attempt to make the process case-independent, it is set a switch from the *First order Jacobian matrix* to the *Second order Jacobian matrix* at the point when a decrease of the residual of one order and three orders of magnitude are experienced.

In the switch point, also the CFL is switched from 30 to 10, as in the case presented before.

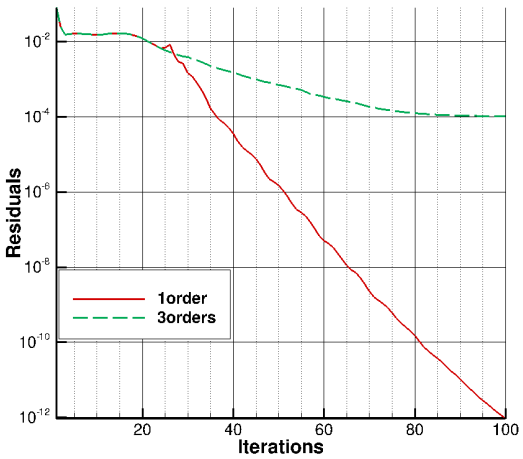


Figure 8.15: Switch point at $\frac{res}{res_{init}} = 0.1$ and $\frac{res}{res_{init}} = 0.001$

The switching point in the first case is the 6th iteration, while in the second case the Jacobian remains first order because of the high residuals.

Assumed the decrease of the residual of one order of magnitude as a good strategy, several tests are done in order to increase the CFL of the simulation for the Jacobian order 2, to 20, 30, 40, 50 and to make the simulation faster.

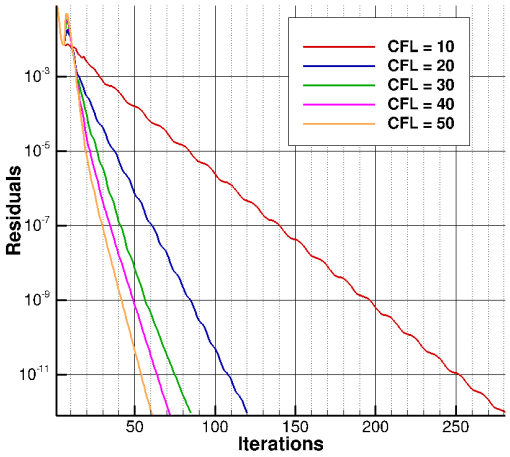


Figure 8.16: Search for the highest affordable CFL number for restart with Jacobian order 2

A good convergence is obtained by all the CFL numbers tested (the peaks after the switch are due to the rapid increasing of the CFL, avoidable by using a ramp between the two values), so the best generalized strategy, in terms of maximizing the speed of the simulation and minimize the utilization of the first order Jacobian matrix seems to be the switch from first order Jacobian to second order Jacobian at the point in which the residual decreases of one order of magnitude, switching also CFL to an higher one (CFL= 50 is chosen for second order Jacobian).

Now a comparison between the external steady solver (using the best strategy found) and elsA steady solver (with a higher CFL=1000), is made in terms of residuals and pressure coefficient distribution.

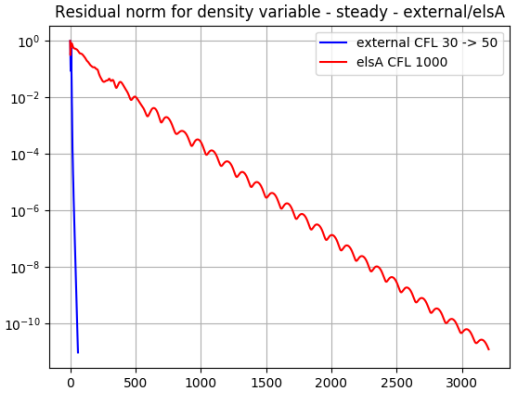


Figure 8.17: Rate of convergence - elsA unsteady, CFL = 1000 - external solver, CFL from 30 to 50 switching the Jacobian matrix

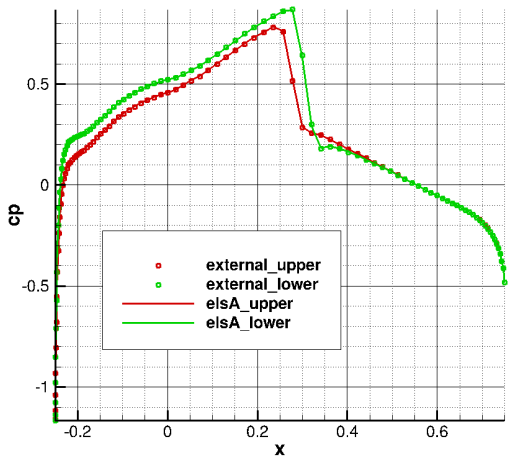


Figure 8.18: Upper and lower pressure coefficients - comparison between the elsA steady solver and the steady external solver

The external solver improved its convergence with respect to the elsA solution process, allowing the simulation to reach the exact solution with the exact Jacobian matrix of second order after only 61 iterations instead of more than 3000 with the steady elsA solver.

The validation of the convergence process is shown below.

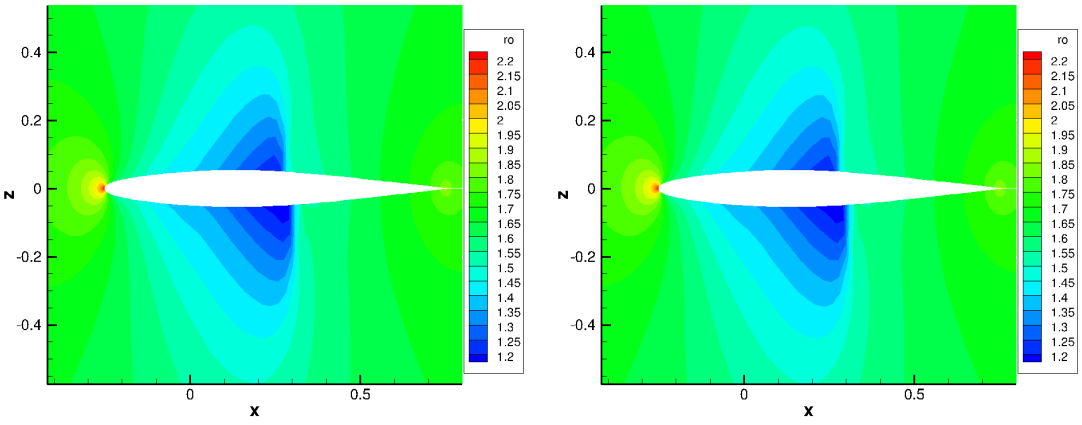


Figure 8.19: ρ field - comparison between the elsA steady solver and the steady external solver

8.2.4 Incomplete factorization in the solution process - ILU

It is also possible to develop a different strategy to avoid the extraction of first order Jacobian during the solution process. An ILU factorization is proposed, with a filling factor = 20, and a dropping tolerance = 0.002 (read chapter 4).

The strategy used in the computation is an adaptive CFL number which increases from 1 to 10, through a ramp from the iteration number 50 to iteration number 200.

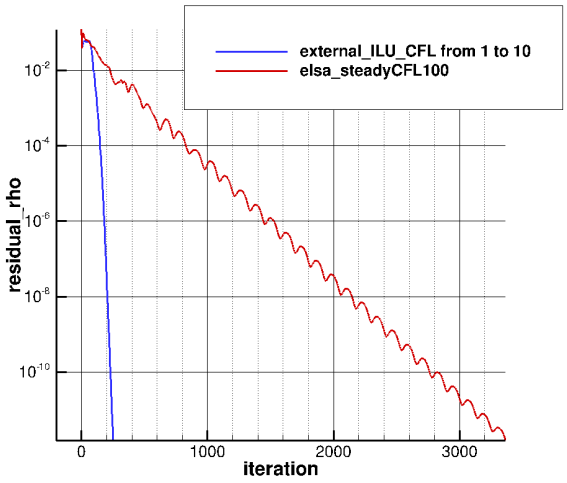


Figure 8.20: Comparison of the residuals behaviour with ILU approach in the external solver, and the elsA steady solver

Also with the ILU factorization it's not possible to achieve a quicker convergence, by using high CFL numbers, since it exploits only the second order Jacobian matrix, less robust than the first order Jacobian, but using this method it is possible to avoid the use of this matrix.

Anyway, despite the convergence achieved by extracting only the second order Jacobian matrix, this method won't be implemented in the steady and TSM codes because it is in fact an approximate solution method, comparable to elsA's one, which doesn't exploit the exact Jacobian matrix for every iteration. Furthermore future releases of the code will implement the GMRES iterative method, so it seems convenient for the moment to keep the direct solver as most simple implementation, facing all the complications that a direct solver can encounter.

Chapter 9

Results of the external TSM solver

Once the results of the steady solver have been validated, it was possible to move to the proper TSM solver. Three cases have been analyzed in this case, the case 29, the case 55 has been split into a case with $f = 10.8Hz$ and a case with $f = 34.4Hz$, because some convergence problems have been met in the high frequency case.

Together with different choices of the CFL strategy, also the behaviour of the convergence with respect to the number of instants sampled has been analyzed.

In each section the results will be proposed as comparisons between the external computation and elsA computation, divided in: the density field visualization, the wall pressure distribution, the aerodynamic coefficients at the given instants with the reconstruction over the period and the convergence of the problem.

9.1 Low Mach number

In this section the resulting fields of the external solver are compared to the unsteady simulation by elsA. As expected in the low Mach number and low frequency case, the process leads to the right solution without convergence issues.

9.1.1 Complete density field around the airfoil - three instants

The density field around the airfoil is shown below.

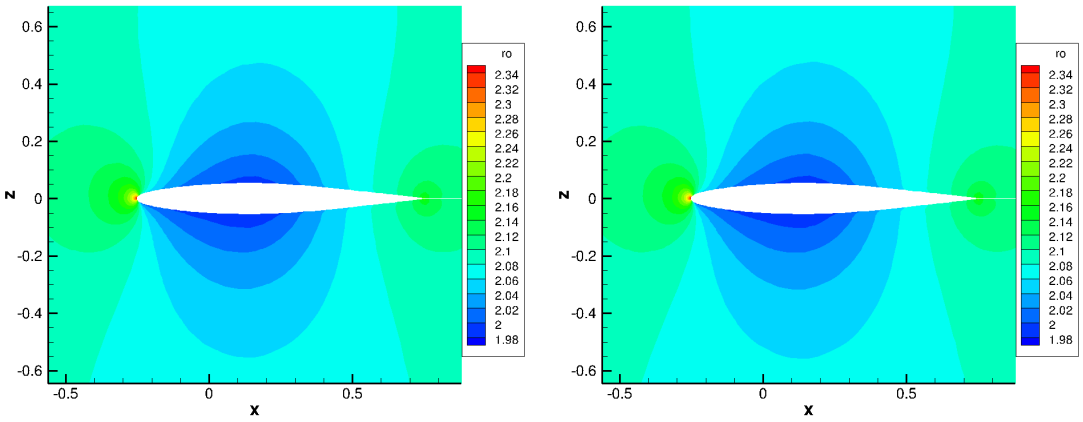


Figure 9.1: ρ field at instant 1 - comparison between the external TSM solver and the elsA unsteady solver

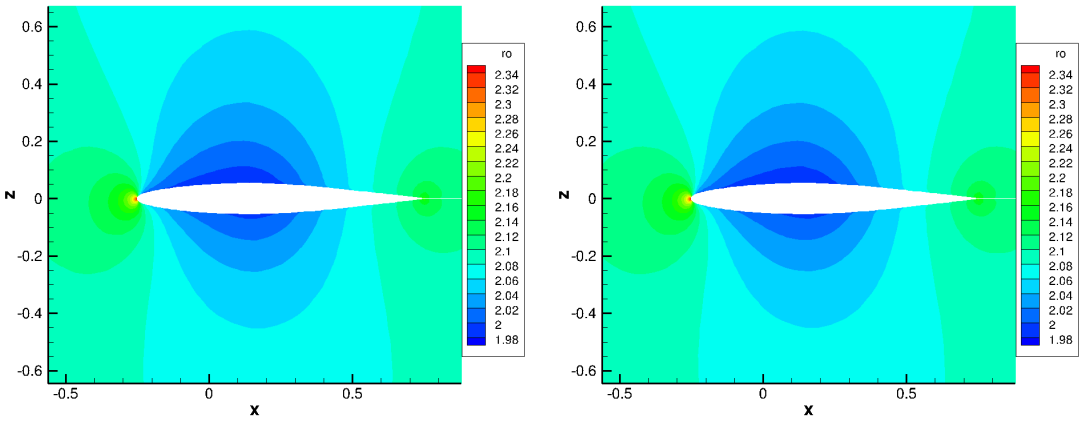


Figure 9.2: ρ field at instant 2 - comparison between the external TSM solver and the elsA unsteady solver

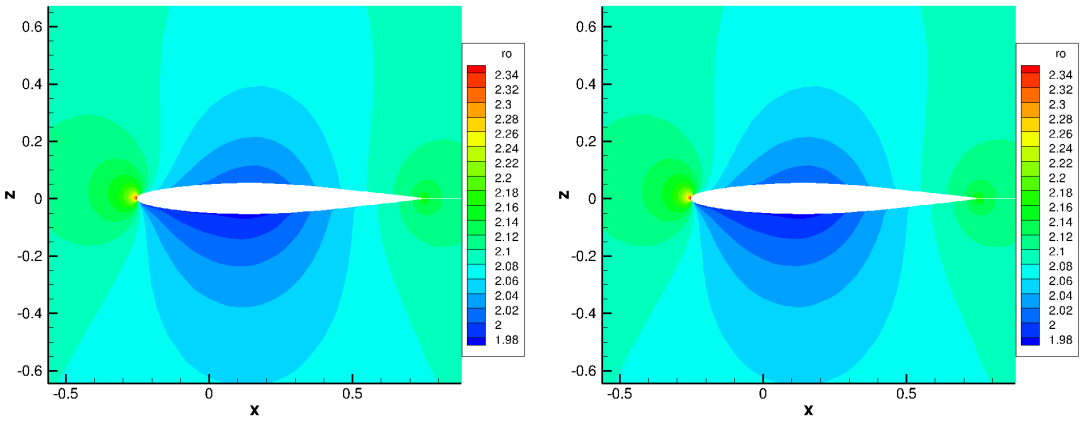


Figure 9.3: ρ field at instant 3 - comparison between the external TSM solver and the elsA unsteady solver

9.1.2 Fields around the stagnation point - three instants

In this section, and only for this case, a zoom of the conservative variables ρ, ρu and ρw around the stagnation point is presented for each instant, in order to show the identical fields and validate the solution.

ρ field

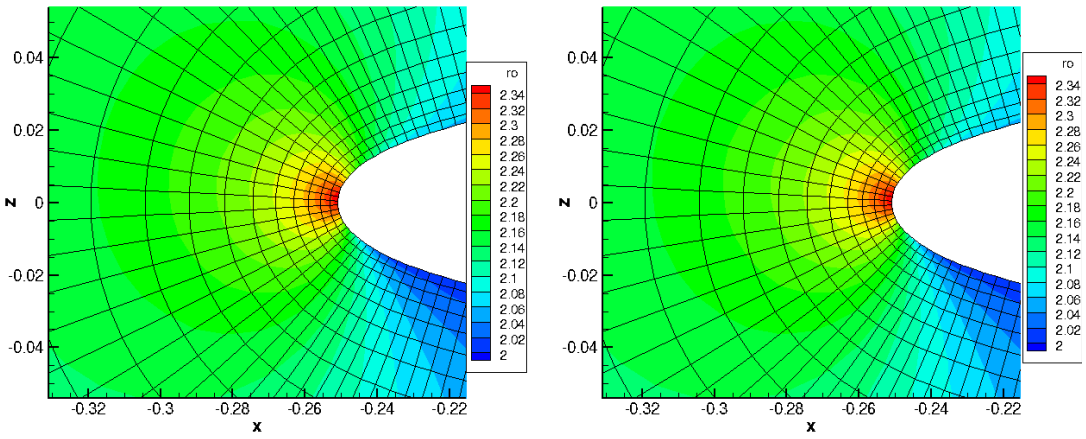


Figure 9.4: ρ field at instant 1 - around stagnation point - comparison between the external TSM solver and the elsA unsteady solver

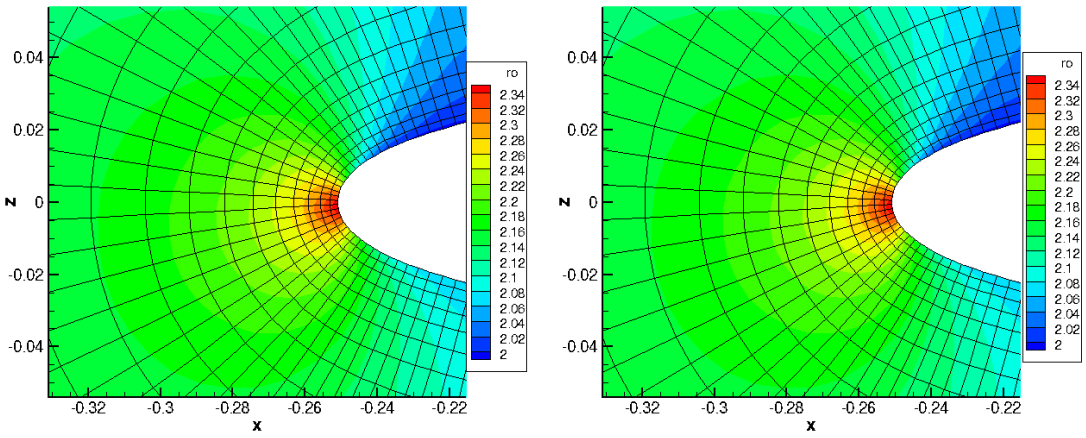


Figure 9.5: ρ field at instant 2 - around stagnation point - comparison between the external TSM solver and the elsA unsteady solver

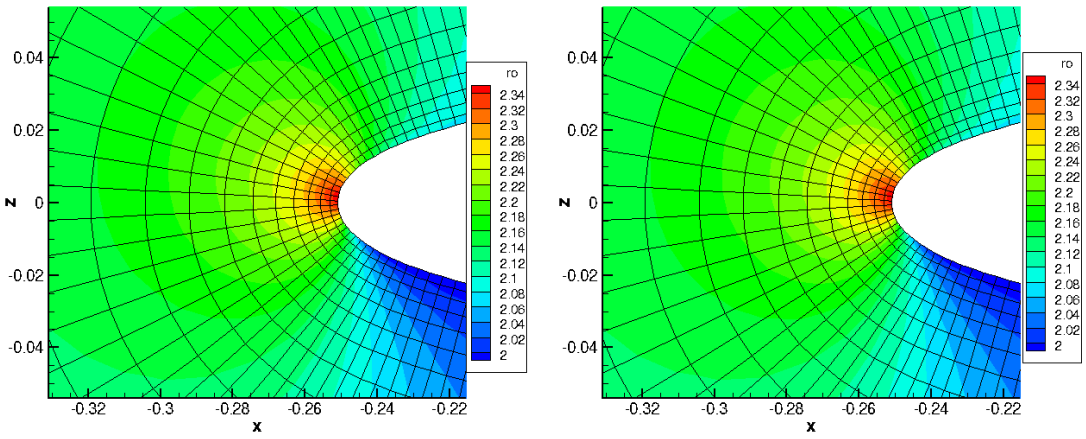


Figure 9.6: ρ field at instant 3 - around stagnation point - comparison between the external TSM solver and the elsA unsteady solver

ρu field

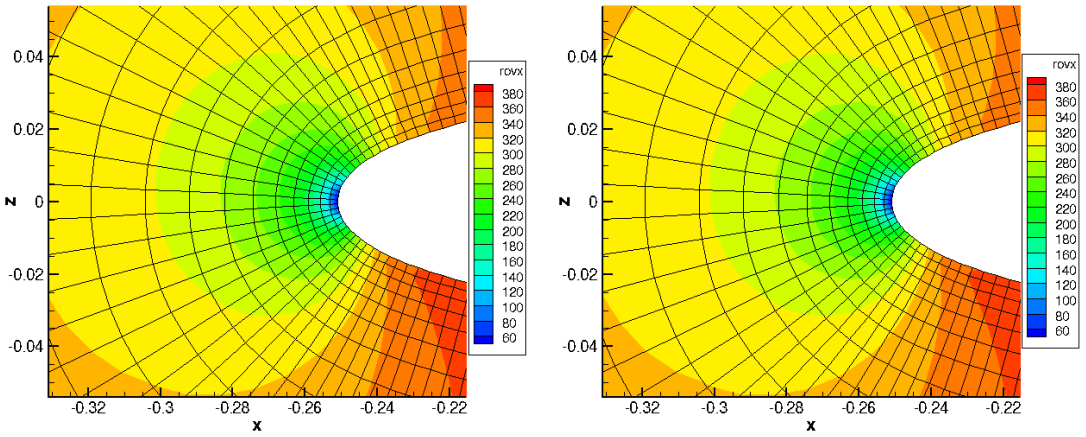


Figure 9.7: ρu field at instant 1 - around stagnation point - comparison between the external TSM solver and the elsA unsteady solver

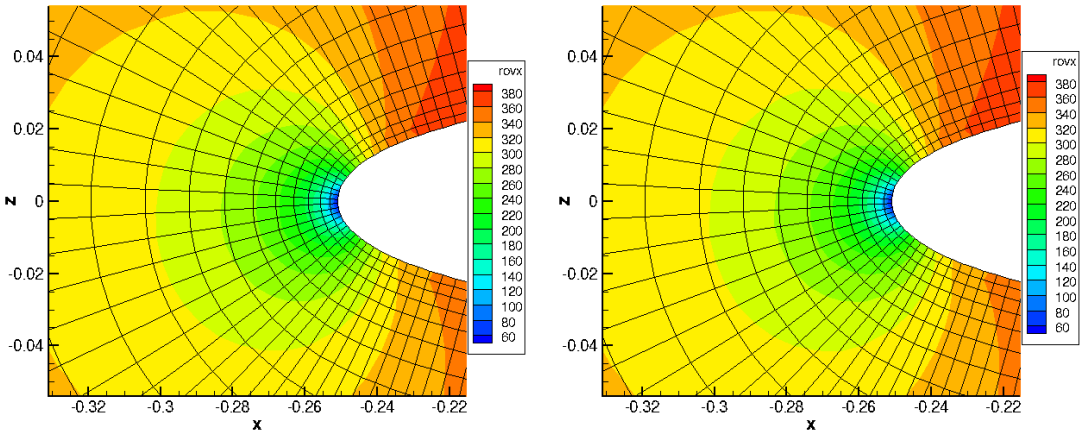


Figure 9.8: ρu field at instant 2 - around stagnation point - comparison between the external TSM solver and the elsA unsteady solver

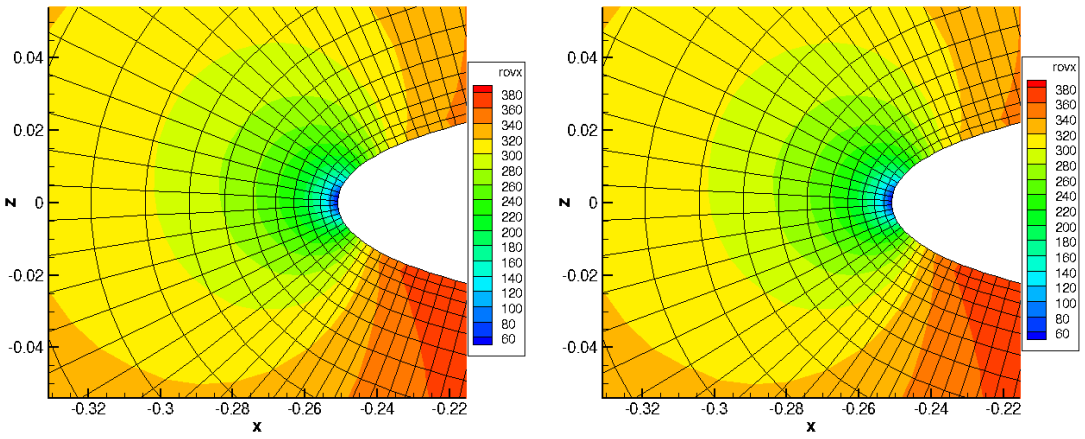


Figure 9.9: ρu field at instant 3 - around stagnation point - comparison between the external TSM solver and the elsA unsteady solver

ρw field

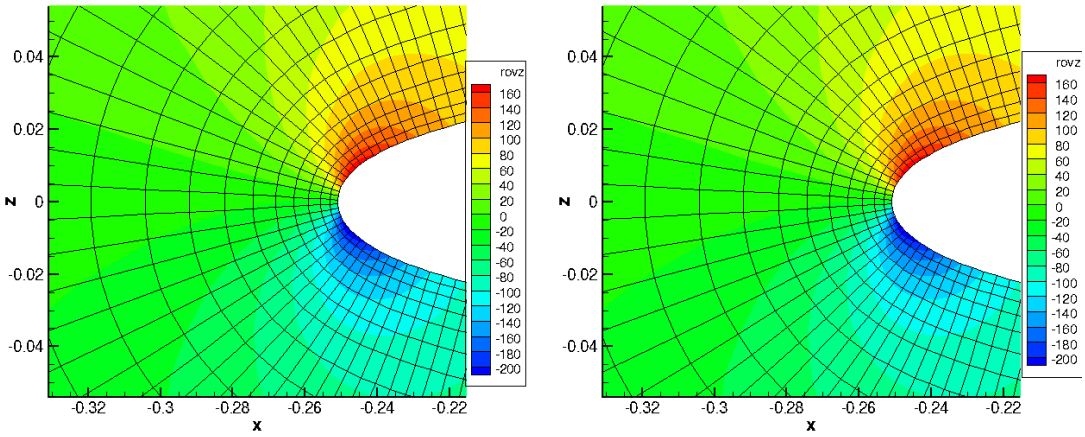


Figure 9.10: ρw field at instant 1 - around stagnation point - comparison between the external TSM solver and the elsA unsteady solver

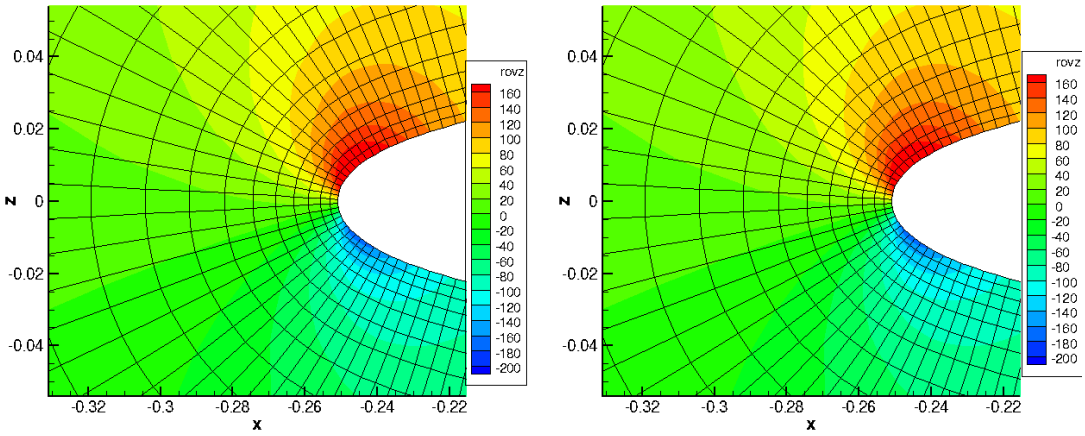


Figure 9.11: ρw field at instant 2 - around stagnation point - comparison between the external TSM solver and the elsA unsteady solver

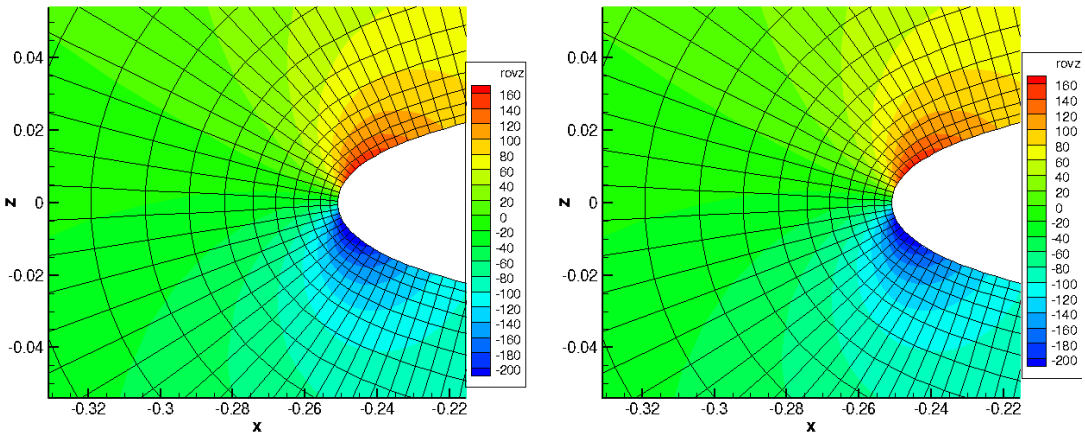


Figure 9.12: ρ field at instant 3 - around stagnation point - comparison between the external TSM solver and the elsA unsteady solver

9.1.3 Wall pressure

In the following, upper and lower pressure at each instant, and the reconstruction of the unsteady pressure for the upper and lower part, are shown for 3, 5, 7 instants, comparing the unsteady reference simulation by elsA and the results by the external solver.

As expected, comparing the wall pressure at the three instants, there is complete accordance between unsteady results and TSM results by the external solver.

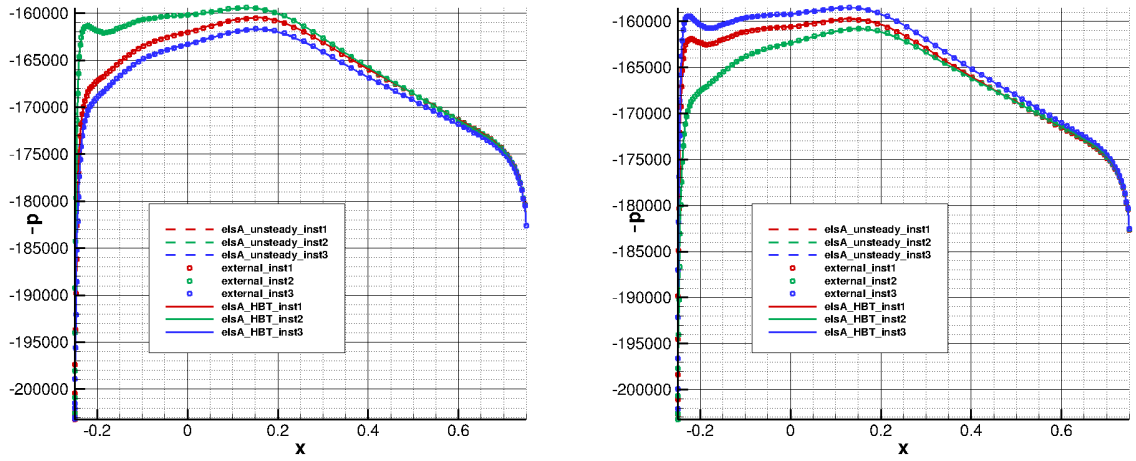


Figure 9.13: Upper and lower pressure at the 3 instants - external TSM solver, elsA TSM solver and elsA unsteady solver

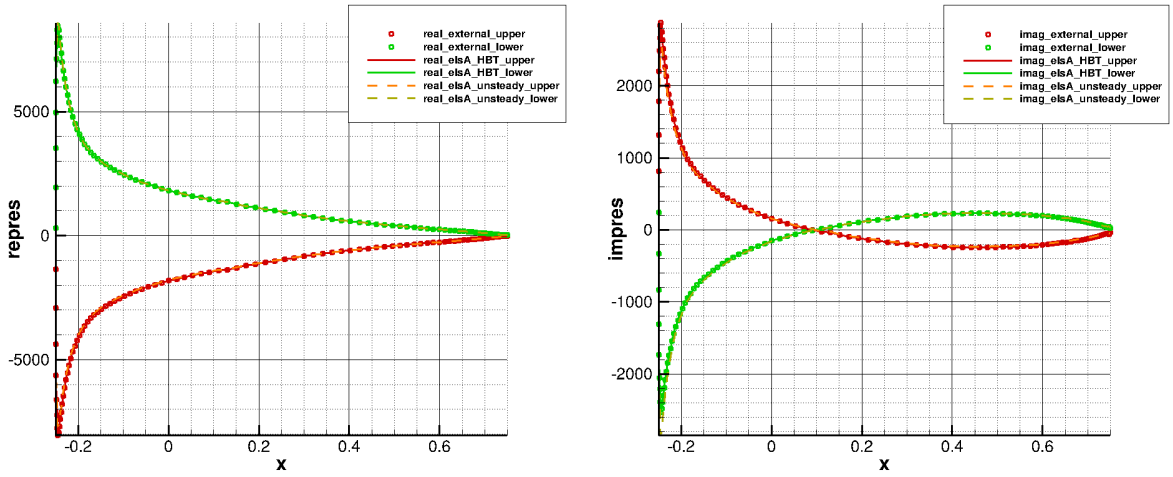


Figure 9.14: Upper and lower unsteady pressure reconstruction - 3 instants - external TSM solver, elsA TSM solver and elsA unsteady solver - real and imaginary part

The accordance between real and imaginary parts of the unsteady pressure on the airfoil wall means that the physics of the problem is well approximated by 3 instants, due to the linearity of the problem with low Mach number and low frequency. It is expected that for a more complex case, 3 instants won't be enough to represent completely the physics of the problem. By the way, especially for the imaginary part near the leading edge (fig. 9.14 at the right), there is a slight discrepancy in the peak at the leading edge that is higher for the TSM case than the unsteady one. It is shown below how it is possible to eliminate this problem using 5 instants, so 2 harmonics.

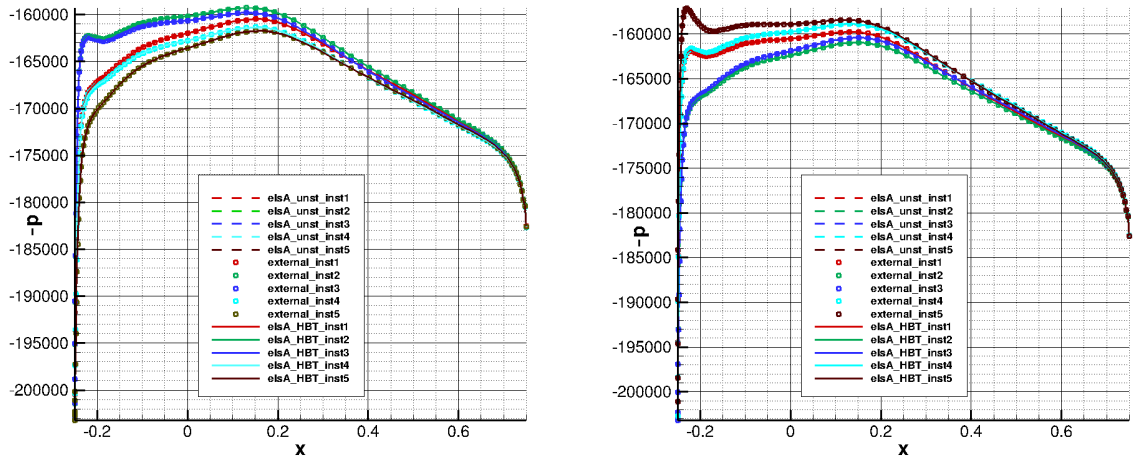


Figure 9.15: Upper and lower pressure at the 5 instants - external TSM solver, elsA TSM solver and elsA unsteady solver

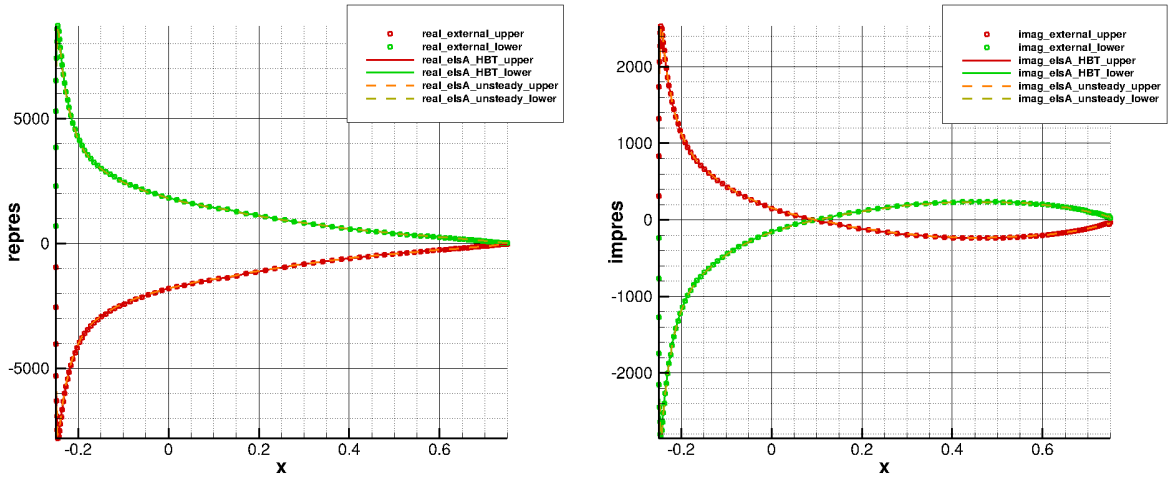


Figure 9.16: Upper and lower unsteady pressure reconstruction - 5 instants - external TSM solver, elsA TSM solver and elsA unsteady solver - real and imaginary part

Increasing the number of harmonics up to 3 (7 instants) the results are identical.

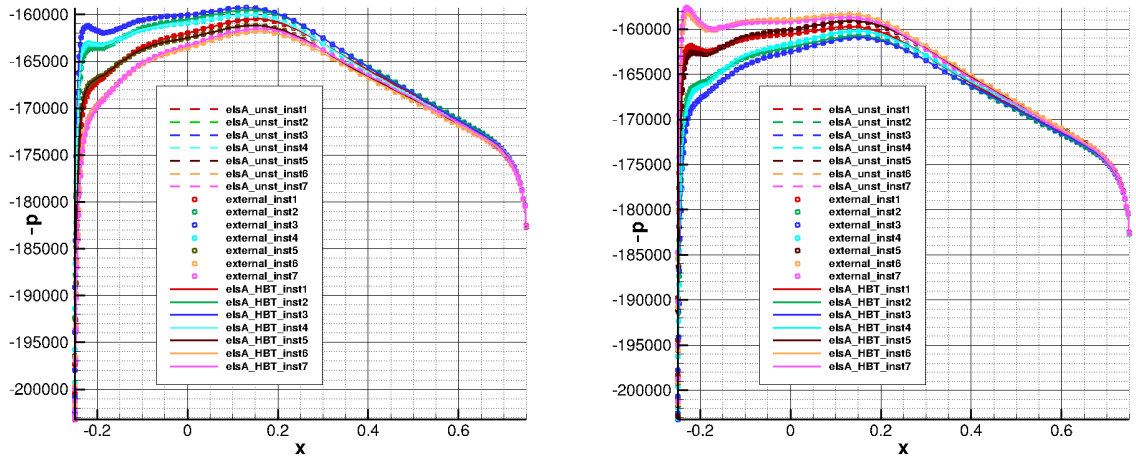


Figure 9.17: Upper and lower pressure at the 7 instants - external TSM solver, elsA TSM solver and elsA unsteady solver

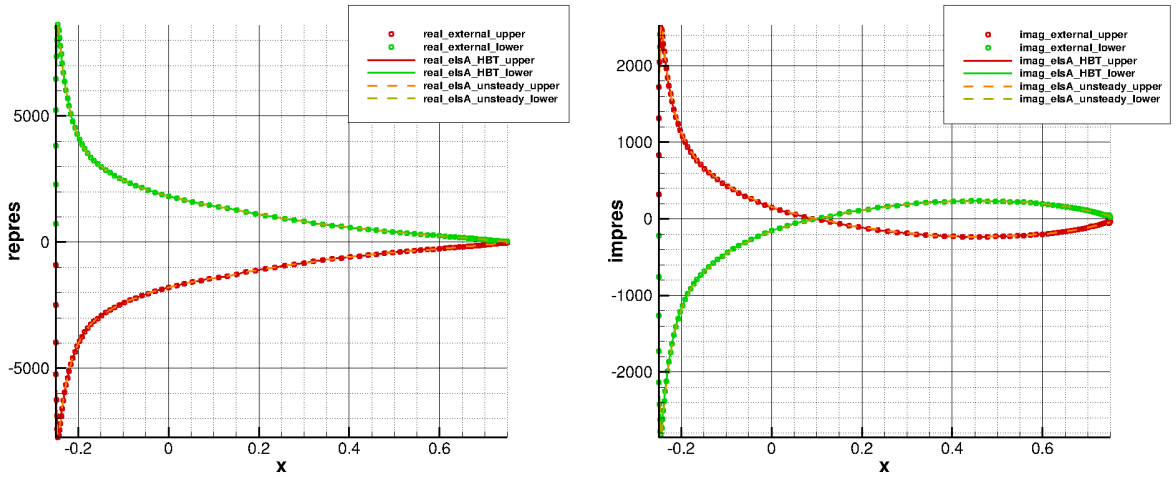


Figure 9.18: Upper and lower unsteady pressure reconstruction - 7 instants - external TSM solver, elsA TSM solver and elsA unsteady solver - real and imaginary part

This means that the linear case 29 can be approximated in a satisfying way with 2 harmonics.

9.1.4 Aerodynamic coefficients

In the following the plots of aerodynamic coefficients are represented. In fact in the elsA extracts it is possible to obtain the x-coefficient C_x , the z-coefficient C_z and the moment coefficient C_m, all expressed in body axis.

The coefficient of interest are the lift coefficient C_L and the pressure drag coefficient C_{D_p}, obtained by simply rotating C_x and C_z in wind axis.

$$C_{D_p} = C_x \cos(\alpha) + C_z \sin(\alpha) \tag{9.1}$$
$$C_L = -C_x \sin(\alpha) + C_z \cos(\alpha) \tag{9.2}$$

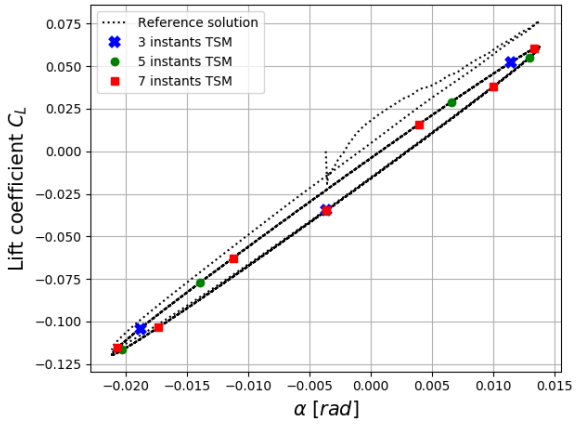


Figure 9.19: C_L-α - reference unsteady solution, with TSM solution for 3, 5, 7 instants

In the first plot the transient that an unsteady computation faces before reaching the periodic

steady state is evident, that is complete when the ellipses are completely superposed period by period.

In the following plots only the last period of oscillation is presented, when it's sure that the periodic steady regime has been reached. The unsteady reference simulation has been performed for 8 periods, each of them split in 1000 time steps.

The three coefficients are reconstructed via Fourier reconstruction, with 1, 2, 3 harmonics.

The Fourier reconstruction is performed applying the definition:

$$f(t_n) = a_0 + \sum_{k=1}^{N} [a_k \cos(\omega t_n) + b_k \sin(\omega t_n)] \tag{9.3}$$

with $f(t_n)$ the reconstructed function at the t_n instant, and $t_n = n\Delta t = \frac{n}{2N+1}$ with $n = 0, ..., 2N$.

It is sufficient to solve a linear system knowing the discrete aerodynamic coefficients (but of course it's valid for each quantity), to find the Fourier coefficients a_k and b_k. Once found them, the continuous function is reconstructed.

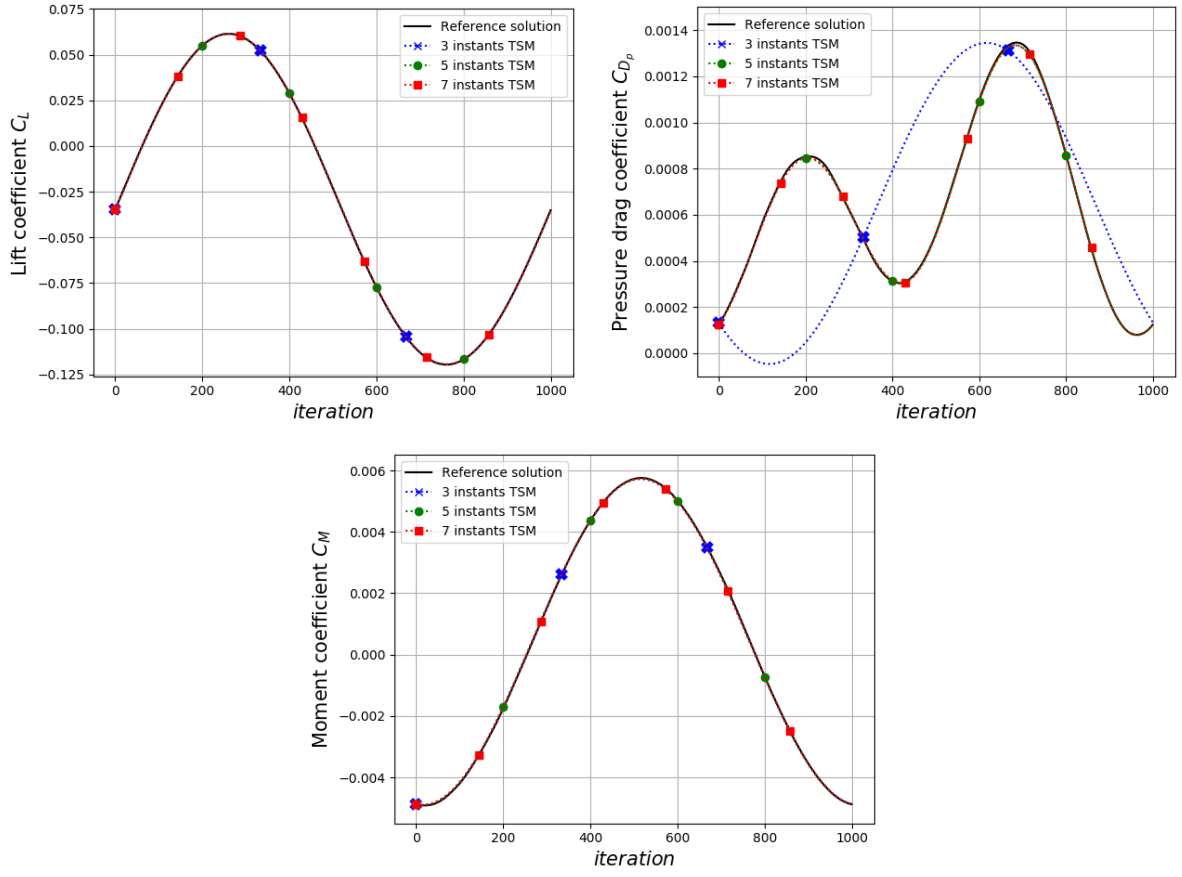


Figure 9.20: C_L, C_{D_p} and C_M vs. iteration (time) in the last period - reference unsteady solution, with TSM solution with Fourier reconsrtuction for 3, 5, 7 instants

The accordance with the reference data is well respected for all the computations with 3, 5 and 7 instants for the C_L, C_D and C_M coefficients. Of course, since the drag coefficient hasn't a sinusoidal behaviour, the reconstruction with 1 harmonic is not sufficient, and at least 2 harmonics are required.

9.1.5 Convergence

Being the linear low Mach number case, trying different strategies for CFL number together with the increase of the number of instants has been possible from the beginning, without introducing extra strategies.

Starting from the explicit version of the code, the trend of the norm of the TSM residuals is given below, using three different CFL numbers, CFL=1, CFL=5, CFL=10, above which the time steps become too large to assure convergence.

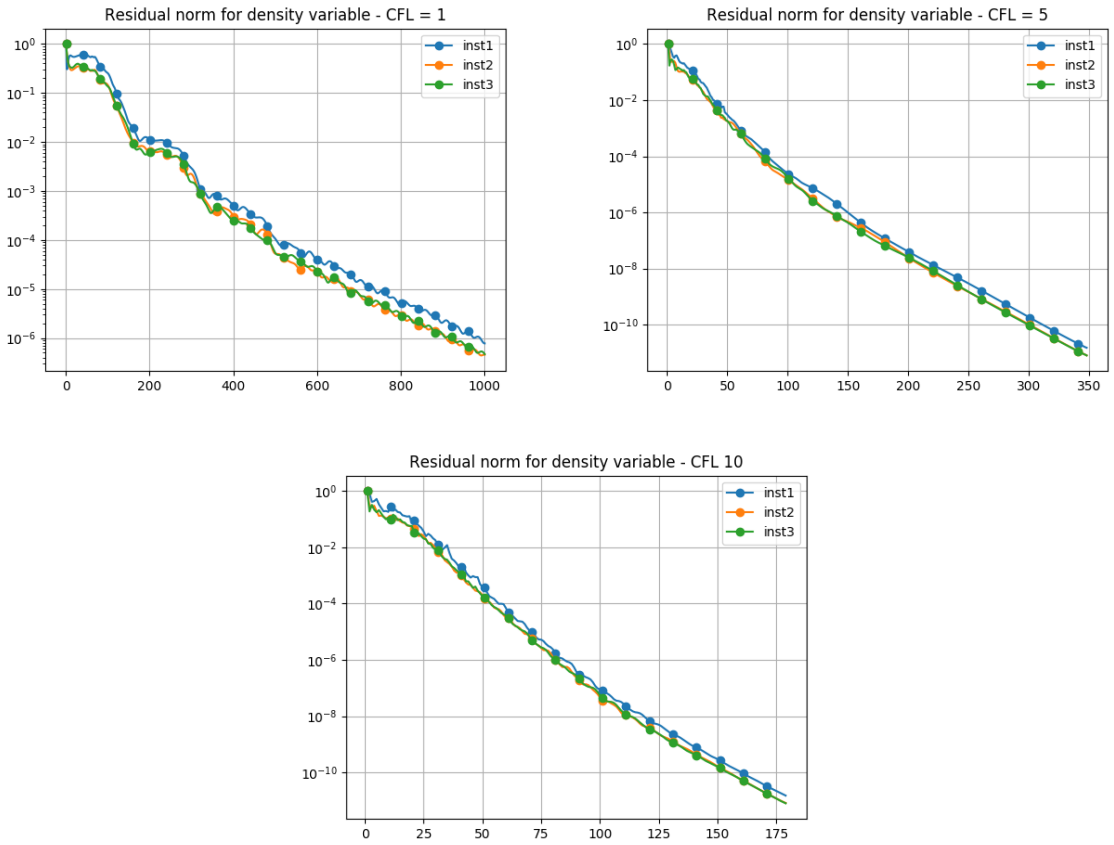


Figure 9.21: TSM residuals - CFL = 1, 5, 10 - 3 instants - explicit source term

By the way, time-spectral problems can still become difficult to converge when the maximum resolvable wave number becomes large, which occurs as the frequency of motion increases and/or the number of time instants used is raised.

In fact the maximum resolvable wave number is inversely proportional to the minimum resolvable wave length, deduced from the mesh size and the pseudo time step.

This comes from the Courant–Friedrichs–Lewy (CFL) condition, that is a necessary condition for convergence while partial differential equations like Euler or Navier-Stokes are solved numerically. The principle behind this condition is that if a wave is travelling across a discrete spatial grid, in order to compute its amplitude at discrete time steps of a certain duration, this duration must be less than the time taken by the wave to travel to neighbouring cells. As a consequence when the grid size is reduced, the the maximum allowed time step also decreases. In our case the time is not physical but numerical (the *pseudo* time).

Increasing the number of instants, the convergence is slowed down and severe constraints on the

CFL number show up, observing that the CFL needs to be decreased in order to converge higher harmonic computations.

A CFL=1 simulation is chosen to show the behaviour of the convergence, that becomes slower increasing the number of instants. This is the maximum CFL affordable for partially implicit method, indeed the simulations performed with 5 instants and 7 instants cannot reach convergence with higher CFLs.

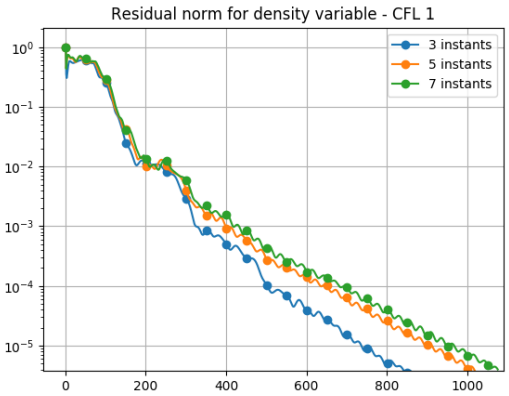


Figure 9.22: TSM residuals - CFL = 1 - 3, 5, 7 instants - explicit source term

For this purpose, the fully implicit formulation of the code was introduced. In the following the CFL=5 case, that is the highest CFL affordable to converge with 7 instants, is shown for the three cases with 3, 5 and 7 instants for both the partial and the full implicitation of the method.

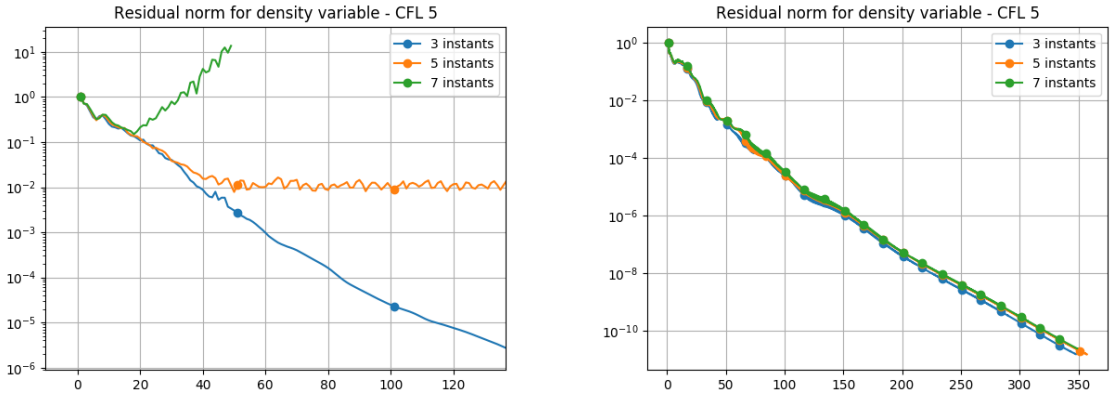


Figure 9.23: TSM residuals - CFL = 5 - 3, 5, 7 instants - explicit and implicit source term

It is shown how the convergence in the fully implicit method is the same for the three harmonics, so the CFL number and the rate of convergence are independent of the number of harmonics.

These results have been achieved thanks to the convergence of the second order Jacobian matrix in the block Jacobi subiteration, in which the norm of the difference of the increments of the conservative variables (the solution of the system) at two subsequent block Jacobi sub-iterations k and $k+1$ is driven to zero ($||\Delta W_{k+1} - \Delta W_k|| \rightarrow 0$ to allow convergence of the method with the implicit source term).

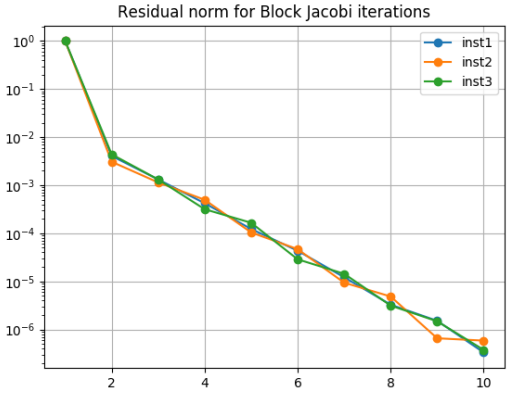


Figure 9.24: Convergence of Block Jacobi

The main goal of the implicitation needs to be remarked: it doesn't allow in every possible case to increase the CFL number for a problem. Instead it prevents the convergence to be slowed down due to the increasing of frequency or number of instants.

In fact, fig. 9.25 the behaviour for 3 CFL numbers (1,5,10) shows the convergence in both the explicit and the implicit formulations of the source term: the rate of convergence is not changed passing from one formulation to the other with the minimum number of instants.

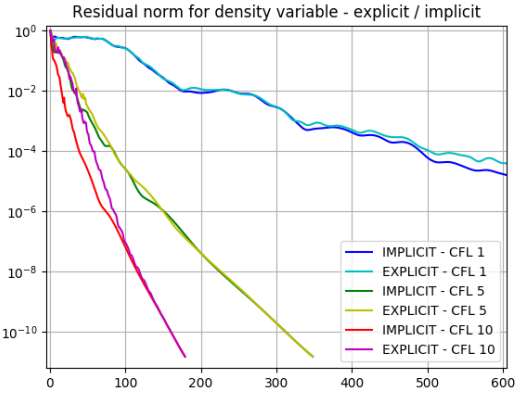


Figure 9.25: TSM residuals - CFL = 1, 5, 10 - 3 instants - explicit and implicit source term

As a conclusion, it is plotted in the following the trend of residuals, comparing the best strategy of the external solver, with CFL=5, with the TSM computation by elsA with CFL=1000. The gain in terms of iterations is well evident, of one eighth.

As already remarked, the CFL numbers which it's possible to use in the external solver are way lower than elsA's ones because elsA uses an approximation of the first order Jacobian matrix, more robust and better conditioned than the second order one. By the way the more accurate second order Jacobian matrix allows to reach much faster the convergence of the problem.

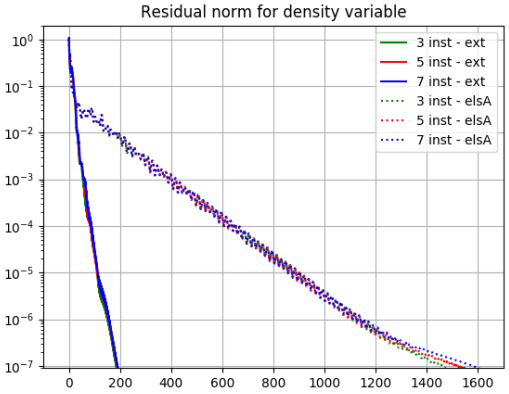


Figure 9.26: Best convergence strategy for 3, 5, 7 instants - external TSM solver vs. elsA TSM solver

9.2 High Mach number - low frequency

As already mentioned, the case 55 has been split into to sub-cases, with two different frequencies, $f = 10.8Hz$ (low frequency case) and $f = 34.4Hz$ (high frequency case). This is due to the convergence problems that the high frequency case has brought with it. This is the main reason why it's been decided to try the strategy of the *implicitation of the source term* to help the convergence of the problem without decreasing too much the CFL number.

9.2.1 Complete field around the airfoil - three instants

In this case the field is not well resolved utilizing only three instants, because of the introduction of non linearities caused by the increase of the Mach number; so the comparison aimed at the validation of the external solver is extended also to the TSM solver internal to elsA.

The only difference is in the visualization of the field, since elsA uses the ALE formulation and it is the profile to be rotated and not the boundary conditions.

Only the density field is shown below.

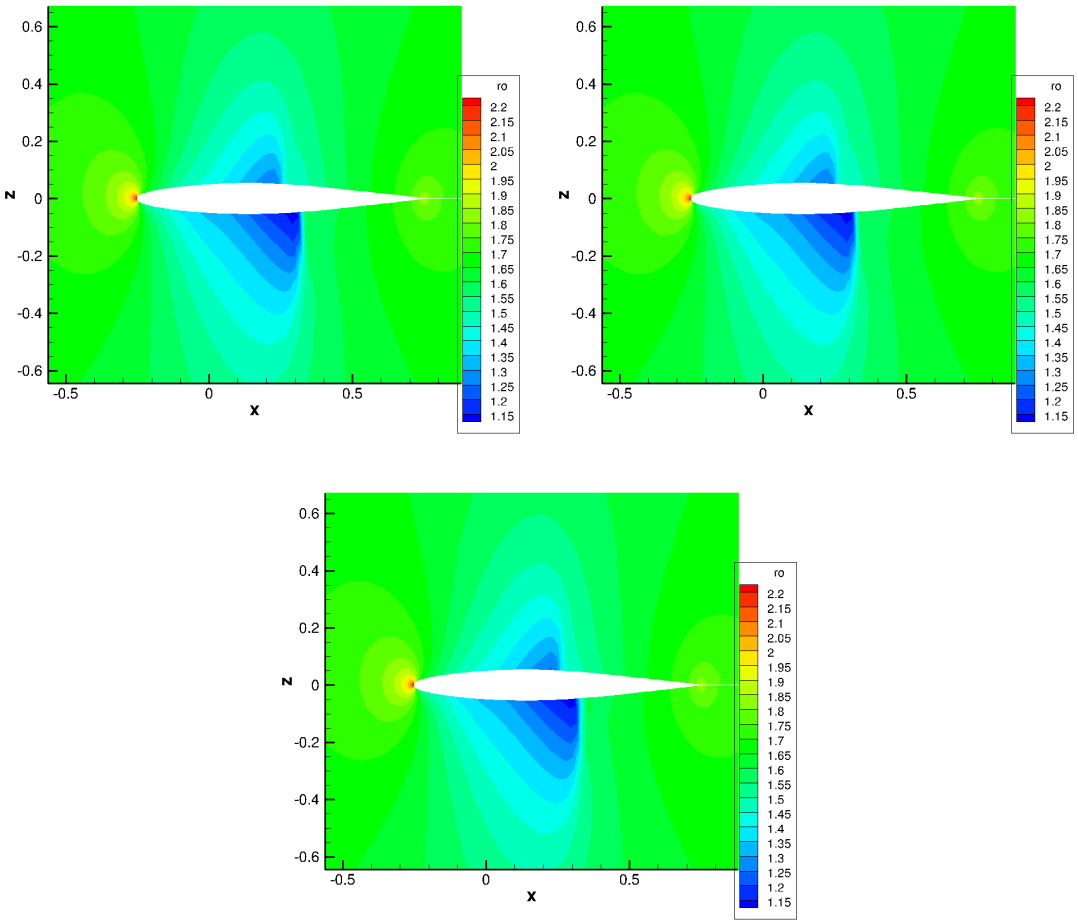


Figure 9.27: ρ field at instant 1 - comparison among the external TSM solver, the elsA TSM solver and the elsA unsteady solver

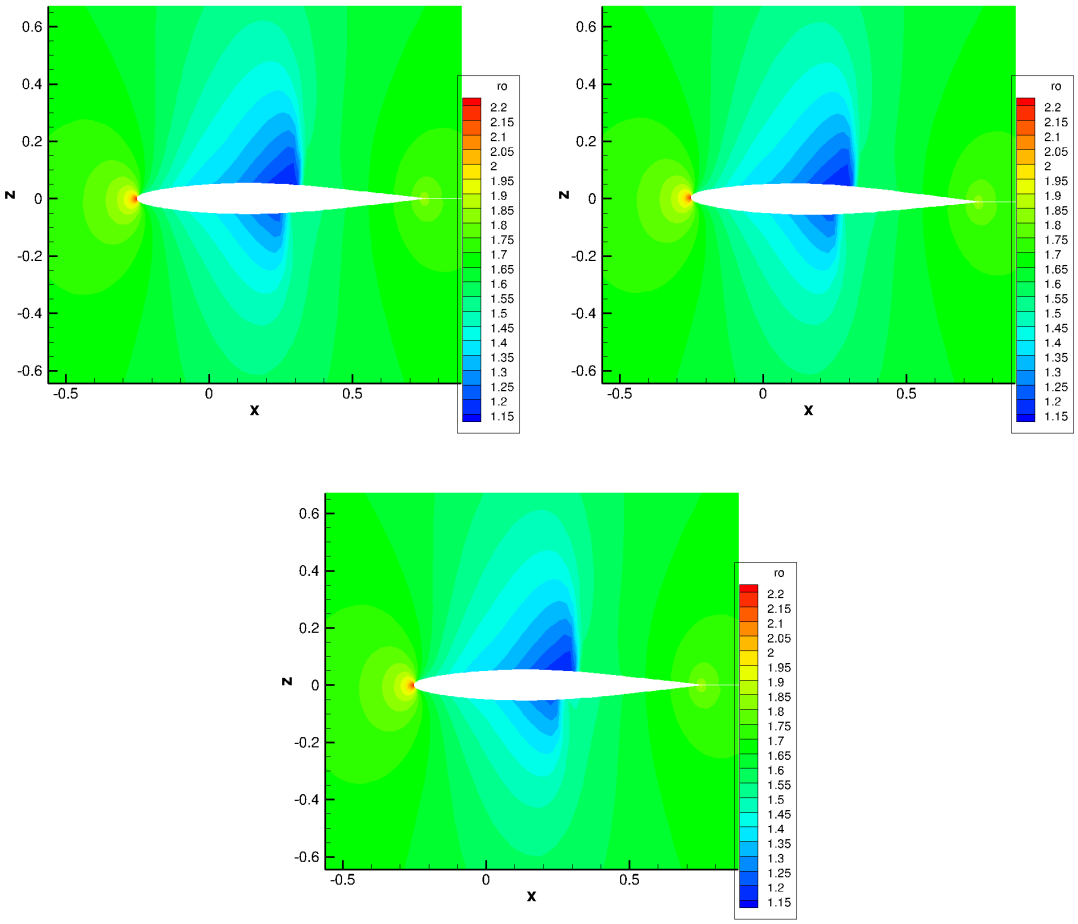


Figure 9.28: ρ field at instant 2 - comparison among the external TSM solver, the elsA TSM solver and the elsA unsteady solver

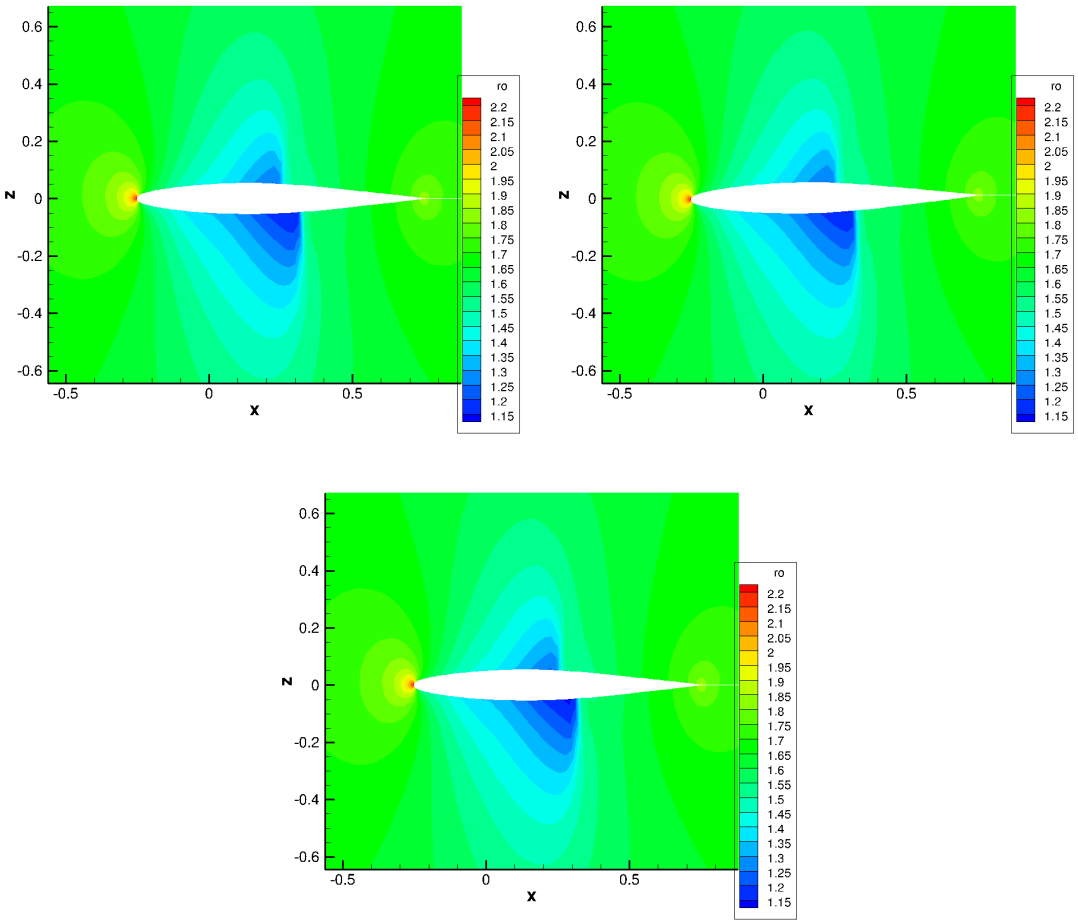


Figure 9.29: ρ field at instant 3 - comparison among the external TSM solver, the elsA TSM solver and the elsA unsteady solver

9.2.2 Wall pressure

In the following, upper and lower pressure at each instant, and the reconstruction of the unsteady pressure for the upper and lower part, are shown for 3, 5, 7 instants, comparing the unsteady reference simulation by elsA, the TSM reference simulation by elsA and the results by the TSM external solver.

The validation of the TSM external code is given by the perfect superposition of the results with the results by elsA's TSM solver.

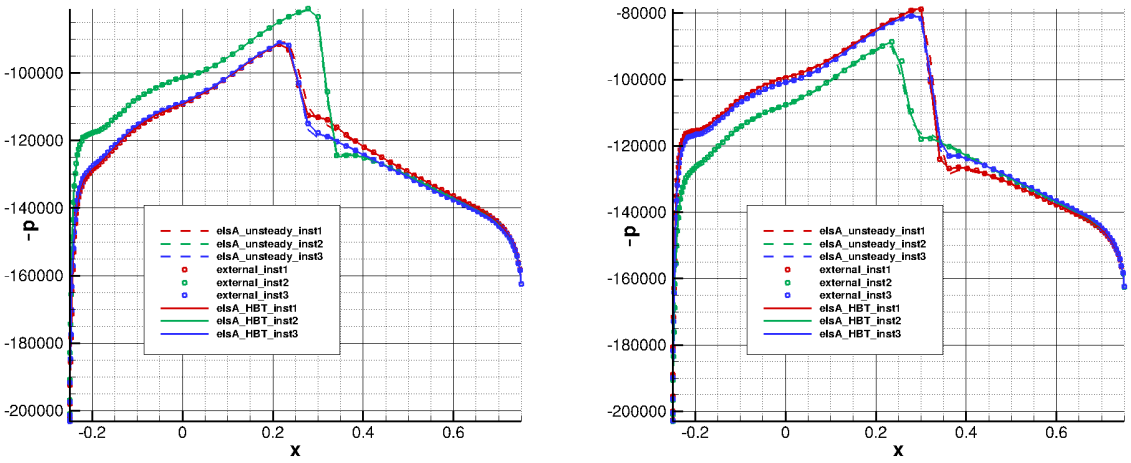


Figure 9.30: Upper and lower pressure at the 3 instants - external TSM solver, elsA TSM solver and elsA unsteady solver

Increasing the Mach number, some differences arise in the area of the shock. These are more evident and better highlighted in the plot of the unsteady pressure around the airfoil.

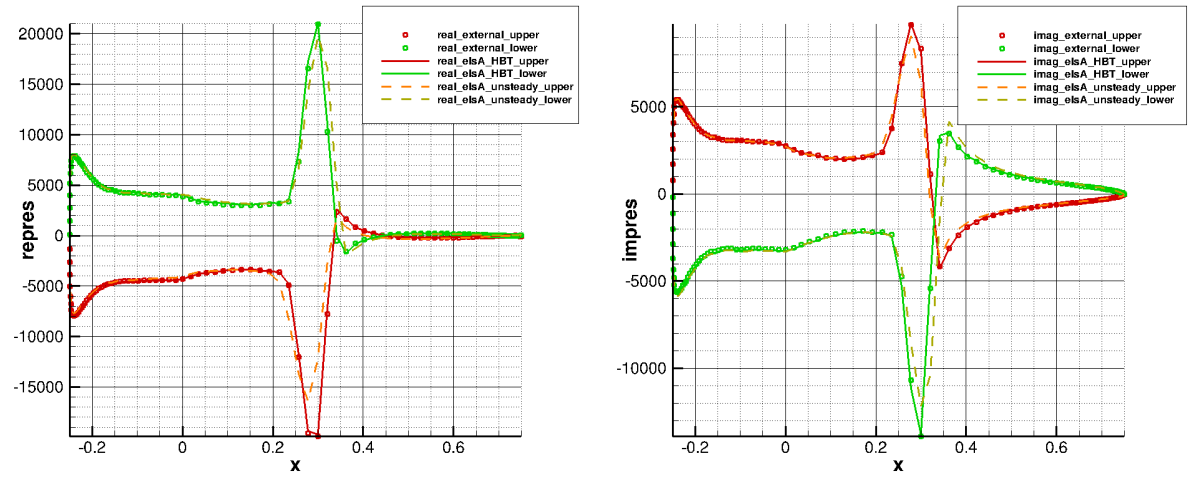


Figure 9.31: Upper and lower unsteady pressure reconstruction - 3 instants - external TSM solver, elsA TSM solver and elsA unsteady solver - real and imaginary part

Increasing the number of instants up to 5, the unsteady pressure is better resolved also in the area of the shock, as shown in fig. 9.33.

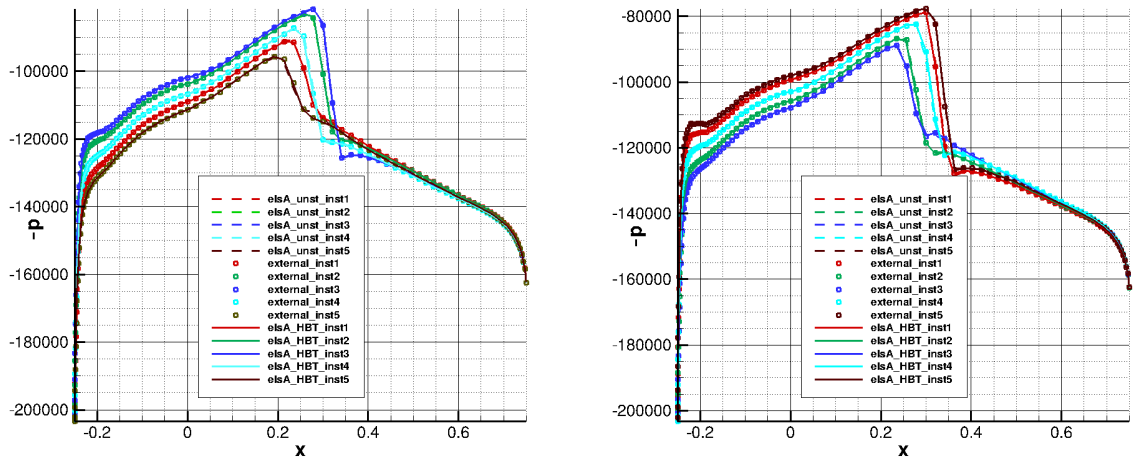


Figure 9.32: Upper and lower pressure at the 5 instants - external TSM solver, elsA TSM solver and elsA unsteady solver

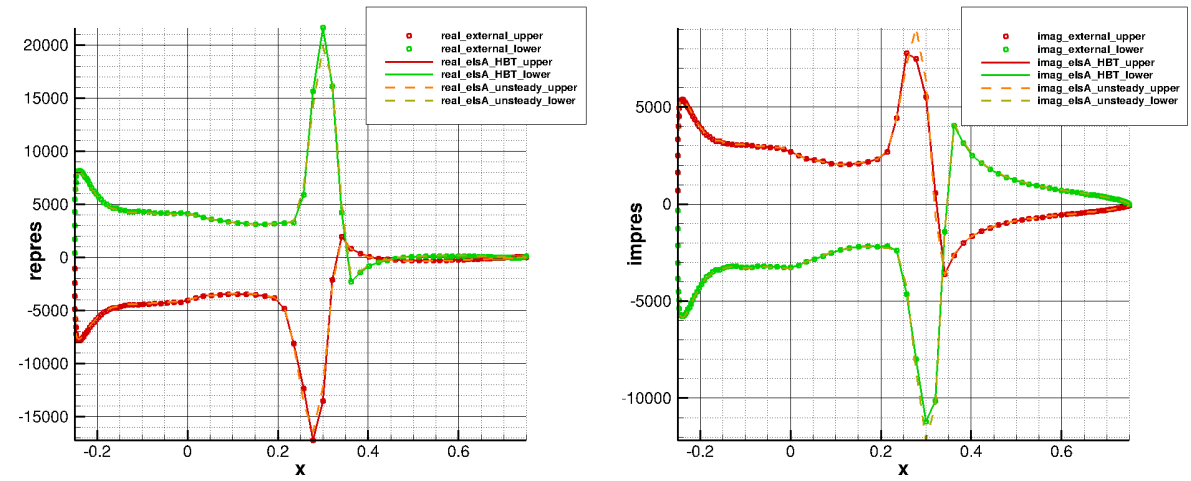


Figure 9.33: Upper and lower unsteady pressure reconstruction - 5 instants - external TSM solver, elsA TSM solver and elsA unsteady solver - real and imaginary part

Increasing up to 7 instants the results are almost superposed to the unsteady ones.

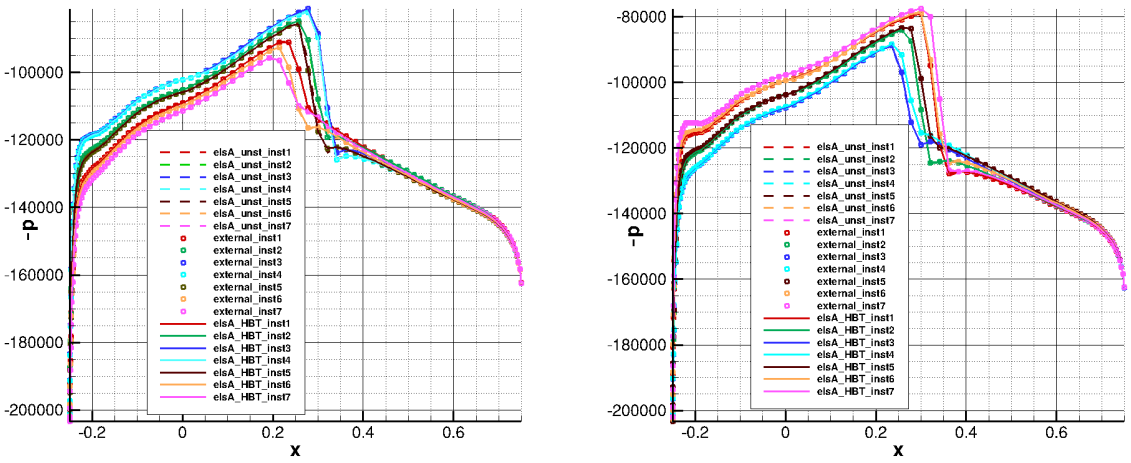


Figure 9.34: Upper and lower pressure at the 7 instants - external TSM solver, elsA TSM solver and elsA unsteady solver

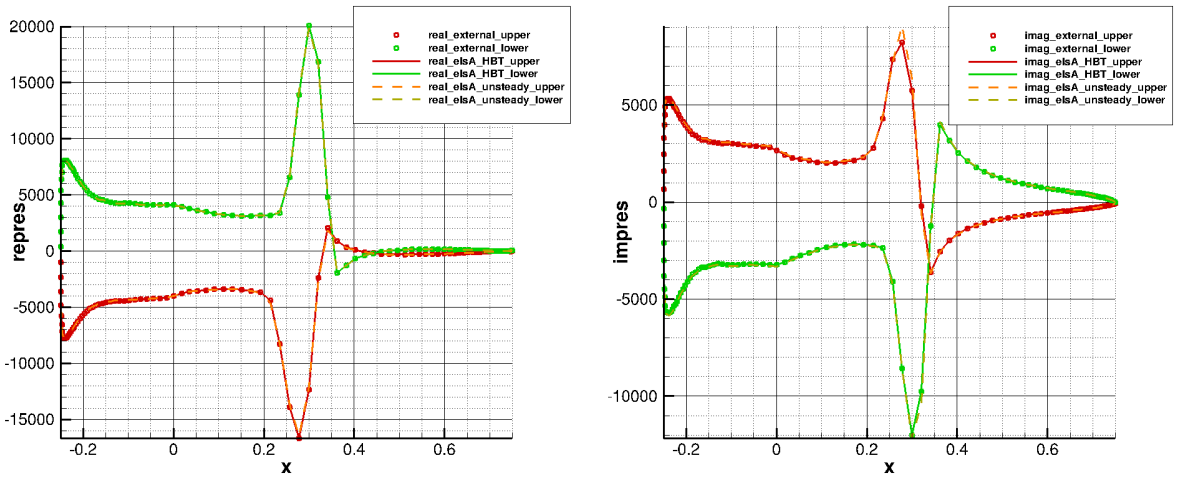


Figure 9.35: Upper and lower unsteady pressure reconstruction - 7 instants - external TSM solver, elsA TSM solver and elsA unsteady solver - real and imaginary part

9.2.3 Aerodynamic coefficients

The C_L coefficient is illustrated in fig. 9.50 with respect to the angle of attack. The C_L coefficient seems well resolved even with 3 instants, comparing it with the unsteady reference solution.

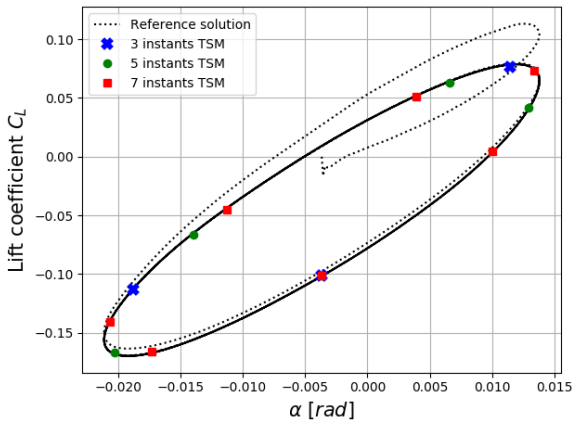


Figure 9.36: C_L-α - reference unsteady solution, with TSM solution for 3, 5, 7 instants

Presenting the reconstruction of the three aerodynamic coefficients C_L, C_{D_p} and C_M, we can notice again that C_L and C_M are sinusoidal signals, well resolved also with 3 instants.

Instead, the non sinusoidal pressure coefficient cannot be reconstructed with 1 harmonic, and moreover the results obtained by the TSM simulation with 1 harmonic are not even superposed to the reference.

This means that also in this case 2 harmonics are the minimum number to approximate efficiently the problem.

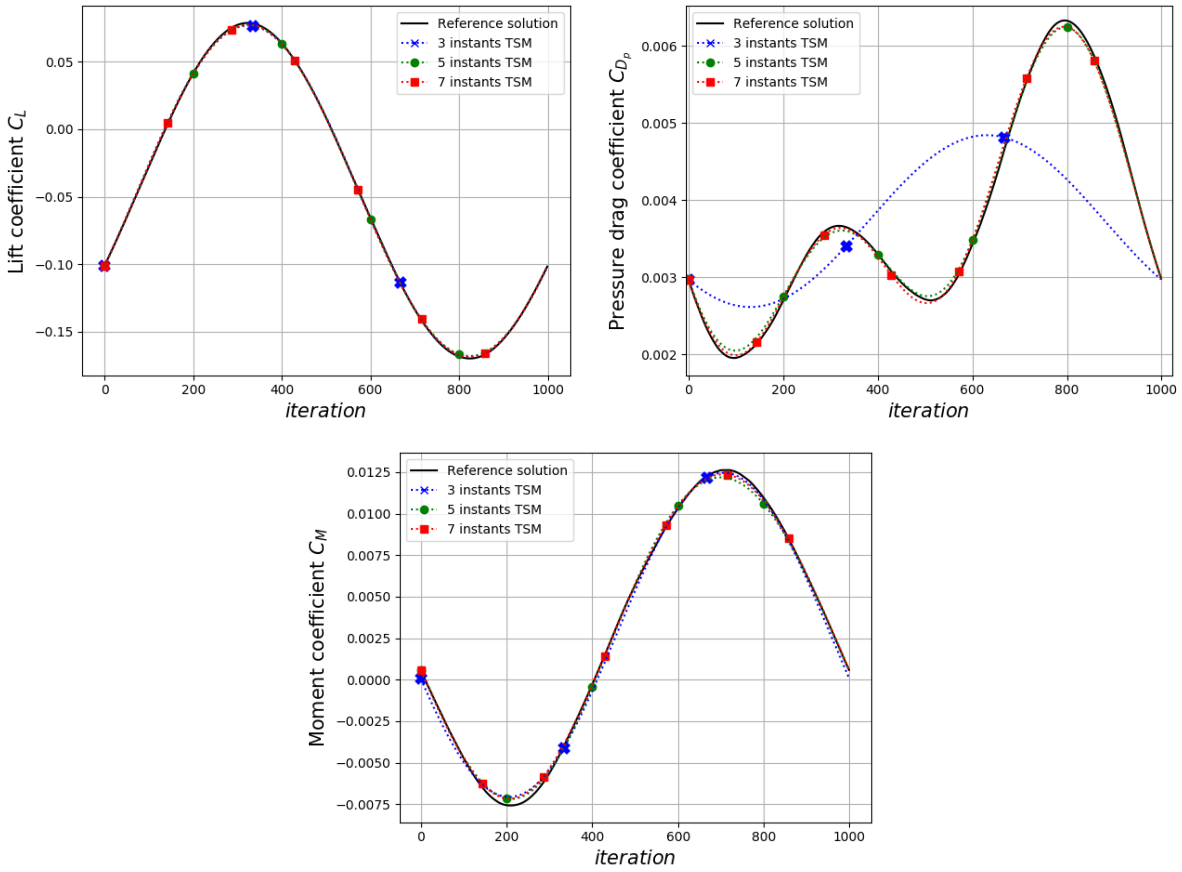


Figure 9.37: C_L, C_{D_p} and C_M vs. iteration (time) in the last period - reference unsteady solution, with TSM solution with Fourier reconstruction for 3, 5, 7 instants

9.2.4 Convergence

The best strategy for convergence in the case of high Mach number with low frequency for 3 instants is shown below, that is also in this case with a CFL=10, but this time helped by the first iterations performed exploiting the first order Jacobian matrix.

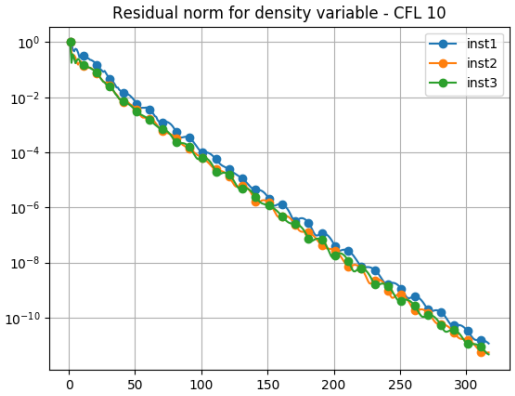


Figure 9.38: TSM residuals - CFL = 10 - 3 instants - explicit source term

Also in this case the benefits given by the implicitation of the source term, in terms of relaxation of the CFL constraints, have been useful to keep constant the CFL number while increasing the number of instants, and keeping the same convergence behaviour.

For a computation with CFL=5, that is the maximum affordable CFL number for this case, the simulations with 5 and 7 instants diverge with the explicit source term formulation. The full implicitation relaxes the problem and allows convergence to all three simulations.

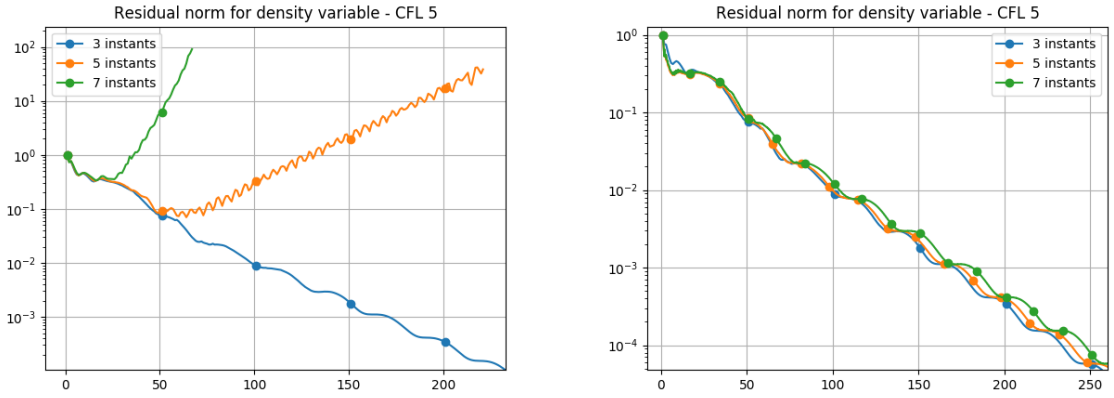


Figure 9.39: TSM residuals - CFL = 5 - 3, 5, 7 instants - explicit and implicit source term

The rate of convergence of the best CFL strategy for the external computation, CFL=5, compared to elsA's TSM simulation with a CFL=1000, is illustrated below.

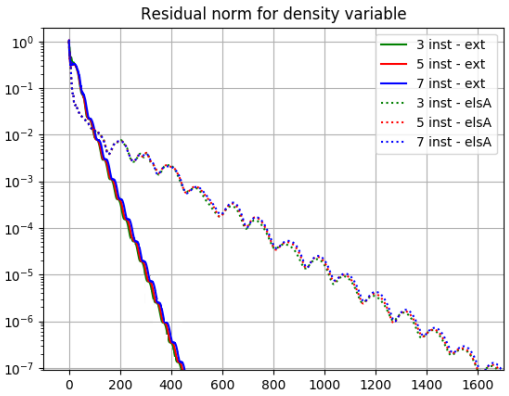


Figure 9.40: Best convergence strategy for 3, 5, 7 instants - external TSM solver vs. elsA TSM solver

9.3 High Mach number - high frequency

This is the final case, which suffers the strictest CFL constraints both in the external TSM solver and in TSM performed by elsA. In fact at the end of the section a short demonstration of the same behaviour due to a lack of robustness of the Block Jacobi full implicitation, is given for both the solvers.

In future implementations of the code, only this case will be taken into account, being the hardest to converge.

9.3.1 Complete field around the airfoil - three instants

For the sake of shortness, only the final density field for a 3 instants simulation is presented. Also in this case, since the phenomenon is not linear and cannot be approximated properly with one harmonic, the validation of the results must be compared also with the TSM by elsA.

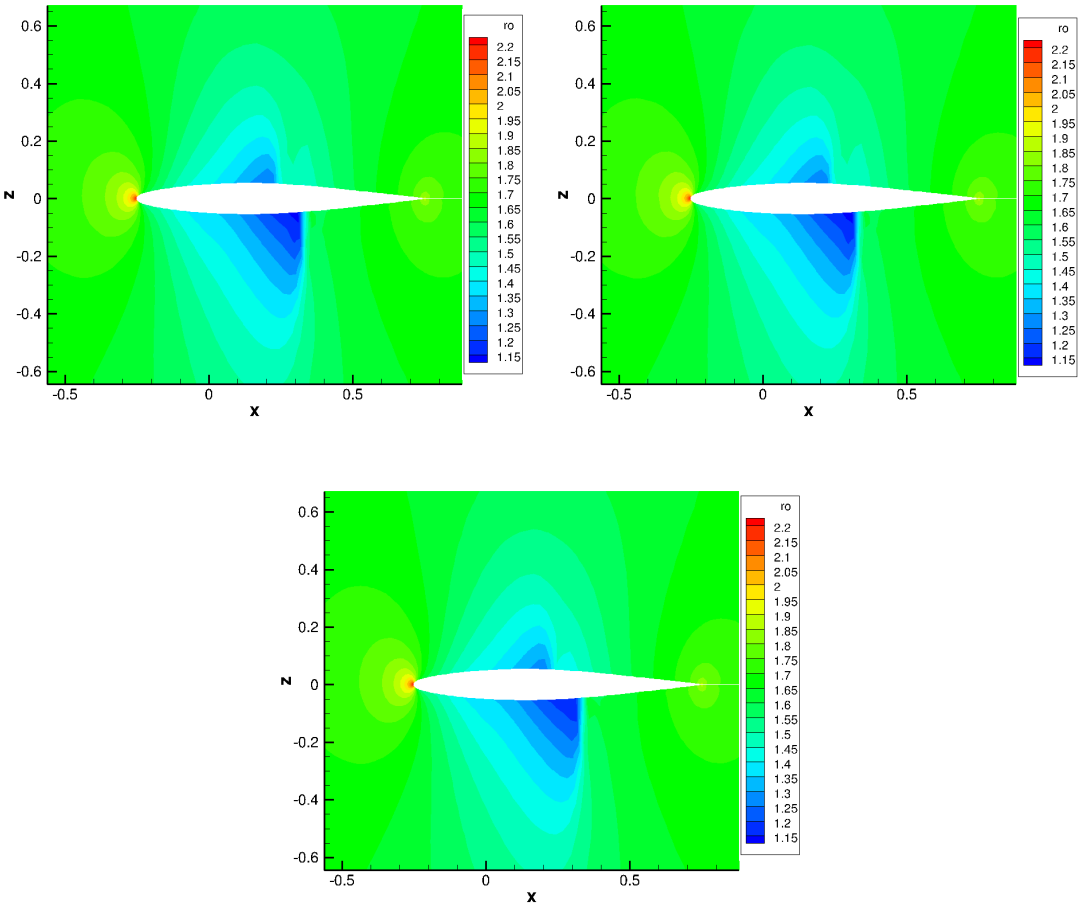


Figure 9.41: ρ field at instant 1 - comparison among the external TSM solver, the elsA TSM solver and the elsA unsteady solver

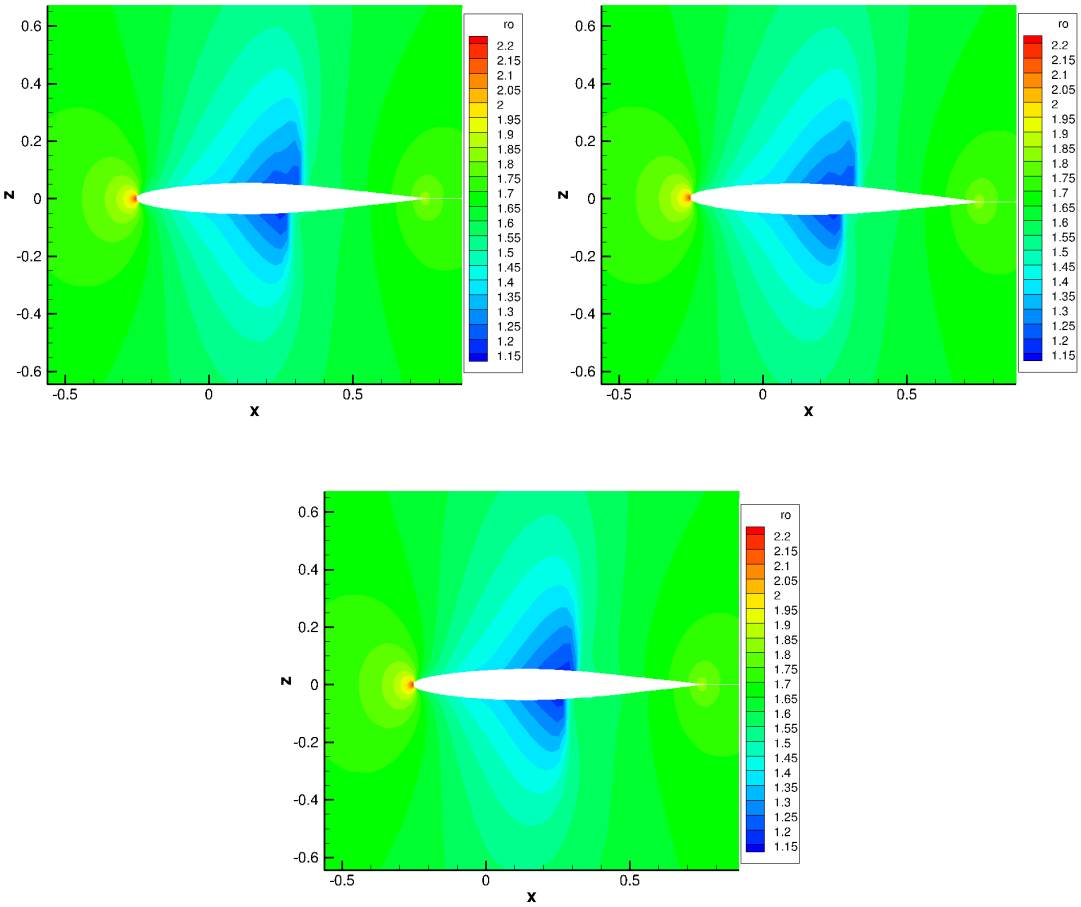


Figure 9.42: ρ field at instant 2 - comparison among the external TSM solver, the elsA TSM solver and the elsA unsteady solver

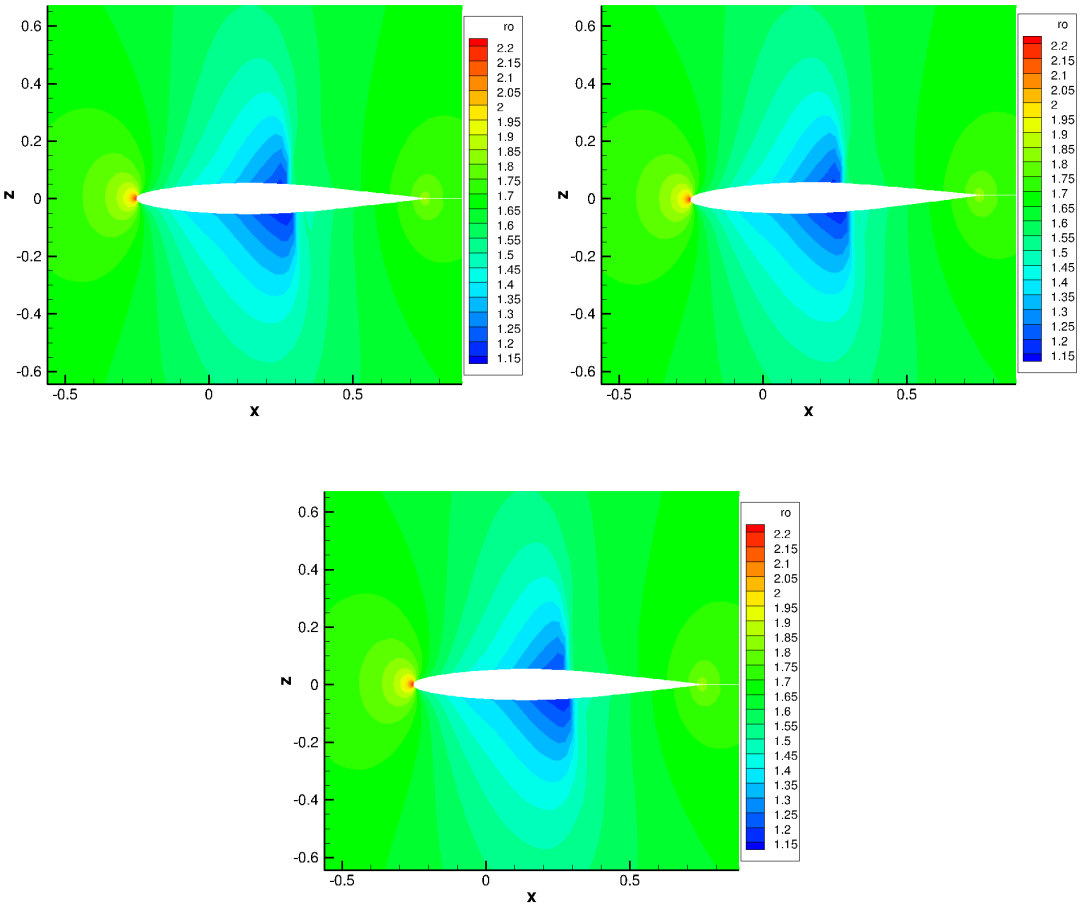


Figure 9.43: ρ field at instant 3 - comparison among the external TSM solver, the elsA TSM solver and the elsA unsteady solver

This time the difference between the two TSM computations (performed by the external solver and by elsA, completely identical) and the unsteady simulation are more consistent. In fact the non linearities due to the transonic condition, are increased by the high fundamental frequency of the motion. To resolve the field in an accurate way, a higher number of harmonics will be needed.

9.3.2 Wall pressure

In the following, upper and lower pressure at each instant, and the reconstruction of the unsteady pressure for the upper and lower part, are shown for TSM computations with 3, 5, 7 instants, comparing the unsteady reference simulation by elsA, the TSM reference simulation by elsA and the results by the TSM external solver.

The validation of the TSM external code is given by the perfect superposition of the results with the results by elsA's TSM solver.

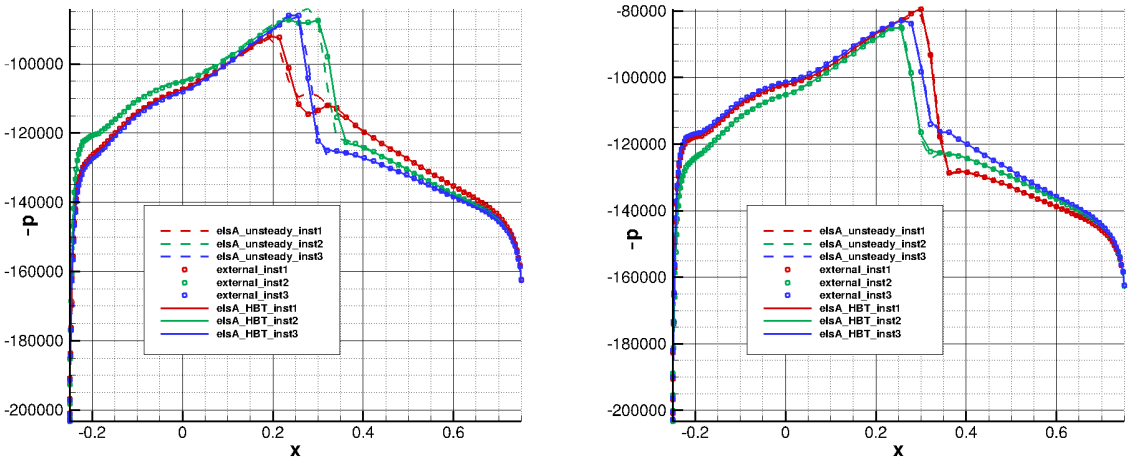


Figure 9.44: Upper and lower pressure at the 3 instants - external TSM solver, elsA TSM solver and elsA unsteady solver

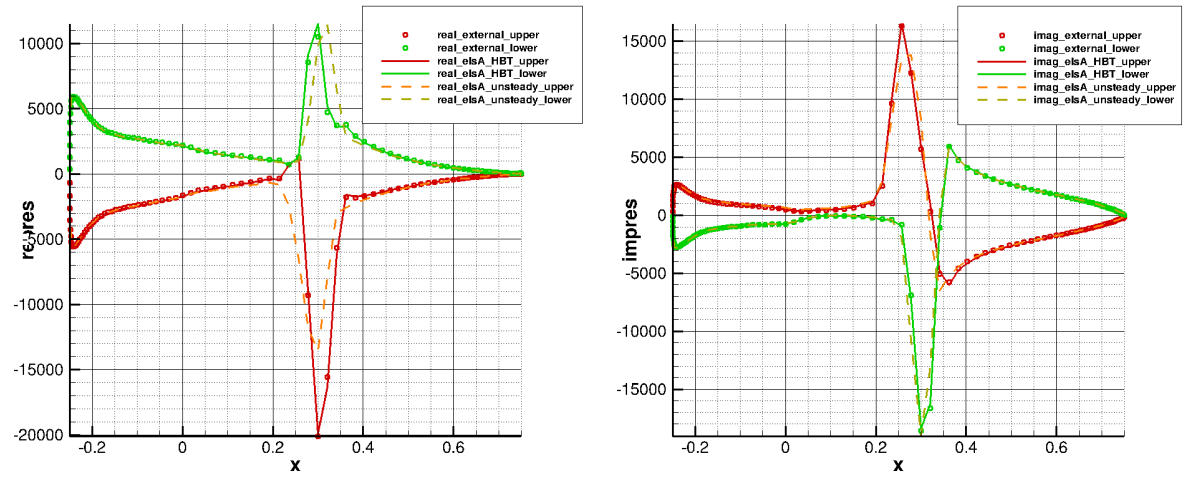


Figure 9.45: Upper and lower unsteady pressure reconstruction - 3 instants - external TSM solver, elsA TSM solver and elsA unsteady solver - real and imaginary part

The field seems well solved everywhere but in the area of the shock. This means that only one harmonic is not sufficient to solve completely the problem. The simulation has been re-performed but this time with 5 instants.

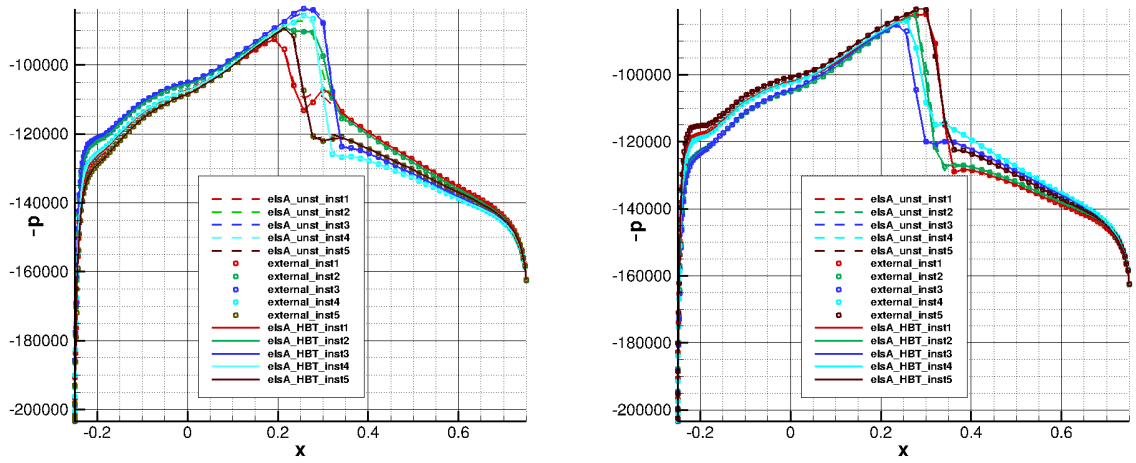


Figure 9.46: Upper and lower pressure at the 5 instants - external TSM solver, elsA TSM solver and elsA unsteady solvert

Performing again the Fourier analysis the improvement obtained increasing the number of instants is evident.

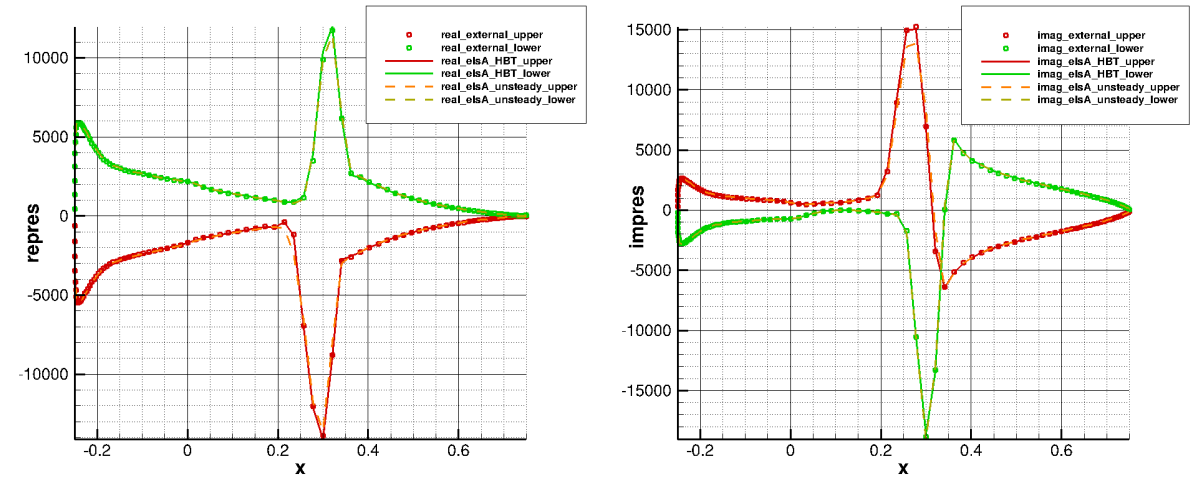


Figure 9.47: Upper and lower unsteady pressure reconstruction - 5 instants - external TSM solver, elsA TSM solver and elsA unsteady solver - real and imaginary part

Using seven instants the results are almost completely superposed to the unsteady ones.

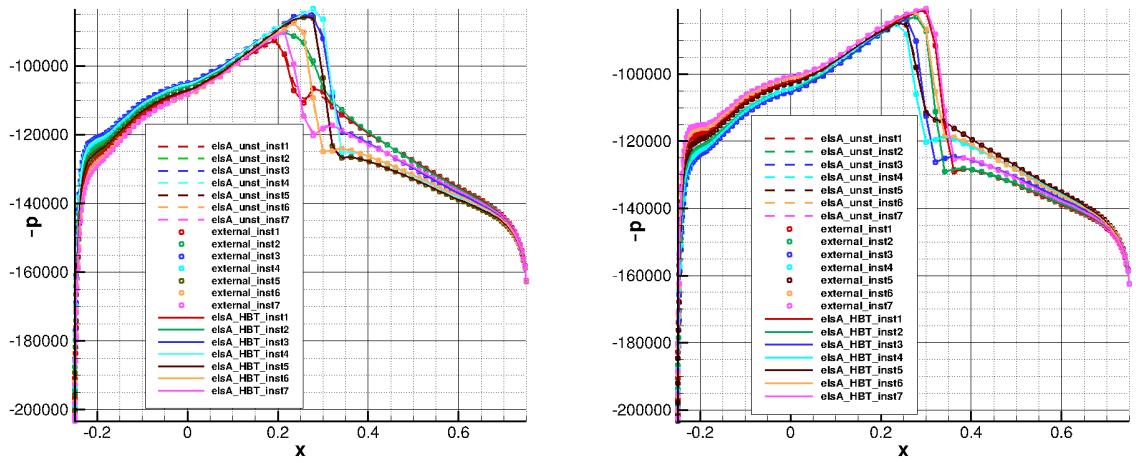


Figure 9.48: Upper and lower pressure at the 7 instants - external TSM solver, elsA TSM solver and elsA unsteady solver

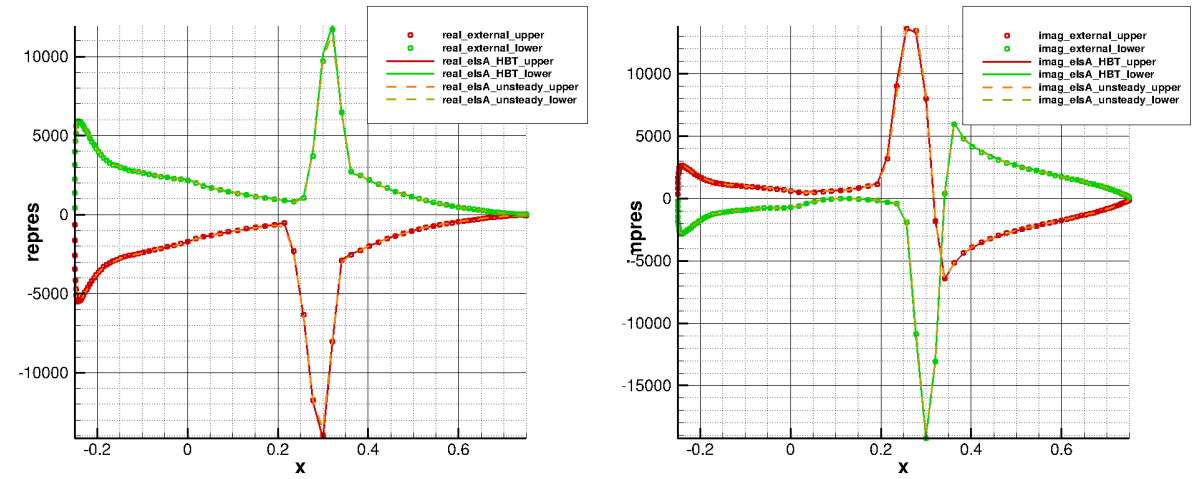


Figure 9.49: Upper and lower unsteady pressure reconstruction - 7 instants - external TSM solver, elsA TSM solver and elsA unsteady solver - real and imaginary part

From this kind of analysis it is noticeable that 3 harmonics yield perfect superposition but 2 harmonics gives also very good results in approximating in a satisfying way the problem.

9.3.3 Aerodynamic coefficients

As experienced in the previous two cases, the C_L obtained by TSM is always (for 3, 5, 7 instants) superposed to the reference computation, being sinusoidal, as also the C_M.

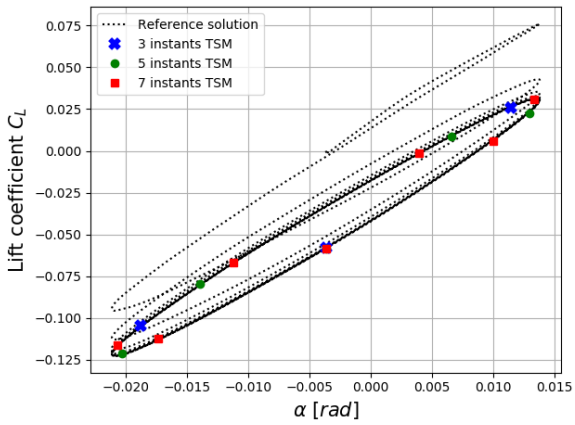


Figure 9.50: C_L-α - reference unsteady solution, with TSM solution for 3, 5, 7 instants

On the other side there is again the pressure drag coefficient that is not sinusoidal. First of all the 3 instants computation cannot output coefficients superposed to the reference ones. And again, a sinusoidal, one harmonic reconstruction is not sufficient for the C_{D_p}. As already noticed from the unsteady pressure analysis, the final case is not linear, and the minimum number of harmonics required is 3.

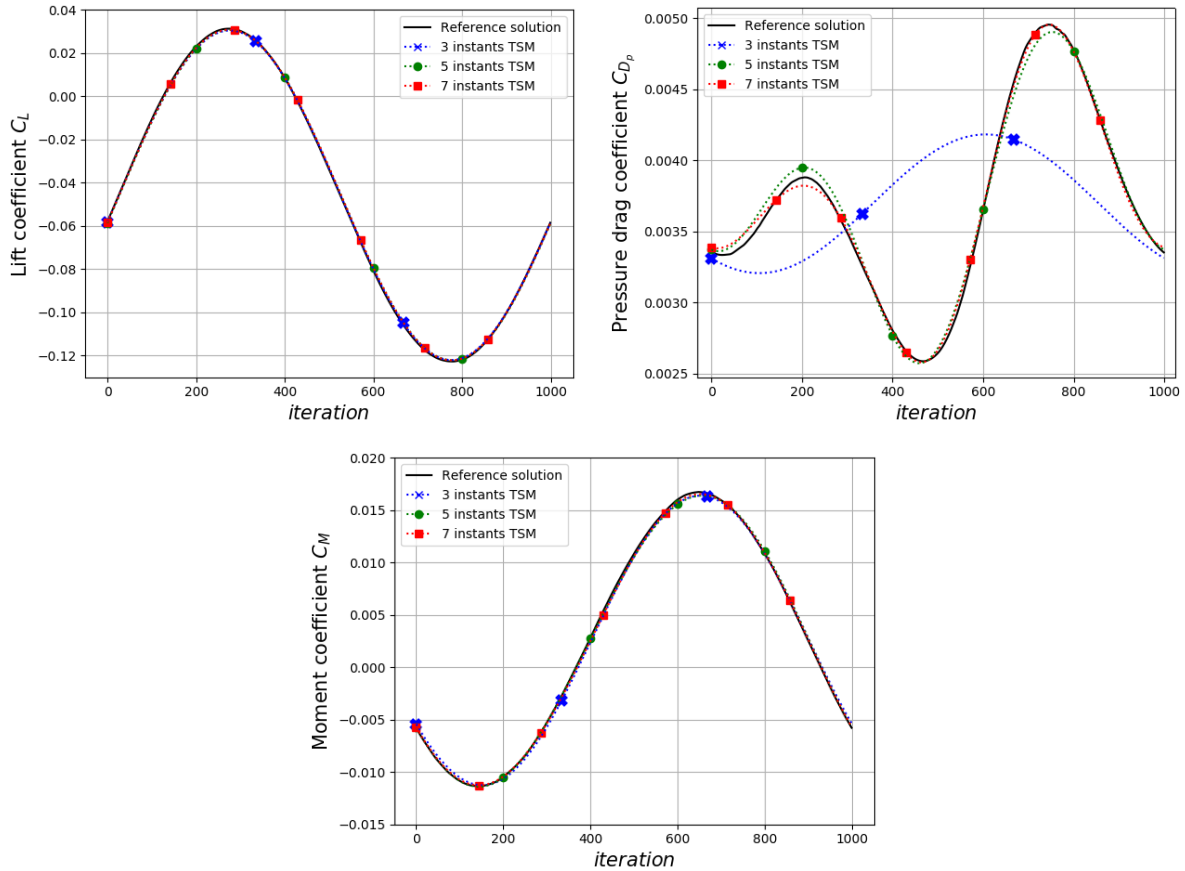


Figure 9.51: C_L, C_{D_p} and C_M vs. iteration (time) in the last period - reference unsteady solution, with TSM solution with Fourier reconstuction for 3, 5, 7 instants

9.3.4 Convergence

Even though the right solution is achieved in any case simply decreasing the CFL number, the convergence of the method seems to be severely affected by the increase of the frequency. Indeed choosing a CFL=1 a strong stagnation is experienced after a decrease of two orders of magnitude of the residuals (nevertheless the right solution is already reached in this case). In order to have a constant rate of convergence, and to make the simulation more reliable, a CFL=0.5 is needed.

Hence a similar effect to the increase of number of instants is experienced by increasing the pitching frequency of the airfoil, since they have the same effect on the pseudo time step $\Delta\tau$, i.e. decreasing $\Delta\tau$ while increasing f and N.

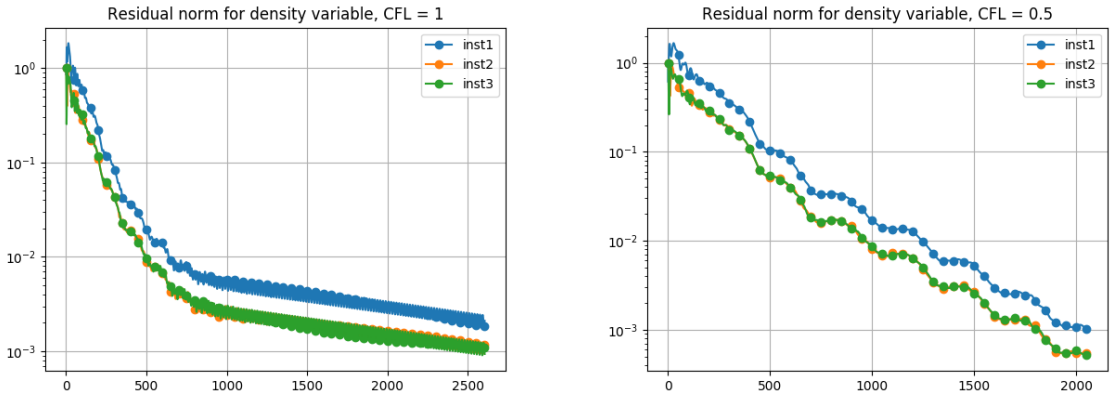


Figure 9.52: TSM residuals - CFL = 1, 0.5 - 3 instants - explicit source term

Two alternative ways have been tested in order to help the convergence: the direct LU solver has been replaced by the approximate direct solver ILU in the first case, and a complete simulation exploiting the Jacobian of the first order in the second case.

None of the alternative methods seem to help the convergence.

In future works this constraint can be relaxed by the use of an iterative solver like GMRES, which doesn't solve exactly the system at each iteration. A loss in the rate of convergence is expected, but on the other hand the solver will become more robust and less inclined to convergence problems.

By the way a maximum CFL=0.5 seems completely inadequate for practical issues, so a Block Jacobi fully implicit implementation seems necessary.

Applying a CFL=1 to the partially implicit formulation, the same behaviour experienced in the two previous cases is obtained (fig. 9.53): for higher number of harmonics the partially implicit solver diverges, while the fully implicit formulation ensures a similar convergence for all the three cases, that is moreover improved with respect to the explicit formulation with 3 instants, preventing the residual from stagnating.

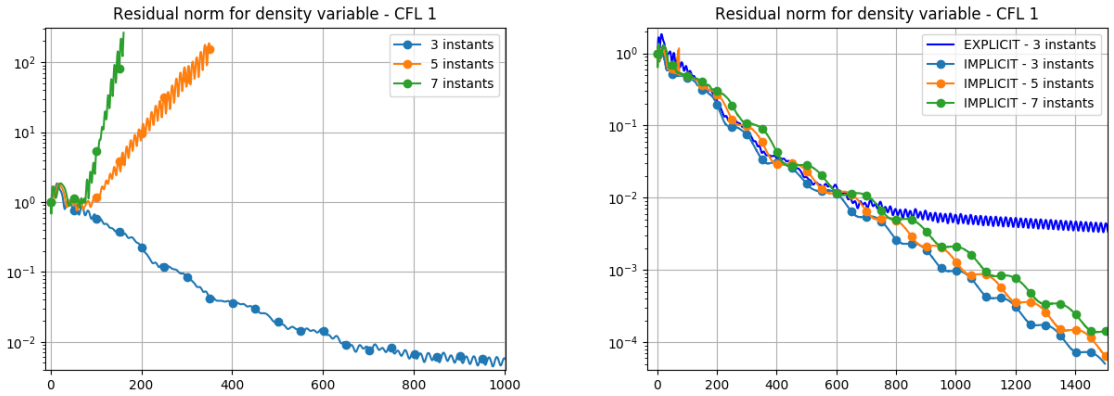


Figure 9.53: TSM residuals - CFL = 1 - 3, 5, 7 instants - explicit and implicit source term

After the analysis on the high frequency and high number of harmonics case, the necessity of the implementation of the implicit source term became relevant in order to be able to go further in the following studies using this type of method.

Finally the improvements obtained thanks to the implicitation of the source term are shown, allowing to solve the hardest test case with a CFL=2 for 3, 5, 7 instants, with a decrease of 7 orders of magnitude in about 1250 iterations, while elsA in 2500 with a CFL=10. This is an important result, that shows the effectiveness of the second order Jacobian matrix, despite the less robust direct solver is still used, penalized by a very low maximum affordable CFL number.

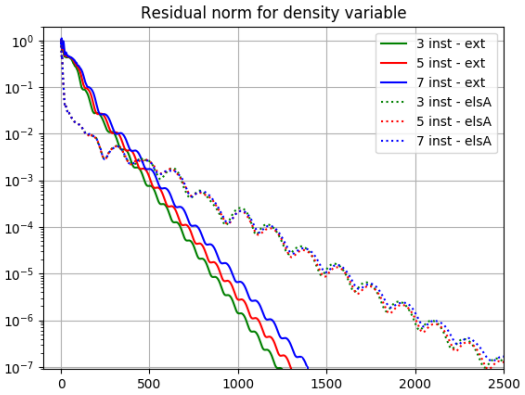


Figure 9.54: Best convergence strategy for 3, 5, 7 instants - external TSM solver vs. elsA TSM solver

9.3.5 Robustness problems: external code and elsA

As shown in the results section, the fully implicit TSM implementation relaxes the CFL constraints given by the partially implicit TSM, but still doesn't allow to completely make the problem independent of the frequency and the number of harmonics.

elsA implements the same *Block Jacobi* full implicitation, and it is hence expected the same robustness problems.

Given the higher stiffness of the exact Jacobian matrix used in the external solver, as several times has been repeated, comparing the TSM solver by elsA and by us is impossible in terms of same CFL number. Indeed it is possible only to compare the "best" CFL strategies for the two

solvers.

In fig. 9.55, the same divergence problems are faced by the external solver for a CFL=5, and by the elsA solver for a CFL=50 while increasing the number of instants.

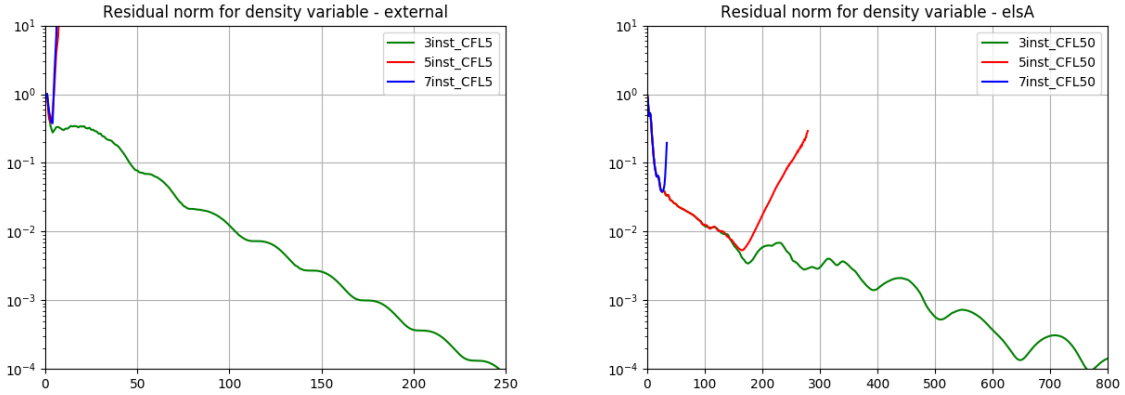


Figure 9.55: Divergence - external TSM solver and elsA TSM solver

Decreasing the CFL number to 2 for the external solver and to 10 for elsA (fig. 9.56), convergence is ensured, and the convergence rate is approximately the same for the simulations performed with different instants.

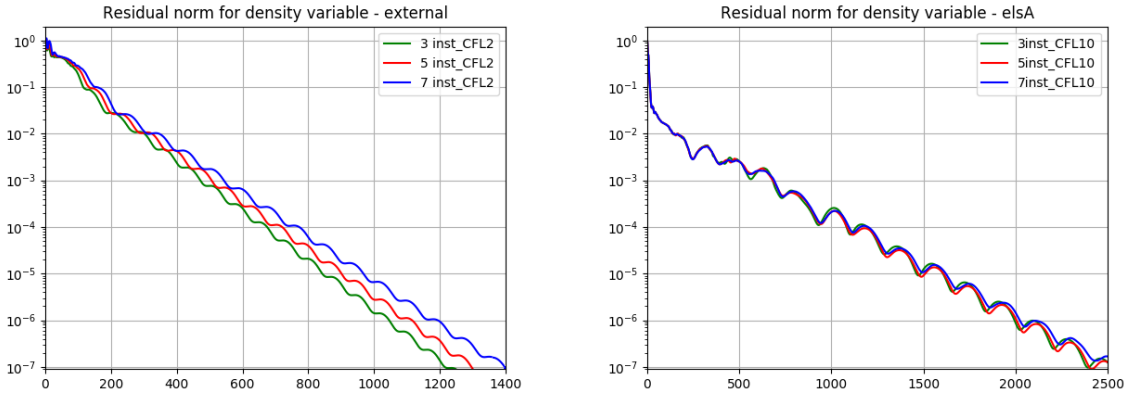


Figure 9.56: Convergence decreasing the CFL number - external TSM solver and elsA TSM solver

Chapter 10

Shape optimization

In this last chapter, a first implementation of a steady shape optimizer is presented. Aerodynamic shape optimization consists in looking for the shape, among a set of parametrized aerodynamic shapes, that significantly improves an objective function while satisfying a set of constraints.

In our case the objective function is the airfoil drag (pressure drag) under the constraints related to lift: the aim of the process is the pressure drag coefficient reduction, such that the lift coefficient doesn't drop below a certain threshold. This threshold is set to the lift coefficient of the nominal airfoil.

Before starting the discussion about the optimization process, it is necessary to define p_k as the vector of design parameters at a certain step of the optimization algorithm k, and J as the objective scalar functions, i.e. the functions that decide the optimization direction, or the constraint function, that is the function which puts limitations in the optimization process. For our purposes the objective function is drag - to minimize, and the constraint function is lift - to keep above a prescribed value. Besides X denotes the coordinates of the grid.

10.1 Theoretical aspects

10.1.1 Aerodynamic shape optimization using numerical simulation

The objective is to solve the linearized system of discretized equations around the steady configuration.

The steady problem solution satisfies, for the Navier-Stokes equations:

$$R(W, X) = 0 \tag{10.1}$$

By differentiating the discrete equation with respect to one of the design parameters p_i, gives the sensitivity equation:

$$\frac{\partial R}{\partial W}\frac{\mathrm{d}W}{\mathrm{d}p_i} + \frac{\partial R}{\partial X}\frac{\mathrm{d}X}{\mathrm{d}p_i} = 0, \quad i = 1, ..., N_p \tag{10.2}$$

That rearranging is:

$$\frac{\partial R}{\partial W}\frac{\mathrm{d}W}{\mathrm{d}p_i} = -\frac{\partial R}{\partial X}\frac{\mathrm{d}X}{\mathrm{d}p_i}, \quad i = 1, ..., N_p \tag{10.3}$$

Computation of the gradient by the discrete direct differentiation method The objective scalar function J can be linearized as follows, and we can express the total derivative of the objective function J_f with respect to one design parameter, depending on the mesh and on the flow field (the whole one and the one at the boundaries), $J = J(W, W_b, X)$, using a direct differentiation method:

$$\frac{\mathrm{d}J_f}{\mathrm{d}p_i} = \frac{\partial J_f}{\partial X} \cdot \frac{\mathrm{d}X}{\mathrm{d}p} + \frac{\partial J_f}{\partial W_b} \cdot \frac{\partial W_b}{\partial X} \cdot \frac{\mathrm{d}X}{\mathrm{d}p_i} + \left(\frac{\partial J_f}{\partial W} + \frac{\partial J_f}{\partial W_b} \cdot \frac{\partial W_b}{\partial W} \right) \cdot \frac{\mathrm{d}W}{\mathrm{d}p_i} \tag{10.4}$$

We can also express the total derivative of the functions with respect to the vector of design parameters:

$$\nabla_p J_f = \frac{\partial J_f}{\partial X} \cdot \frac{\mathrm{d}X}{\mathrm{d}p} + \frac{\partial J_f}{\partial W_b} \cdot \frac{\partial W_b}{\partial X} \cdot \frac{\mathrm{d}X}{\mathrm{d}p} + \left(\frac{\partial J_f}{\partial W} + \frac{\partial J_f}{\partial W_b} \cdot \frac{\partial W_b}{\partial W} \right) \cdot \frac{\mathrm{d}W}{\mathrm{d}p} \tag{10.5}$$

If we substitute the term $\dfrac{\mathrm{d}W}{\mathrm{d}p}$ in the previous relations, the gradient of the objective and the constraint functions is written:

$$\nabla_p J_f = \frac{\partial J_f}{\partial X} \cdot \frac{\mathrm{d}X}{\mathrm{d}p} + \frac{\partial J_f}{\partial W_b} \cdot \frac{\partial W_b}{\partial X} \cdot \frac{\mathrm{d}X}{\mathrm{d}p} - \left(\frac{\partial J_f}{\partial W} + \frac{\partial J_f}{\partial W_b} \cdot \frac{\partial W_b}{\partial W} \right) \cdot \frac{\partial R}{\partial W}^{-1} \frac{\partial R}{\partial X} \frac{\mathrm{d}X}{\mathrm{d}p} \tag{10.6}$$

Computation of the gradient by the discrete adjoint method Another approach alternative to the tangent method is the *adjoint method*. Here it has been considered in a discrete form (adjoint equation of discrete scheme) and not in a continuous form (discretization of the continuous adjoint equations of the continuous fluid dynamics equations).

The equations of the adjoint method are introduced from the transposed of the relation 10.6.

$$\nabla_p J_f^T = \left(\frac{\partial J_f}{\partial X} \frac{\mathrm{d}X}{\mathrm{d}p} \right)^T + \left(\frac{\partial J_f}{\partial W_b} \frac{\partial W_b}{\partial X} \frac{\mathrm{d}X}{\mathrm{d}p} \right)^T - \left(\frac{\partial R}{\partial X} \frac{\mathrm{d}X}{\mathrm{d}p} \right)^T \left(\frac{\partial R}{\partial W} \right)^{-T} \left(\frac{\partial J_f}{\partial W} + \frac{\partial J_f}{\partial W_b} \frac{\partial W_b}{\partial W} \right)^T \tag{10.7}$$

reminding that the inverse of a transposed matrix is the transposed of the inverse matrix.

Hence when an adjoint method is chosen, the equation to solve, in which the adjoint vector λ_f for the calculation of the gradient of the f^{th} function of interest, is defined so as to eliminate the terms involving the flow sensitivity $\dfrac{\mathrm{d}W}{\mathrm{d}p}$, is:

$$\left(\frac{\partial R}{\partial W} \right)^T \lambda_f = - \left(\frac{\partial J_f}{\partial W_b} \frac{\partial W_b}{\partial W} + \frac{\partial J_f}{\partial W} \right)^T \tag{10.8}$$

By using this formula for $\nabla_p J_f$, we can rewrite it as:

$$\nabla_p J_f^T = \left(\frac{\partial J_f}{\partial X} \frac{\mathrm{d}X}{\mathrm{d}p} \right)^T + \left(\frac{\partial J_f}{\partial W_b} \frac{\partial W_b}{\partial X} \frac{\mathrm{d}X}{\mathrm{d}p} \right)^T + \left(\frac{\partial R}{\partial X} \frac{\mathrm{d}X}{\mathrm{d}p} \right)^T \lambda_f \tag{10.9}$$

or for the line form:

$$\nabla_p J_f = \frac{\partial J_f}{\partial X}\frac{\mathrm{d}X}{\mathrm{d}p} + \frac{\partial J_f}{\partial W_b}\frac{\partial W_b}{\partial X}\frac{\mathrm{d}X}{\mathrm{d}p} + \lambda_f^T \frac{\partial R}{\partial X}\frac{\mathrm{d}X}{\mathrm{d}p} \tag{10.10}$$

For the derivation of the continuous adjoint equations with respect to a given objective function is derived, before being discretized, the reader is referred to [18].

Let us note that the adjoint method requires the inversion of N_f (the number of functions to be differentiated), while the direct differentiation method requires the solution of N_p 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.

10.1.2 Optimization algorithm

The individuation of the sets of parameters to investigate during the optimization process consists in the problem of finding a local solution to the problem:

$$\textit{minimizing } J(p), \qquad p \in \mathbb{R}^n \qquad \rightarrow \qquad \nabla J(p) = 0 \tag{10.11}$$

The method presented below is the *Line search method*, in which the computation of the following vector of design parameters p_{k+1} is obtained by:

$$p_{k+1} = p_k + \alpha_k S_k \tag{10.12}$$

where S_k is the vector called *descent direction*, which indicates the direction in which the choice of p_{k+1} will head, and α is a scalar coefficient called *descent coefficient* which decides the magnitude of the step to perform in the descent direction. This is not the algorithm that will be used in the optimization performed by python, but it is the easiest to present clearly the optimization subject.

It is convenient to point out the property that the slope $\mathrm{d}J/\mathrm{d}p^*$ at α_k must be zero (to minimize the function J), being p^* the solution, which gives:

$$\nabla J_{k+1}^T S_k = 0 \tag{10.13}$$

From the expression of the derivatives of J along any line $p(\alpha)$, applying the chain rule, the slope of J $(= J(p(\alpha)))$ is:

$$\frac{\mathrm{d}J}{\mathrm{d}\alpha} = \frac{\partial J}{\partial p}\frac{\mathrm{d}p}{\mathrm{d}\alpha} = \nabla J^T S \tag{10.14}$$

The concept of a descent method is associated with the property 10.13, a line search method for which the slope $\mathrm{d}J/\mathrm{d}\alpha$ is always negative at $\alpha = 0$ unless p_k is a stationary point. This condition guarantees that the function can be reduced in the line search for some α_k.

The choice of the descent direction gives the name to several algorithms. In this case the most simple first-order iterative optimization algorithm for finding the minimum of a function could be used, the *gradient direction*, called also *steepest descent*.

The steps are taken proportional to the negative of the gradient of the function at the current point, i.e. the chosen descent direction is $S_k = -\nabla_p J$, because we want to move against the gradient, searching in the steepest descent direction, along which the objective function decreases most rapidly.

To justify this claim, Taylor's theorem is used (read [19]), which says that for any search direction S and parameter α, we have:

$$J(p_k + \alpha S) = J(p_k) + \alpha S^T \nabla J_k + \frac{1}{2}\alpha^2 S^T \nabla^2 J(p_k + tS)S, \text{ for some t } \in (0, \alpha) \tag{10.15}$$

The rate of change in J along the direction S at p_k is simply $S^T \nabla J_k$ (eq. 10.14). Thus direction S of the steepest descent is the solution to the problem:

$$\min_S S^T \nabla J_k, \text{ subject to } ||S|| = 1 \tag{10.16}$$

Since $S^T \nabla J_k = \parallel S \parallel \parallel \nabla J_k \parallel \cos\theta$, where θ is the angle between S and ∇J_k, having $\parallel S \parallel = 1$, $S^T \nabla J_k = \parallel \nabla J_k \parallel \cos\theta$, and the objective function is minimized when $\cos\theta$ is equal to its minimum value -1 at $\theta = \pi$. Finally, the solution to 10.16 is:

$$S = -\frac{\nabla J_k}{\parallel \nabla J_k \parallel} \tag{10.17}$$

as previously anticipated. This direction is orthogonal to the contours of the function for example in fig. 10.1.

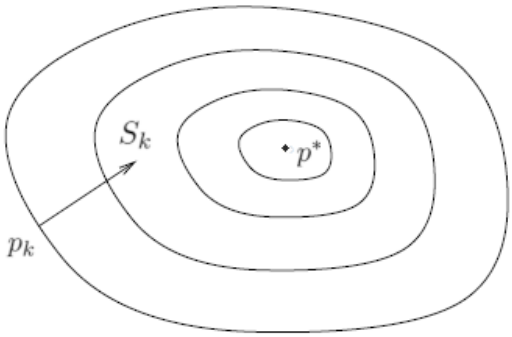


Figure 10.1: Steepest descent direction for a function with two parameters

Unfortunately this method usually exhibits oscillatory behaviour, but for this simple case to examine, it could be sufficient.

The new vector of design functions p_{k+1} is computed such that the objective function obtained at the following step of the optimization process is lower than the previous one:

$$J(p_{k+1}) < J(p_k) \tag{10.18}$$

Developing the inequality,

$$J(p_k + \alpha_k S_k) < J(p_k) \tag{10.19}$$

and linearizing it:

$$J(p_k) + \alpha_k \nabla J^T S_k < J(p_k) \tag{10.20}$$

$$\alpha_k \nabla J_k^T S_k < 0 \tag{10.21}$$

Knowing the descent property

$$\nabla J_k^T S_k < 0 \tag{10.22}$$

it is shown how α_k must be positive.

Thus, ∇J_k is known, and only α_k is unknown.

$J(p(\alpha))$ is determined from several values of J computed for a set of α values, and from interpolation between those computed values.

A simple model to use to find α_k is a *quadratic model*, in which the function is approximated as a quadratic function and the problem consists though in minimizing a quadratic function. In our case it is convenient to use a quadratic method which involves the value of the objective function J, the drag coefficient C_D, at two points for different choices of α_k, and the relative first derivatives $\mathrm{d}J/\mathrm{d}\alpha$.

Set $J = J(p_k + \alpha_k S_k)$, and known the value of the objective function for two values of α_k, α_{k_1} and α_{k_2} and the derivative in one of the two points, it is possible to interpolate to find a parabola.

It is straightforward to use $\alpha_{k_1} = 0$ (for which $J(\alpha_k = 0)$ is known from the previous computation) and its derivative $\dfrac{\mathrm{d}J}{\mathrm{d}\alpha_k}(\alpha_{k_1}) = \nabla J_k^T(\alpha_{k_1})S_k$ (for which $\nabla J_k^T(\alpha_{k_1})$ is again already known from the previous iteration).

Better precision could be obtained through higher order approximations (spline, fourth order etc.), but they would require more computations (each point require a complete computation of C_D with different coefficients α_k). Found α_k which minimize the approximate C_D function, the vector p_{k+1} is ready to perform from the beginning the process.

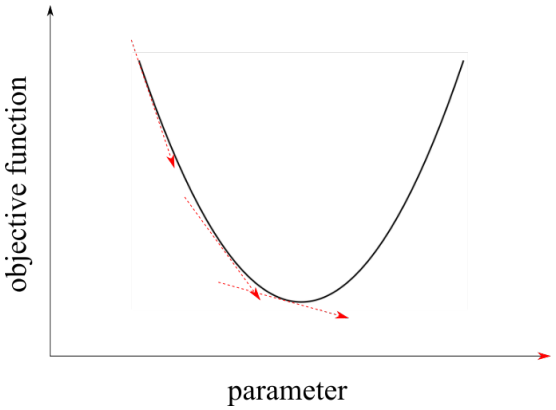


Figure 10.2: Minimization of the objective function - decrease of gradients

10.2 Solution algorithm

The following procedure will be repeated for all the k parameters p_k that the optimizer will suggest, until convergence is reached.

1. Defined a vector of design parameters p_k, an external python solid-surface generation module, after parametrizing the surface with two Ferguson splines (appendix B), provides an external mesh deformer with the new shape, i.e. the coordinates of the mesh points of the solid surfaces $S(p)$ and its derivative with respect to the design parameters $\dfrac{\partial S}{\partial p}(p)$.

2. An external python volume grid deformation module computes the new volume mesh $X(p)$ and also the derivatives of the grid coordinates with respect to the design parameters, via *Inverse Distance Weighted* deformation method (appendix C). The volume mesh corre-

sponding to a new set of parameters is constructed by mesh deformation of the original volume mesh $\frac{\partial X}{\partial p}(p)$.

3. The external steady solver (or elsA for its shorter computational times in this development phase), computes the aerodynamic flow field variables $W(t)$ for iteration k in the same way as has been done for the steady solver.

4. The optimization module by elsA, known the field $W(p)$ and the derivative of the mesh $\frac{\partial X}{\partial p}(p)$, computes the Jacobian matrix and the right hand side $-\frac{\partial R}{\partial X}\frac{\mathrm{d}X}{\mathrm{d}p}$.

   Besides the *elsA opt module* solves the linearized equation 10.3, and $\frac{\mathrm{d}W}{\mathrm{d}p}$, the derivative of the flow conservative variables - called *sensitivity* - is obtained (in following updates of the optimizer this step could be performed externally by the python solver).

5. The external solver evaluates the objective functions J_f and their derivatives with respect to the flow field (cell-centered and boundary values) and mesh coordinates, $\frac{\partial J_f}{\partial W}$, $\frac{\partial J_f}{\partial W_b}$, $\frac{\partial J_f}{\partial X}$, with $J_f = C_{D_p}, C_L$.

6. The elsA optimization module assembles the gradients, in order to compute the gradient of the objective functions with respect to the design parameters $\frac{\mathrm{d}C_{D_p}}{\mathrm{d}p}$ and $\frac{\mathrm{d}C_L}{\mathrm{d}p}$.

7. The external solver, exploiting the *scipy.optimize.minimize* python module, using a Sequential Least SQuares Programming, implements a gradient based optimization, in which it computes the following values of the design parameters p_{k+1}. The process starts again from point number 1, with the new set of design parameters.

8. The process is stopped, and the parameters which reduce the Pressure Drag Coefficient C_{D_p} keeping the Lift Coefficient C_L above a prescribed value are found, when the chosen convergence criterion is reached.

10.2.1 Computation of the new mesh and its derivatives

After computing $S(p)$ and $\frac{\mathrm{d}S}{\mathrm{d}p}$ thanks to a Ferguson spline interpolation, these two quantities are provided to the mesh deformation module.

In order to evaluate the new coordinates of the grid X_{k+1} and the new sensitivities $\frac{\mathrm{d}X}{\mathrm{d}p}$, the same method is used, the *Inverse Distance Weighted* method:

$$X_{k+1} = X_k + \delta X \tag{10.23}$$

$$\left.\frac{\mathrm{d}X}{\mathrm{d}p}\right|_{k+1} = \left.\frac{\mathrm{d}X}{\mathrm{d}p}\right|_{k+1} + \delta\,\frac{\mathrm{d}X}{\mathrm{d}p} \tag{10.24}$$

The increments δX, is given in eq. C.1, with δS, i.e. the displacements of the nodes on the surface of the airfoil with respect to the initial surface, given as input.

Simply changing the input into $\frac{\mathrm{d}S}{\mathrm{d}p}$, the output of the mesh deformation equation is $\delta \frac{\mathrm{d}X}{\mathrm{d}p}$, that is the increment of the sensitivity of the mesh with respect to the nominal one.

In first instance building a parametrized clone of the nominal shape of the *NACA 64A10* is required: starting from a random airfoil shape, a minimized residual based python optimization is carried out, in order to approximate in the best possible way the initial shape (10.3). This step is performed only once.

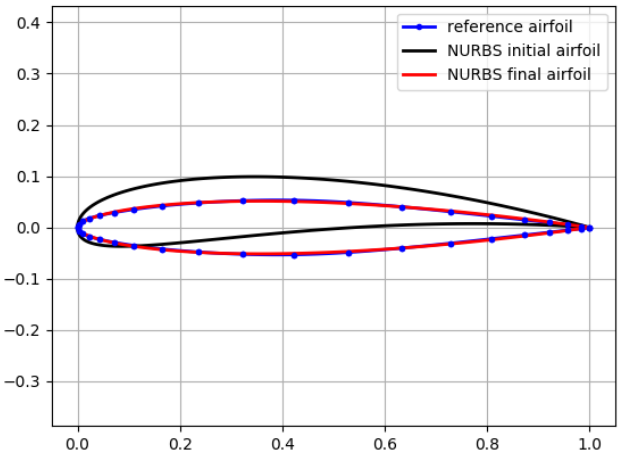


Figure 10.3: *NACA64A10* airfoil initial parametrization starting from a random shape

10.2.2 Computation of the flow field variables

The first equation to solve is the usual one for a steady flow:

$$R(W) = 0 \tag{10.25}$$

which, applying backward Euler, becomes:

$$\left(\frac{V}{\Delta\tau} + \frac{\partial R}{\partial W}\right)\Delta W = -R \tag{10.26}$$

The result of this first step is the solution vector of conservative variables of the steady problem, then exactly the solution of the steady solver built in first instance.

10.2.3 Computation of the flow field sensitivities

Obtained the field and the deformed mesh, it is necessary to compute the derivatives of the conservative variables of this field with respect to the design parameters p_i, $\frac{\mathrm{d}W}{\mathrm{d}p_i}$ for the tangent/direct approach, or the adjoint vector λ for the adjoint approach.

This step is temporarily performed by elsA in the optimization module, restarting from the steady mean flow solution $W(p)$ and providing the sensitivity of the mesh with respect to the design parameters $W(p)$ and $\frac{\mathrm{d}X}{\mathrm{d}p}$ for the tangent/direct approach, and exploiting also the provided

gradients of the objective and constraint functions (see following paragraphs) to solve the linear system.

In next updates of the optimizer, the computation of $\frac{\mathrm{d}W}{\mathrm{d}p}$ (or λ) will be external, thanks to the possibility of the extraction of the Jacobian matrix $\frac{\partial R}{\partial W}$ left to elsA as previously done for the steady and TSM external solver, as well as the right hand side $\frac{\partial R}{\partial X}\frac{\mathrm{d}X}{\mathrm{d}p}$.

The gradients $\frac{\partial W_b}{\partial X}$ and $\frac{\partial W_b}{\partial W}$ are computed internally in elsA, but the possibility to compute them externally is not excluded.

10.2.4 Evaluation of the objective functions and their derivatives

To compute the gradients of the objective and constraint function with respect to the flow field, the flow field at the boundaries and the mesh, a definition of $\frac{\partial J}{\partial W}$, $\frac{\partial J}{\partial W_b}$, $\frac{\partial J}{\partial X}$, that are still unknown in equation 10.4, needs to be found.

- $\boxed{\frac{\partial J}{\partial W}}$
  Since in this case the objective functions J are lift and drag coefficients, they don't depend on the whole mesh and state vector, but only on the mesh and the state vector on the wing surface.

$$\frac{\partial J}{\partial W} = 0 \tag{10.27}$$

- $\boxed{\frac{\partial J}{\partial W_b}}$
  Retrieving the formulation of the x and z coefficients (x and z are the body axis, while C_L and C_{D_p} are expressed in wind axis).

$$C_x = \frac{\int_{\mathcal{S}} p(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_x}{q_\infty \cdot S_{ref}} \tag{10.28}$$

$$C_z = \frac{\int_{\mathcal{S}} p(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_z}{q_\infty \cdot S_{ref}} \tag{10.29}$$

  the dependence of the x and z coefficient on the flow field conservative variables, consists in a dependence only of the pressure on the conservative variables, because the term involving the normal vector of the surface depends only on the mesh.

$$\frac{\partial C_x}{\partial W_b} = \frac{\int_{\mathcal{S}} \frac{\partial p}{\partial W_b}(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_x}{q_\infty \cdot S_{ref}} \tag{10.30}$$

$$\frac{\partial C_z}{\partial W_b} = \frac{\int_{\mathcal{S}} \frac{\partial p}{\partial W_b} (\mathbf{n} \cdot \mathbf{d}\mathcal{S})_z}{q_\infty \cdot S_{ref}} \tag{10.31}$$

In order to find the derivative of the pressure with respect to the conservative variables it is necessary to express the pressure, p, in terms of combination of the conservative variables.

$$p = (\gamma - 1)\left[\rho E - \frac{1}{2}(\rho u^2 + \rho v^2 + \rho w^2)\right] \tag{10.32}$$

After some developments, the gradients of the pressure are written below:

$$\frac{\partial p}{\partial W_b} = \begin{pmatrix} \frac{\partial p}{\partial \rho} \\ \frac{\partial p}{\partial \rho u} \\ \frac{\partial p}{\partial \rho v} \\ \frac{\partial p}{\partial \rho w} \\ \frac{\partial p}{\partial \rho E} \end{pmatrix}_b = \begin{pmatrix} \frac{1}{2}(\gamma - 1)(u^2 + v^2 + w^2) \\ -(\gamma - 1)u \\ -(\gamma - 1)v \\ -(\gamma - 1)w \\ (\gamma - 1) \end{pmatrix} \tag{10.33}$$

Concerning the projections in directions x and z of the normal vector to the surface multiplied by the surface of the cell itself $\mathbf{n} \cdot \mathbf{d}\mathcal{S}$ is given by cross product. On the airfoil, four points of the mesh (1,2,3,4) define a cell interface, as in fig. 10.4.

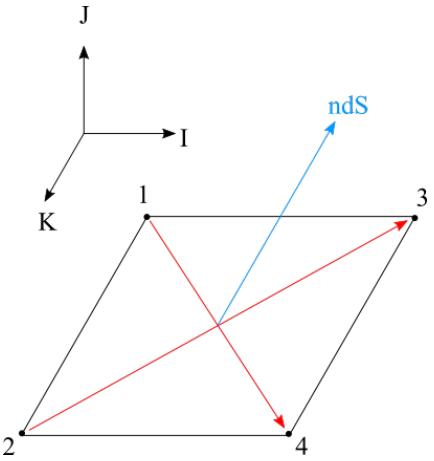


Figure 10.4: Normal vector to a mesh cell interface

$\mathbf{n} \cdot \mathbf{d}\mathcal{S}$ is obtained by the product $\mathbf{14} \times \mathbf{23}$.

$$(\mathbf{n} \cdot \mathbf{d}\mathcal{S})_x = \frac{1}{2}\left[(y_4 - y_1)(z_3 - z_2) - (y_3 - y_2)(z_4 - z_1)\right] \tag{10.34}$$

$$(\mathbf{n} \cdot \mathbf{d}\mathcal{S})_z = \frac{1}{2}\left[(x_4 - x_1)(y_3 - y_2) - (x_3 - x_2)(y_4 - y_1)\right] \tag{10.35}$$

Last thing to keep in mind is the necessity to rotate the drag and lift coefficients gradients to transform them from a body axis reference to a wind axis reference (as required by elsA).

$$\frac{\partial C_{D_p}}{\partial W_b} = \frac{\partial C_x}{\partial W_b}\cos\alpha_0 + \frac{\partial C_z}{\partial W_b}\sin\alpha_0 \tag{10.36}$$

$$\frac{\partial C_L}{\partial W_b} = -\frac{\partial C_x}{\partial W_b}\sin\alpha_0 + \frac{\partial C_z}{\partial W_b}\cos\alpha_0 \tag{10.37}$$

- $\boxed{\frac{\partial J}{\partial X}}$

The derivative of J with respect to the mesh coordinates has an equivalent expression (it was demostrated that $\frac{\partial p}{\partial x} = \frac{\partial p}{\partial y} = \frac{\partial p}{\partial z} = 0$):

$$\frac{\partial C_x}{\partial x,y,z} = \frac{\int_{\mathcal{S}} p\frac{\partial(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_x}{\partial x}}{q_\infty \cdot S_{ref}}, \frac{\int_{\mathcal{S}} p\frac{\partial(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_x}{\partial y}}{q_\infty \cdot S_{ref}}, \frac{\int_{\mathcal{S}} p\frac{\partial(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_x}{\partial z}}{q_\infty \cdot S_{ref}} \tag{10.38}$$

$$\frac{\partial C_z}{\partial x,y,z} = \frac{\int_{\mathcal{S}} p\frac{\partial(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_z}{\partial x}}{q_\infty \cdot S_{ref}}, \frac{\int_{\mathcal{S}} p\frac{\partial(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_z}{\partial y}}{q_\infty \cdot S_{ref}}, \frac{\int_{\mathcal{S}} p\frac{\partial(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_z}{\partial z}}{q_\infty \cdot S_{ref}} \tag{10.39}$$

The derivatives of the normal vectors with respect to the mesh coordinates, keeping in mind the figure 10.5 and the equations 10.34 and 10.35 are the following, for the x-vector:

$$\frac{\partial(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_x}{\partial x_1} = 0, \quad \frac{\partial(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_x}{\partial y_1} = -\frac{1}{2}(z_3 - z_2), \quad \frac{\partial(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_x}{\partial z_1} = +\frac{1}{2}(y_3 - y_2) \tag{10.40}$$

$$\frac{\partial(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_x}{\partial x_2} = 0, \quad \frac{\partial(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_x}{\partial y_2} = +\frac{1}{2}(z_4 - z_1), \quad \frac{\partial(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_x}{\partial z_2} = -\frac{1}{2}(y_4 - y_1) \tag{10.41}$$

$$\frac{\partial(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_x}{\partial x_3} = 0, \quad \frac{\partial(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_x}{\partial y_3} = -\frac{1}{2}(z_4 - z_1), \quad \frac{\partial(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_x}{\partial z_3} = +\frac{1}{2}(y_4 - y_1) \tag{10.42}$$

$$\frac{\partial(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_x}{\partial x_4} = 0, \quad \frac{\partial(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_x}{\partial y_4} = +\frac{1}{2}(z_3 - z_2), \quad \frac{\partial(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_x}{\partial z_4} = -\frac{1}{2}(y_3 - y_2) \tag{10.43}$$

and for the z-vector:

$$\frac{\partial(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_z}{\partial x_1} = -\frac{1}{2}(y_3 - y_2), \quad \frac{\partial(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_z}{\partial y_1} = +\frac{1}{2}(x_3 - x_2), \quad \frac{\partial(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_z}{\partial z_1} = 0 \tag{10.44}$$

$$\frac{\partial(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_z}{\partial x_2} = +\frac{1}{2}(y_4 - y_1), \quad \frac{\partial(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_z}{\partial y_2} = -\frac{1}{2}(x_4 - x_1), \quad \frac{\partial(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_z}{\partial z_2} = 0 \tag{10.45}$$

$$\frac{\partial(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_z}{\partial x_3} = -\frac{1}{2}(y_4 - y_1), \quad \frac{\partial(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_z}{\partial y_3} = +\frac{1}{2}(x_4 - x_1), \quad \frac{\partial(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_z}{\partial z_3} = 0 \tag{10.46}$$

$$\frac{\partial(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_z}{\partial x_4} = +\frac{1}{2}(y_3 - y_2), \quad \frac{\partial(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_z}{\partial y_4} = -\frac{1}{2}(x_3 - x_2), \quad \frac{\partial(\mathbf{n}\cdot\mathbf{d}\mathcal{S})_z}{\partial z_4} = 0 \tag{10.47}$$

The overall number of terms will be twice the number of nodes on the airfoil surface (193x2 = 386). It is noticeable that, being the space discretization a *finite volume* approach, the pressure is known in the cell centers, while the derivatives of the normal vectors with respect to the grid coordinates are computed in each of the 4 vertices of each of the interfaces of the cells over the airfoil surface.

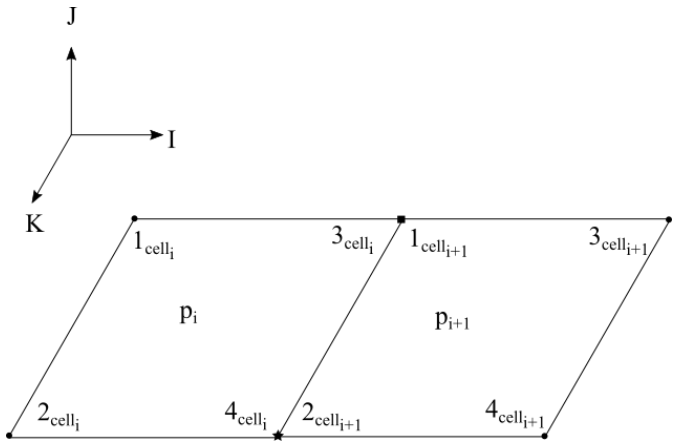


Figure 10.5: Numerical grid for gradients computation

From a computational point of view, given $i = 1$ (so $i + 1 = 2$) and marking with ★ the node at the interface $y = 0$ and ■ the node at the interface $y = 1$, the gradient of the x coefficient with respect to the mesh coordinates in the z-directions in the two nodes is computed as:

$$\left(\frac{\partial C_x}{\partial z}\right)_{node\bigstar} = p_{cell1}\left(\frac{\partial(\mathbf{n}\cdot\mathbf{dS})_x}{\partial z_4}\right)_{cell1} + p_{cell2}\left(\frac{\partial(\mathbf{n}\cdot\mathbf{dS})_x}{\partial z_2}\right)_{cell2} \tag{10.48}$$

$$\left(\frac{\partial C_x}{\partial z}\right)_{node\blacksquare} = p_{cell1}\left(\frac{\partial(\mathbf{n}\cdot\mathbf{dS})_x}{\partial z_3}\right)_{cell1} + p_{cell2}\left(\frac{\partial(\mathbf{n}\cdot\mathbf{dS})_x}{\partial z_1}\right)_{cell2} \tag{10.49}$$

The same reasoning is valid for the other two directions and in an analogue way for the z coefficient (using the z-component of the normal).

Last thing to compute is the rotation of the gradients in a wind axis frame of reference, to obtain the actual objective function C_{D_p} and constraint function C_L:

$$\frac{\partial C_{D_p}}{\partial x,y,z} = \frac{\partial C_x}{\partial x,y,z}\cos\alpha_0 + \frac{\partial C_z}{\partial x,y,z}\sin\alpha_0 \tag{10.50}$$

$$\frac{\partial C_L}{\partial x,y,z} = -\frac{\partial C_x}{\partial x,y,z}\sin\alpha_0 + \frac{\partial C_z}{\partial x,y,z}\cos\alpha_0 \tag{10.51}$$

10.2.5 Computation of the updated values of p

This step of the optimization algorithm is performed by the optimization module in python, which, known the values of the drag and lift coefficients and their gradients with respect to the design parameters C_{D_p}, C_L, $\nabla_p C_{D_p}$, $\nabla_p C_L$, finds as output of one step of the optimization process the new set of design parameters p_{k+1}.

The python external optimizer uses a *Sequential Least SQuares Programming*approach. Sequential quadratic programming (SQP) is an iterative method for constrained non-linear optimization problems ([20] and [19]).

Considering the SQP methodology to non linear optimization problems (NLP) of the form (read [21]):

$$\text{minimize } f(x) \tag{10.52}$$
$$\text{over } x \in \mathbb{R}^n \tag{10.53}$$
$$\text{subject to } h(x) = 0 \tag{10.54}$$
$$g(x) \leq 0 \tag{10.55}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is the objective function, the functions $h : \mathbb{R}^n \to \mathbb{R}^m$ and $g : \mathbb{R}^n \to \mathbb{R}^p$ describe the equality and inequality constraints. SQP is an iterative procedure which models the NLP for a given iterate x^k, $k \in \mathbb{N}$, by a Quadratic Programming (QP) sub-problem, solves that QP sub-problem, and then uses the solution to construct a new iterate x^{k+1}. This construction is done in such a way that the sequence $(x^k)_{k\in\mathbb{N}}$ converges to a local minimum of the non-linear problem, as $k \to \infty$.

In our case C_{D_p} is the objective function,the vector of the six parameters $x \in \mathbb{R}$ is:

$$p \in \mathbb{R}^6, \quad p = [T_{A_u}, T_{A_l}, T_{B_u}, T_{B_l}, \alpha_b, \alpha_c] \tag{10.56}$$

the inequality constraint is:

$$C_L \geq C_{L_{min}} \tag{10.57}$$

The whole process is summed up in the following flow chart.

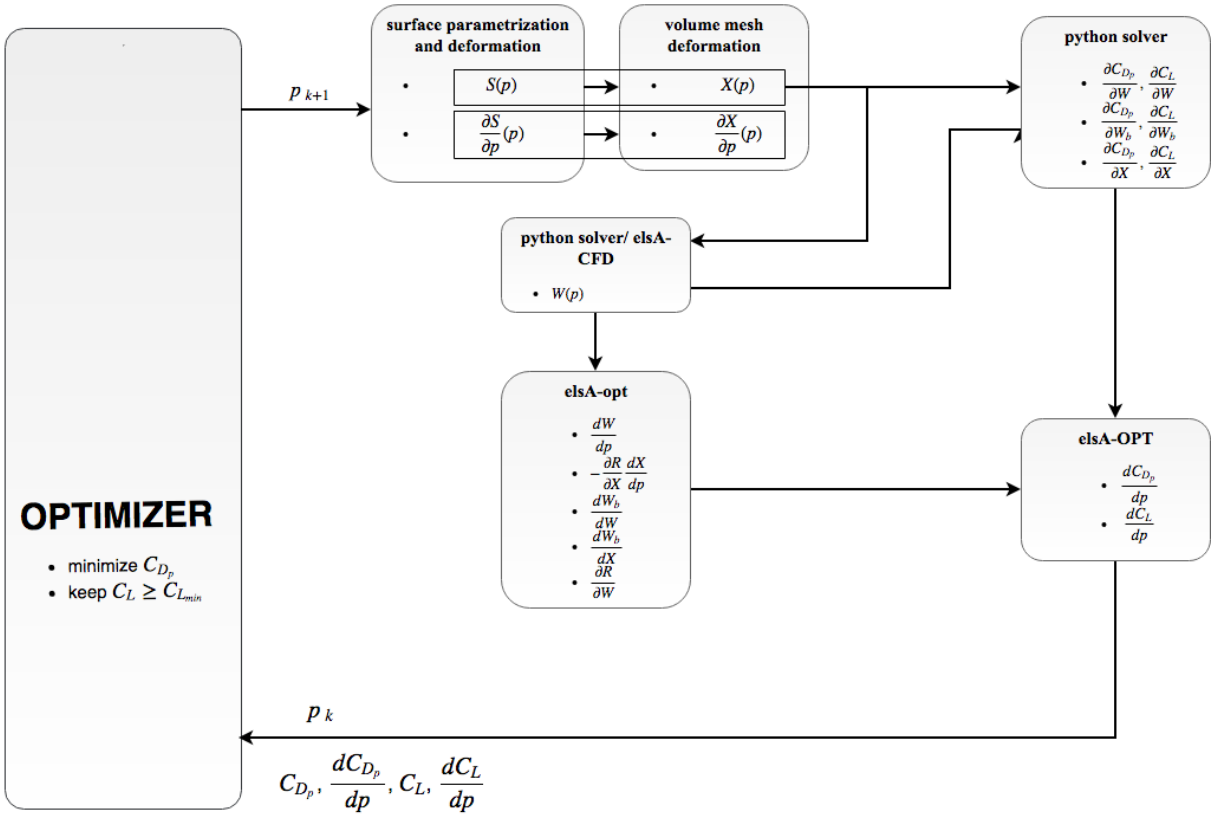


Figure 10.6: Optimization process algorithm

10.3 Results

The shape optimization process is carried out on the airfoil *NACA 64A10*, in a transonic regime, with an angle of attack of $\alpha = 1°$.

M	=	0.796
p_∞	=	133912 Pa
$p°$	=	203321 Pa
q	=	59395 Pa
α	=	1°

Table 10.1: Conditions for the optimization test case

The boundaries imposed in the optimization process to the six parameters are:

$$\begin{array}{lcccl} 0.05 & \leq & T_{A_u} & \leq & 0.4 \\ 0.05 & \leq & T_{A_l} & \leq & 0.4 \\ 0.05 & \leq & T_{B_u} & \leq & 3 \\ 0.05 & \leq & T_{B_l} & \leq & 3 \\ 1 & \leq & \alpha_b & \leq & 30 \\ -15 & \leq & \alpha_c & \leq & 15 \end{array}$$

The initial airfoil parametrization returns a clone wall surface with the following parameters and aerodynamic coefficients:

Initial C_{D_p} value		$4.82164 \cdot 10^{-3}$
Initial C_L value		$2.31423 \cdot 10^{-1}$
Initial parameters	T_{A_u}	0.0913119
	T_{A_l}	0.0913155
	T_{B_u}	2.14342
	T_{B_l}	2.14339
	α_b	15.8717
	α_c	−7.93587

Table 10.2: Initial airfoil parametrization and coefficients

The $C_{L_{min}} = 2.31423 \cdot 10^{-1}$ is the lift coefficient of the nominal initial airfoil, that is chosen as constraint.

In order to check the gradients computed by the interaction of the external solver and elsA, they are compared by finite differences at the first iteration of the optimizer:

	Finite differences	Adjoint mode	Direct mode
$\frac{\partial C_D}{\partial T_{A_u}}$	$7.66351 \cdot 10^{-2}$	$7.02767 \cdot 10^{-2}$	$7.02759 \cdot 10^{-2}$
$\frac{\partial C_D}{\partial T_{A_l}}$	$-3.42698 \cdot 10^{-3}$	$-3.91359 \cdot 10^{-3}$	$-3.91397 \cdot 10^{-3}$
$\frac{\partial C_D}{\partial T_{B_u}}$	$1.74965 \cdot 10^{-2}$	$1.71265 \cdot 10^{-2}$	$1.71263 \cdot 10^{-2}$
$\frac{\partial C_D}{\partial T_{B_l}}$	$-1.11892 \cdot 10^{-3}$	$-1.19655 \cdot 10^{-3}$	$-1.19663 \cdot 10^{-3}$
$\frac{\partial C_D}{\partial \alpha_b}$	$4.03267 \cdot 10^{-3}$	$3.94100 \cdot 10^{-3}$	$3.94086 \cdot 10^{-3}$
$\frac{\partial C_D}{\partial \alpha_c}$	$5.44220 \cdot 10^{-3}$	$5.28892 \cdot 10^{-3}$	$5.28886 \cdot 10^{-3}$
$\frac{\partial C_L}{\partial T_{A_u}}$	$7.87137 \cdot 10^{-1}$	$7.56865 \cdot 10^{-1}$	$7.56831 \cdot 10^{-1}$
$\frac{\partial C_L}{\partial T_{A_l}}$	$-5.53981 \cdot 10^{-1}$	$-5.53092 \cdot 10^{-1}$	$-5.53106 \cdot 10^{-1}$
$\frac{\partial C_L}{\partial T_{B_u}}$	$1.62053 \cdot 10^{-1}$	$1.45485 \cdot 10^{-1}$	$1.45475 \cdot 10^{-1}$
$\frac{\partial C_L}{\partial T_{B_l}}$	$-1.15664 \cdot 10^{-1}$	$-1.16509 \cdot 10^{-1}$	$-1.16512 \cdot 10^{-1}$
$\frac{\partial C_L}{\partial \alpha_b}$	$7.73444 \cdot 10^{-2}$	$7.29992 \cdot 10^{-2}$	$7.29939 \cdot 10^{-2}$
$\frac{\partial C_L}{\partial \alpha_c}$	$1.47271 \cdot 10^{-1}$	$1.40437 \cdot 10^{-1}$	$1.40434 \cdot 10^{-1}$

Table 10.3: Finite Differences based check of gradients - Adjoint and Direct methods

After performing the optimization process, the results are in table 10.4.

Convergence tolerance		10^{-7}
Δp for gradient computation		10^{-4}
Iterations		19
Function evaluations		44
Gradient computations		19
CPU time		$\sim 2hours$
Final C_{D_p} value		$8.01731 \cdot 10^{-4}$
Final C_L value		$2.31704 \cdot 10^{-1}$
Final parameters	T_{A_u}	0.117169
	T_{A_l}	0.175912
	T_{B_u}	1.29071
	T_{B_l}	1.22150
	α_b	15.9975
	α_c	-7.38225

Table 10.4: Optimized airfoil parametrization and coefficients, with computational features

The pressure drag coefficient has decreased of 6 times, keeping the lift coefficient above the initial value. A slight asymmetry is introduced in the shape of the airfoil, especially in the front part (10.7). The upper surface, as expected, has become flatter than the lower one, useful to smooth and/or delete the discontinuity that a transonic regime brings with it.

Possibly because of both the too restrictive Ferguson splines' parametrization of the airfoil, and the mixing of length and angles in optimization process, the camber, driven by the the angles α_b and α_c did not change consistently. The optimization process mostly lead to an expected thinning of the airfoil surface.

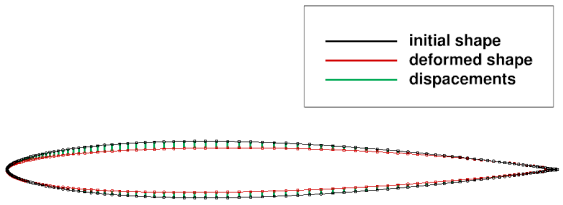


Figure 10.7: Initial and optimized shape

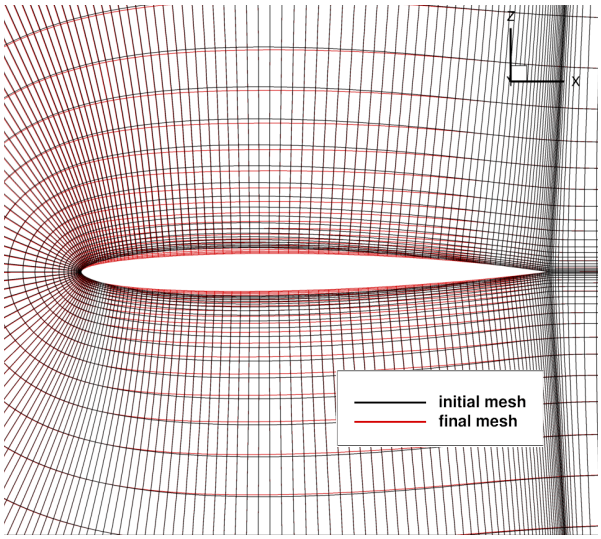


Figure 10.8: Initial and optimized computational mesh

In fact, as shown in fig. 10.9 and fig. 10.10 the initial shock wave present with the nominal airfoil is suppressed for the optimized shape.

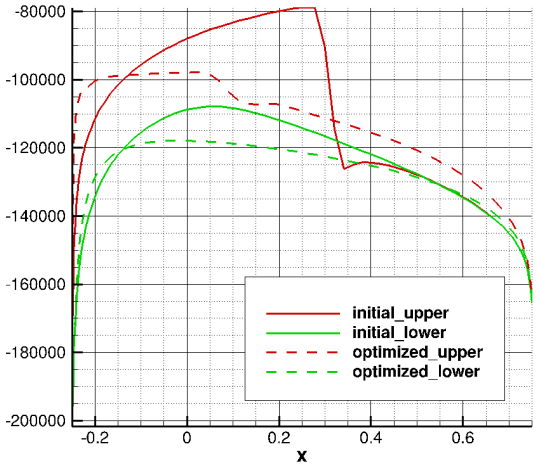


Figure 10.9: Wall pressure around the initial and optimized airfoil

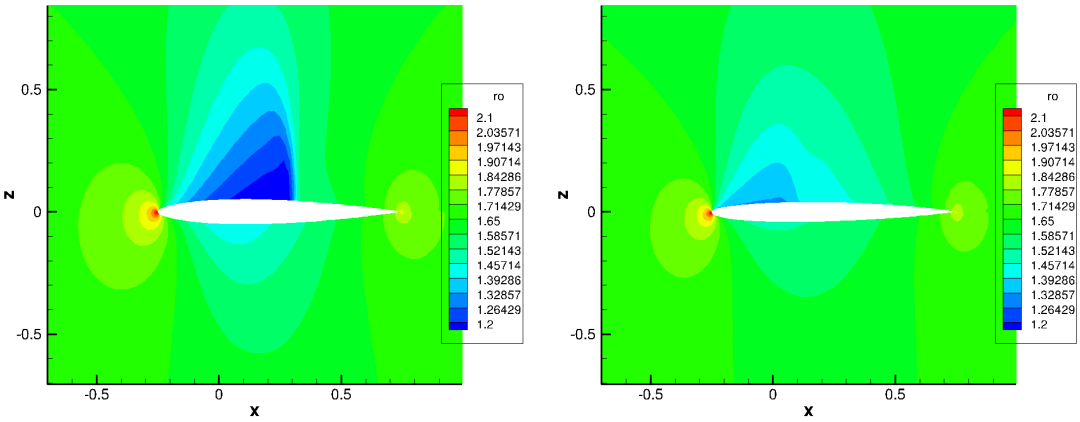


Figure 10.10: Density field around the initial and optimized airfoil

The optimization history is illustrated in fig. 10.11: the drag coefficient drops to almost the final minimum value in only 7 iterations, but with a decrease of the lift coefficient, that is not allowed. The following iterations are necessary to stabilize the lift coefficient.

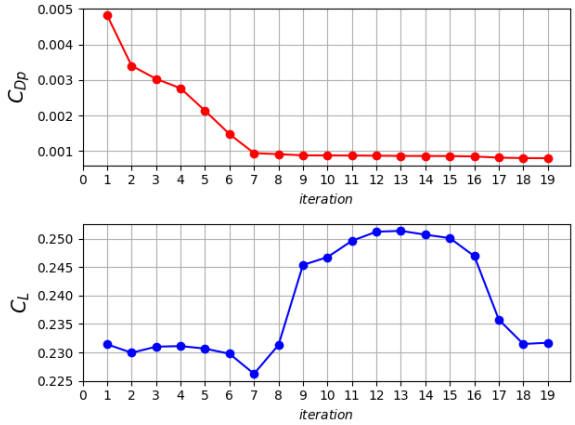


Figure 10.11: Optimization history, coefficients C_{D_p} and C_L

Observing the optimization history of the six parameters in fig. 10.12, the two angles α_b and α_c, together with the magnitudes of the trailing edge tangent vectors T_{B_u} and T_{B_l}, seem well stabilized.

Instead, the optimization process for the magnitudes of the leading edge tangent vectors T_{A_u} and T_{A_l} has not stabilized yet.

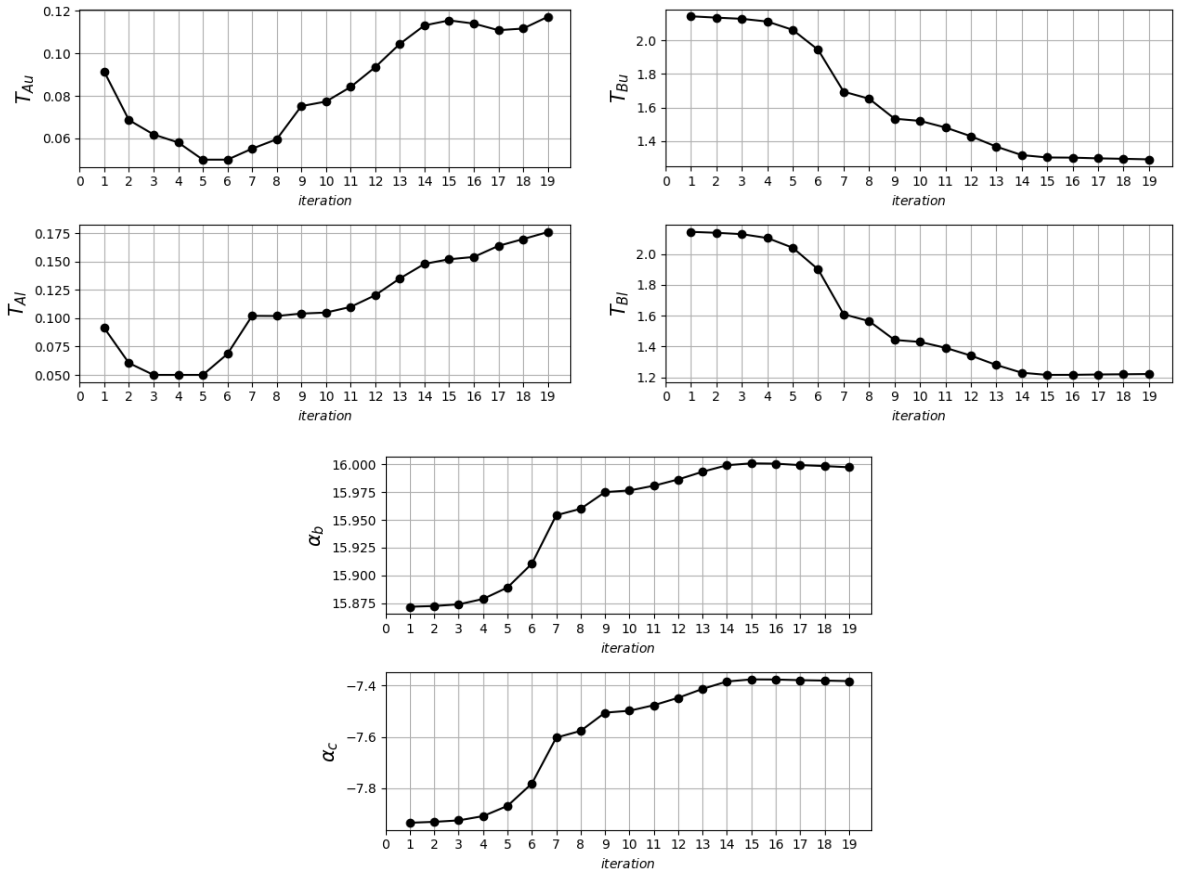


Figure 10.12: Optimization history, parameters T_{A_u}, T_{A_l}, T_{B_u}, T_{B_l}, α_b, α_c.

As observed for the optimization history of the parameters, also the gradients of the objective function C_{D_p} with respect to the parameters α_c, α_b, T_{B_u}, T_{B_l} follow the expected behaviour, that is a decrease of the gradient in absolute value during the optimization process (fig. 10.2).

On the other hand, the gradients of C_{D_p} with respect to the parameters T_{A_u} and especially T_{A_l} changes sign and don't properly stabilize to the minimum value, but by the way they indicate the right optimization path, that leads to a decrease of the pressure drag coefficient of 6 times in only 19 iterations.

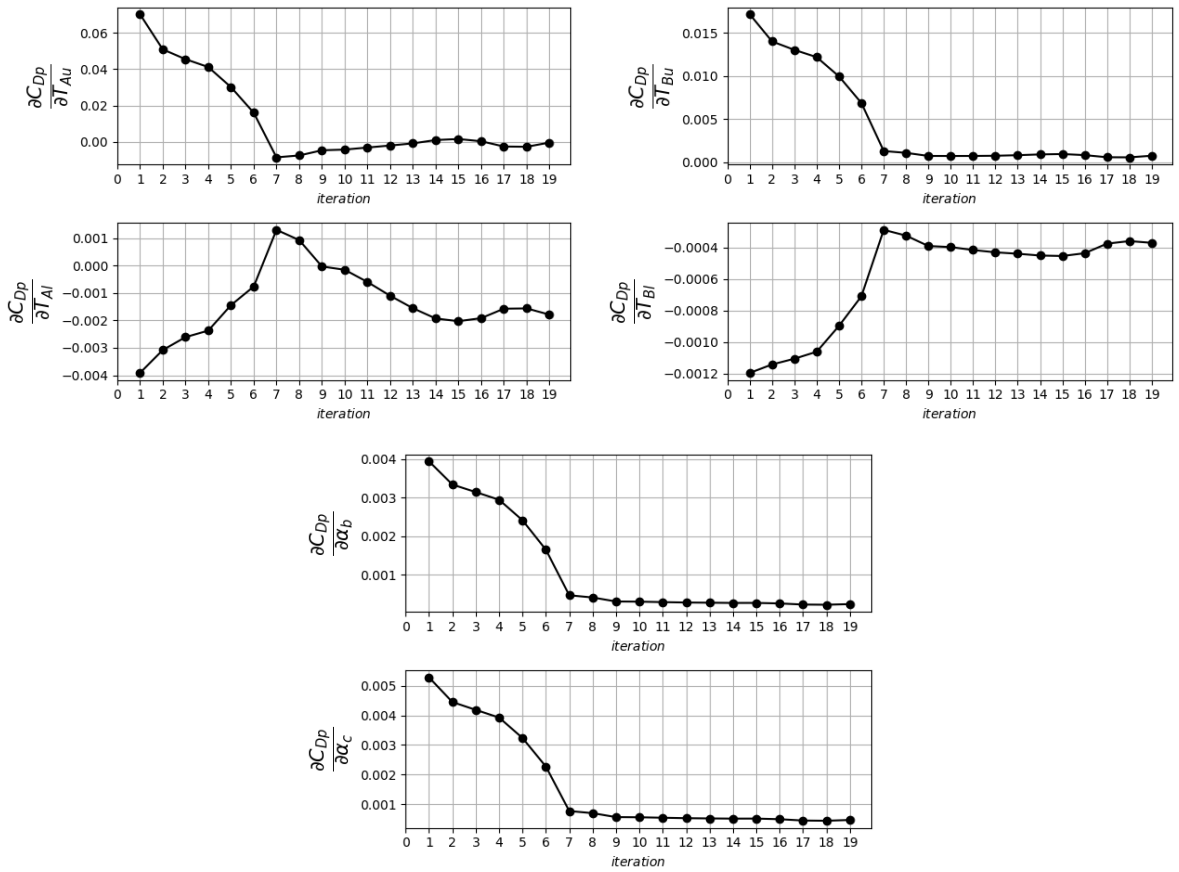


Figure 10.13: Optimization history, C_{D_p} gradients: $\frac{\partial C_{D_p}}{\partial T_{A_u}}, \frac{\partial C_{D_p}}{\partial T_{A_l}}, \frac{\partial C_{D_p}}{\partial T_{B_u}}, \frac{\partial C_{D_p}}{\partial T_{B_l}}, \frac{\partial C_{D_p}}{\partial \alpha_b}, \frac{\partial C_{D_p}}{\partial \alpha_c}$.

Chapter 11

Conclusion and perspectives

11.1 TSM external solver

The main aim of the work was developing a TSM external solver written in python, based on the exchange of data between the external solver and the CFD solver elsA.

The aim has been achieved, the solver has been validated by checks using both the *unsteady* simulations and the *TSM* simulations by elsA. Moreover it has been improved by computing in parallel the instants, leading to a consistent saving of time, and by fully impiciting the TSM method, gaining in terms of robustness.

In terms of rate of convergence, a gain in terms of iterations with respect to elsA's TSM solver to obtain the same residuals decrease has been also demonstrated, despite the use of a direct solver put severe constraints on the CFL number. Because of the different nature of the Jacobian matrix exploited in the external solver and the one exploited in elsA, making comparisons between the two at the same CFL number is impossible, but comparisons can be done analysing the best CFL strategy for each one.

The presented work was a first implementation of the TSM solver, on which many further improvements will be added during the prosecution of the project.

A final remark about the Time Spectral Method is that, despite the great saving of time they can bring with it with respect to unsteady classical methods, it is not universally well adapted to every kind of application: in fact if a highly non-linear case is treated, a high number of harmonics needs to be performed. This leads to a loss of all benefits of TSM, because the computation becomes less robust, longer and much more expensive, and a classical unsteady computation can still be the right option.

11.1.1 Limitations

As mentioned several times in the report, a lack of robustness has been experienced. This is due to two different reasons:

- One is linked to the nature of the exact Jacobian matrix of second order, that is more dense than an approximated Jacobian matrix or an exact Jacobian matrix of first order. This means that this kind of matrix is stiffer than the one exploited by elsA in the steady mean flow solution process, and thus in the inversion of the second order exact Jacobian matrix only low CFL numbers can be afforded, especially if a direct solver is used.

- The second reason is linked to the nature of the Time Spectral Method implemented using the *fully implicit TSM*. Even if the robustness is increased with respect to the *partially implicit TSM* with the explicit source term, this method still suffers divergence while increasing both the number of harmonics and the frequency (also elsA, which implements the same method suffers this kind of divergence).

Another limitation due to the use of a direct solver is in the mesh size: a direct solver involves storing the whole matrix (in a sparse mode in our case) and directly inverting it. It is easy to understand that increasing the size of the mesh, the size of the Jacobian matrix will increase exponentially and memory problems can be faced.

Moreover, being this one a first implementation, it is not very efficient from an IT point of view: the simulations have been launched in local, with strong limitations in the use of processors, and the exchanging of data (residuals and Jacobian matrix from elsA to the external solver, and the restart field from the external solver to elsA) was carried out by writing and reading files. This has enormous costs in terms of CPU time.

11.1.2 Perspectives

The first improvement to add will be a new resolution strategy for linear systems: a *Generalized Minimal Residual Method* solver with an ILU preconditioner. This iterative solver, together with an efficient preconditioner, is both much robust than the direct solver, and less memory storage expensive, because the Jacobian matrix is not fully stored in memory. This first new implementation is necessary to solve problems with higher mesh sizes, and also it is expected to solve or at least relax the constraints due to the initial stiffness of the matrix, being this a solver that does not require diagonal dominance of the Jacobian. The use of GMRES as the linear solver is expected to make time spectral method more robust and efficient, allowing it to be applied to a greater subset of time-accurate problems, including those with a broad range of harmonic content, that would further increase the stiffness of the matrix making impossible the use of direct solvers.

Moreover a new formulation of the problem will be adopted, in order to avoid dependence on frequency and number of instants. A wave-number independent preconditioner that mitigates the increased stiffness of the time-spectral method when applied to problems with large resolvable wave numbers will be developed, as Mundis and Mavriplis suggest [22].

From the IT efficiency point of view, the exploitation of elsA's modularity through an in memory pointers approach is already on going. This will highly speed up the exchange between elsA and the external solver.

Also cluster computations have been recently allowed, after compiling the adapted elsA version on the ONERA cluster, leading, for the moment, to the possibility to launch several and faster computations. In the future the cluster computations will be mandatory, because a Message Passing Interface (MPI) implementation will be developed for multi-block meshes. This means that the simulations will be much faster because to a first level of parallel computing (instants), a second one (the blocks) will be added.

Finally the code will move to the implementation of RANS equations, and on a supercritical airfoil, adding new non-linearities and difficulties.

11.2 Optimization

Even if a shorter period of time has been dedicated to the development of an external python steady optimizer, the results obtained are very promising: despite the initial development phase, a decrease in pressure drag of a factor 6 has been achieved in few iterations.

11.2.1 Limitations

This is not properly a limitation, but an incomplete implementation: the optimizer developed is a steady optimizer, while the final goal is the building of an unsteady optimizer.

Other proper limitations to be improved soon are the airfoil parametrization that is too restrictive and cannot face accurately all kinds of shape, and the implementation in a black-boxed python function that does not permit to handle easily all setting parameters and function and gradient evaluations which leads to redundant expensive computations. In fact in the current implementation a steady solution with the mesh computed with the new parameters is performed each time C_{D_p}, C_L, ∇C_{D_p} and ∇C_L are computed, since the ordering python recalls function and gradient computations in not known a priori but depends on the solution algorithm. This is really time expensive and useless.

11.2.2 Perpectives

A new airfoil parametrization will be required when different airfoils will be treated, especially the supercritical ones, for which the parametrization performed using two Ferguson splines proves inaccurate.

Also a smarter implementation able to save redundant steady computations will be necessary, with a check of the steady computation already performed with the current set of parameters, which would easily reduce consistently computational time.

Concerning the steady optimizer, the step which computes the sensitivity of the conservative variables with respect to the parameters $\frac{\mathrm{d}W}{\mathrm{d}p}$ will be performed externally extracting, as already done for the steady and TSM solver, the Jacobian matrix from elsA. The quantities $\frac{\partial W_b}{\partial W}$ and $\frac{\partial W_b}{\partial X}$, that are temporarily computed internally by elsA can be computed externally in the python optimizer, leaving to elsA only the final assembly of gradients.

But the ultimate aim of the whole work is the development of an unsteady optimizer: the TSM solver will be implemented in the optimizer, and will solve externally the N_p (number of parameters) sensitivity equations for a TSM problem:

$$\frac{\partial R}{\partial W}\frac{\mathrm{d}W}{\mathrm{d}p_i} = -\frac{\partial R}{\partial X}\frac{\mathrm{d}X}{\mathrm{d}p_i} - \underline{\underline{\mathbf{D_t}}}\frac{\mathrm{d}W}{\mathrm{d}p_i}, \qquad i = 1, ..., N_p \tag{11.1}$$

The unsteady optimization process will be carried out with the average of the $2N+1$ (number of instants for TSM) pressure drag coefficients over the period as objective function to minimize, and the average of the $2N+1$ lift coefficients over the period as constraint function.

The solution process of the sensitivity TSM equation will be identical to the approach adopted for the TSM mean flow solver, by substituting the unknowns and the right-hand side with their sensitivities.

A supercritical airfoil will be then optimized, implementing a RANS mean flow simulation.

Bibliography

[1] S. S. Davis, "NACA 64A010 (NASA Ames model) oscillatory pitching," *AGARD report*, vol. 702, 1982.

[2] C. Nelson, "Making Sense of CFD Grid Types." `http://www.innovative-cfd.com/cfd-grid.html`.

[3] A. Kierkegaard, *Frequency Domain Linearized Navier-Stokes Equations Methods for Low Mach Number Internal Aeroacoustics*. PhD thesis, KTH Royal Institute of Technology, 2011.

[4] M. McMullen, A. Jameson, and J. Alonso, "Acceleration of convergence to a periodic steady state in turbomachinery flows," in *39th Aerospace Sciences Meeting and Exhibit*, p. 152, 2001.

[5] K. C. Hall, J. P. Thomas, and W. S. Clark, "Computation of unsteady nonlinear flows in cascades using a harmonic balance technique," *AIAA journal*, vol. 40, no. 5, pp. 879–886, 2002.

[6] A. Gopinath and A. Jameson, "Time spectral method for periodic unsteady computations over two-and three-dimensional bodies," in *43rd AIAA aerospace sciences meeting and exhibit*, p. 1220, 2005.

[7] A. Dumont, *Calculs de gradients pour l'optimisation des performances aérodynamiques d'un rotor d'hélicoptère en vol stationnaire*. PhD thesis, Poitiers, 2010.

[8] F. Inquimbert, *Calculs d'écoulements instationnaires dans un étage de turbine transsonique en interaction rotor-stator*. PhD thesis, Université des sciences et technologies de Lille, 1994.

[9] J. Donea, A. Huerta, J. P. Ponthot, and A. Rodriguez-Ferran, "Arbitrary Lagrangian-Eulerian Methods, Vol. 1: Fundamentals, E. Stein, R.de Borst & TJR Hugues of Encyclopedia of Computational Mechanics," 2004.

[10] Y. Saad, *Iterative methods for sparse linear systems*. Society for Industrial and Applied Mathematics, 2003.

[11] G. H. Golub and C. F. Van Loan, *Matrix computations*, vol. 3. JHU Press, 2012.

[12] Y. Saad, "ILUT: A dual threshold incomplete LU factorization," *Numerical linear algebra with applications*, vol. 1, no. 4, pp. 387–402, 1994.

[13] © Copyright 2017-2018, Onera, "elsA - Theoretical Manual." `http://elsa.onera.fr/restricted/MU_MT_tuto/Doc_v3.8/STB-97020/_index.html`.

[14] P. L. Roe, "Approximate Riemann solvers, parameter vectors, and difference schemes," *Journal of computational physics*, vol. 43, no. 2, pp. 357–372, 1981.

[15] R. J. LeVeque, *Finite volume methods for hyperbolic problems*, vol. 31. Cambridge university press, 2002.

[16] P. S. Foundation, " Python Language Reference, version 2.7." `http://www.python.org`.

[17] F. Sicot, *Simulation efficace des écoulements instationaires périodiques en turbomachines*. PhD thesis, Ecully, Ecole centrale de Lyon, 2009.

[18] M. B. Giles and N. A. Pierce, "An introduction to the adjoint approach to design," *Flow, turbulence and combustion*, vol. 65, no. 3-4, pp. 393–415, 2000.

[19] S. Wright and J. Nocedal, "Numerical optimization," *Springer Science*, 1999.

[20] E. Jones, T. Oliphant, P. Peterson, *et al.*, "SciPy: Open source scientific tools for Python." `http://www.scipy.org/`.

[21] P. D. R. H. Hoppe, "Lecture notes from "Optimization theory" course," 2006.

[22] N. L. Mundis and D. J. Mavriplis, "Toward an optimal solver for time-spectral fluid-dynamic and aeroelastic solutions on unstructured meshes," *Journal of Computational Physics*, vol. 345, pp. 132–161, 2017.

[23] K. Naik, "The time-spectral method: A primer," Master's thesis, Stanford University, 2011.

[24] A. Sóbester and A. I. Forrester, *Aircraft aerodynamic design: geometry and optimization*. John Wiley & Sons, 2014.

[25] T. Achard, *Techniques de calcul de gradient aéro-structure haute-fidélité pour l'optimisation de voilures flexibles*. PhD thesis, Conservatoire national des arts et metiers-CNAM, 2017.

Appendix A

Evaluation of the time derivative

As mentioned in chapter 2, in TSM time is sampled, as $(t_0, ..., t_{2N})$ and the conservative variables, dependent on time, are sampled as well as the corresponding values to the sampled instants:

$$\mathbf{W} = (W(t_0), W(t_1), ..., W(t_{2N}))^T = (W_0, W_1, ..., W_{2N})^T \tag{A.1}$$

Naik [23] derives the formulation of the time-spectral matrix.

Reminding the Fourier Transform applied to the conservative variables as:

$$\widehat{\mathbf{W}}(k) = \frac{1}{2N+1} \sum_{j=0}^{2N} \mathbf{W}(j) e^{-\frac{2\pi i k j}{2N+1}} \tag{A.2}$$

and applying the sampling in a similar way to the Fourier coefficients:

$$\widehat{\mathbf{W}} = (\widehat{W}_{-N}, \widehat{W}_{1-N}, ..., \widehat{W}_N)^T \tag{A.3}$$

the formulation of the solution vector $\mathbf{W}$ can be developed as follows, defined over the discrete frequencies k:

$$\widehat{\mathbf{W}}(k) = [\underline{\underline{\boldsymbol{\varepsilon}}}\mathbf{W}]_k = \frac{1}{2N+1} \sum_{j=0}^{2N} \mathbf{W}(j) e^{-\frac{2\pi i k j}{2N+1}} = \frac{1}{2N+1} \sum_{j=0}^{2N} \mathbf{W}(j) e^{-\frac{2\pi i k t_j}{T}} \tag{A.4}$$

(The notation $[\,]_k$ means the *k-th* line of the product matrix-vector.)

Since derivatives need to be expressed in the time domain, it is possible to take the expression back to the time domain, by using the *Inverse Discrete Fourier Transform* $\underline{\underline{\boldsymbol{\varepsilon}}}^{-1}$ of $\widehat{\mathbf{W}}$:

$$\mathbf{W}(j) = [\underline{\underline{\boldsymbol{\varepsilon}}}^{-1}\widehat{\mathbf{W}}]_j = \sum_{k=0}^{2N} \widehat{\mathbf{W}}(k) e^{\frac{2\pi i k j}{2N+1}} = \sum_{k=-N}^{N} \widehat{\mathbf{W}}(k) e^{\frac{2\pi i k j}{2N+1}} = \sum_{k=-N}^{N} \widehat{\mathbf{W}}(k) e^{\frac{2\pi i k t_j}{T}} \tag{A.5}$$

The switch of the frequencies is done in order to satisfy the Nyquist Criterion, which states that the frequency at which the original signal is sampled, f_s, must be greater than or equal to twice the largest frequency present in the signal's spectrum, f_{max}, i.e. $f_{max} \leq \frac{f_s}{2}$.

When violating this criterion, the reconstruction of the signal will be an alias of the original.

Since $\mathbf{W}$ may be conceptualized as a set of $2N+1$ sampled values spanning a single period, the effective sampling rate is $f_s = \frac{2N+1}{1\,period}$. Applying the Nyquist criterion, the maximum

admissible discrete frequency is given by:

$$f_{max} = \frac{2N_{max}+1}{1\ period} \leq \frac{(2N+1)/(1\ period)}{2} \rightarrow f_{max} = 2N_{max}+1 \leq \frac{2N+1}{2} \tag{A.6}$$

(These frequencies are not to be confused with the frequencies of the pitching motion.)

In general, for odd $2N+1$ the Nyquist Criterion is satisfied by translating the frequency domain from $\{0,1,...,2N\}$ to $\{-N,-N+1,...,N-1,N\}$, while even case should be treated separately.

Coming back to the derivation of the time-spectral matrix, taking the *IDFT* of the *DTF* of a signal, the original signal $\mathbf{W}(l)$ is returned (the index j is replaced by the index l for the sake of clarity)

$$\underline{\underline{\boldsymbol{\varepsilon}}}^{-1}\widehat{\mathbf{W}} = \underline{\underline{\boldsymbol{\varepsilon}}}^{-1}\underline{\underline{\boldsymbol{\varepsilon}}}\mathbf{W} = \mathbf{W} \quad \longrightarrow \quad [\underline{\underline{\boldsymbol{\varepsilon}}}^{-1}\widehat{\mathbf{W}}]_j = [\underline{\underline{\boldsymbol{\varepsilon}}}^{-1}\underline{\underline{\boldsymbol{\varepsilon}}}\mathbf{W}]_l = \mathbf{W}(l) \tag{A.7}$$

By applying eqs. A.5 and A.7, the value $\mathbf{W}(l)$ at one instant t_l is now in a form that can be differentiated:

$$\mathbf{W}(l) = \sum_{k=-N}^{N} \widehat{\mathbf{W}}(k) e^{-\frac{2\pi i k t_l}{T}} \tag{A.8}$$

The time derivative, as function of time, is obtained as follows:

$$\frac{\partial \mathbf{W}(l)}{\partial t} = \frac{\partial}{\partial t}\sum_{k=-N}^{N} \widehat{\mathbf{W}}(k) e^{\frac{2\pi i k t_l}{T}} = \tag{A.9}$$

$$= \frac{2\pi}{T}\sum_{k=-N}^{N} ik\widehat{\mathbf{W}}(k) e^{\frac{2\pi i k t_l}{T}} = \tag{A.10}$$

$$= \frac{2\pi}{T}\sum_{k=-N}^{N} ik \left(\frac{1}{2N+1}\sum_{k=-N}^{N} \mathbf{W}(j) e^{-\frac{2\pi i k t_j}{T}} \right) e^{\frac{2\pi i k t_l}{T}} = \tag{A.11}$$

$$= \frac{2\pi}{T}\frac{1}{2N+1}\sum_{k=-N}^{N}\sum_{j=0}^{2N} ik\mathbf{W}(j) e^{-\frac{2\pi i k t_j}{T}} e^{\frac{2\pi i k t_l}{T}} = \tag{A.12}$$

$$= \frac{2\pi}{T}\frac{1}{2N+1}\sum_{k=-N}^{N}\sum_{j=0}^{2N} ik\mathbf{W}(j) e^{2\pi i k\left(\frac{t_l}{T}-\frac{t_j}{T}\right)} = \tag{A.13}$$

$$= \frac{2\pi}{T}\frac{1}{2N+1}\sum_{k=-N}^{N}\sum_{j=0}^{2N} ik\mathbf{W}(j) e^{\frac{2\pi i k(l-j)}{2N+1}} = \tag{A.14}$$

$$= \sum_{j=0}^{2N}\left[\frac{2\pi}{T}\frac{1}{2N+1}\sum_{k=-N}^{N} ike^{\frac{2\pi i k(l-j)}{2N+1}}\right]\mathbf{W}(j) = \tag{A.15}$$

$$= \sum_{j=0}^{2N} \underline{\underline{\mathbf{D}_t}}(l,j)\mathbf{W}(j) \tag{A.16}$$

where:

$$\underline{\underline{\mathbf{D}_t}}(l,j) = \frac{2\pi}{T}\frac{1}{2N+1}\sum_{k=-N}^{N} ike^{\frac{2\pi i k(l-j)}{2N+1}} \tag{A.17}$$

This significant result shows that the time derivative of the discrete periodic solution $\mathbf{W}$ is found by simply multiplying by an operator matrix:

$$\frac{\partial \mathbf{W}}{\partial t} = \underline{\underline{\mathbf{D}_t}}\mathbf{W} \tag{A.18}$$

The derivation of the operator matrix $\underline{\underline{\mathbf{D}_t}}$, is applicable only when the number of time instants is odd (that is why it is preferred in this discussion to define N as the number of harmonics and $2N+1$ as the number of time instances).

Renaming a group of variables in the complex exponential; $\chi = \frac{2\pi}{N}(l-j)$ the operator matrix can be simplified:

$$\underline{\underline{\mathbf{D}_t}}(l,j) = \frac{2\pi}{T}\frac{1}{2N+1}\sum_{k=-N}^{N} ike^{\frac{2\pi ik(l-j)}{2N+1}} = \tag{A.19}$$

$$= \frac{2\pi}{T}\frac{1}{2N+1}\sum_{k=-N}^{N} ike^{ik\chi} = \tag{A.20}$$

$$= \frac{2\pi}{T}\frac{1}{2N+1}\sum_{k=-N}^{N} \frac{\partial}{\partial\chi}e^{ik\chi} = \tag{A.21}$$

$$= \frac{2\pi}{T}\frac{1}{2N+1}\frac{\partial}{\partial\chi}\sum_{k=-N}^{N} e^{ik\chi} = \tag{A.22}$$

$$= \frac{2\pi}{T}\frac{1}{2N+1}\frac{\partial}{\partial\chi}\sum_{k=-N}^{N} e^{(i\chi)^k} \tag{A.23}$$

It is possible now to recognize the summation as a geometric series and solve it.

$$\sum_{k=-N}^{N}\left(e^{i\chi}\right)^k = \frac{\left(e^{i\chi}\right)^{-N}\left\{1-\left(e^{i\chi}\right)^{[N-(-N)+1]}\right\}}{1-e^{i\chi}} = \tag{A.24}$$

$$= \frac{\left(e^{i\chi}\right)^{-N}\left[1-\left(e^{i\chi}\right)^{(N+N+1)}\right]}{1-e^{i\chi}} = \tag{A.25}$$

$$= \frac{\left(e^{i\chi}\right)^{-N}\left[1-\left(e^{i\chi}\right)^{(2N+1)}\right]}{1-e^{i\chi}} = \tag{A.26}$$

$$= \frac{e^{-iN\chi}\left[1-e^{i\chi(2N+1)}\right]}{1-e^{i\chi}} = \tag{A.27}$$

$$= \frac{e^{-iN\chi}-e^{-iN\chi}e^{i(2N+1)\chi}}{1-e^{i\chi}} = \tag{A.28}$$

$$= \left(\frac{e^{-\frac{i\chi}{2}}}{e^{-\frac{i\chi}{2}}}\right)\frac{e^{-iN\chi}-e^{i(N+1)\chi}}{1-e^{i\chi}} = \tag{A.29}$$

$$= \frac{e^{-i\left(\frac{2N+1}{2}\right)\chi}-e^{i\left(\frac{2N+1}{2}\right)\chi}}{e^{-\frac{i\chi}{2}}-e^{\frac{i\chi}{2}}} = \tag{A.30}$$

$$= \frac{-2i\sin\left(\frac{2N+1}{2}\chi\right)}{-2i\sin\left(\frac{\chi}{2}\right)} = \tag{A.31}$$

$$= \frac{\sin\left(\frac{2N+1}{2}\chi\right)}{\sin\left(\frac{\chi}{2}\right)} \tag{A.32}$$

The result can be placed back into the original derivation to give:

$$\underline{\underline{\mathbf{D}_t}}(l,j) = \frac{2\pi}{T}\frac{1}{2N+1}\frac{\partial}{\partial\chi}\sum_{k=-N}^{N} e^{(i\chi)^k} = \tag{A.33}$$

$$= \frac{2\pi}{T}\frac{1}{2N+1}\frac{\partial}{\partial\chi}\frac{\sin\left(\frac{2N+1}{2}\chi\right)}{\sin\left(\frac{\chi}{2}\right)} = \tag{A.34}$$

$$= \frac{2\pi}{T}\frac{1}{2N+1}\frac{\sin\left(\frac{\chi}{2}\right)\left(\frac{2N+1}{2}\right)\cos\left(\frac{2N+1}{2}\chi\right) - \sin\left(\frac{2N+1}{2}\chi\right)\left(\frac{1}{2}\right)\cos\left(\frac{\chi}{2}\right)}{\sin^2\left(\frac{\chi}{2}\right)} \tag{A.35}$$

Some terms can be simplified independently as follows:

$$\sin\left(\frac{2N+1}{2}\chi\right) = \sin\left[\frac{2N+1}{2}\frac{2\pi}{2N+1}(l-j)\right] = \sin\left[\pi(l-j)\right] = 0 \tag{A.36}$$

$$\cos\left(\frac{2N+1}{2}\chi\right) = \cos\left[\frac{2N+1}{2}\frac{2\pi}{2N+1}(l-j)\right] = \cos\left[\pi(l-j)\right] = (-1)^{l-j} \tag{A.37}$$

because

$$\cos\left[\pi(l-j)\right] = \begin{cases} 1 & \text{if}(l-j) \text{ is even} \\ -1 & \text{if}(l-j) \text{ is odd} \end{cases} \tag{A.38}$$

These simplified expressions can be substituted back into the equation A.35:

$$\underline{\underline{\mathbf{D}_t}}(l,j) = \frac{2\pi}{T}\frac{1}{2N+1}\frac{\sin\left(\frac{\chi}{2}\right)\left(\frac{2N+1}{2}\right)\cos\left(\frac{2N+1}{2}\chi\right) - \sin\left(\frac{2N+1}{2}\chi\right)\left(\frac{1}{2}\right)\cos\left(\frac{\chi}{2}\right)}{\sin^2\left(\frac{\chi}{2}\right)} = \tag{A.39}$$

$$= \frac{2\pi}{T}\frac{1}{2N+1}\frac{\sin\left(\frac{\chi}{2}\right)\left(\frac{2N+1}{2}\right)(-1)^{l-j}}{\sin^2\left(\frac{\chi}{2}\right)} = \tag{A.40}$$

$$= \frac{2\pi}{T}\frac{1}{2}(-1)^{l-j}\frac{1}{\sin\left(\frac{\chi}{2}\right)} = \tag{A.41}$$

$$= \frac{2\pi}{T}\frac{1}{2}(-1)^{l-j}\csc\left(\frac{\chi}{2}\right) = \tag{A.42}$$

$$= \frac{2\pi}{T}\frac{1}{2}(-1)^{l-j}\csc\left[\frac{\pi}{2N+1}(l-j)\right] \tag{A.43}$$

The derivation of the time spectral operator matrix is complete and this form of the matrix is now easier to implement computationally, as an antisymmetric matrix, with the null main diagonal.

$$\underline{\underline{\mathbf{D}_t}}(l,j) = \begin{cases} \frac{2\pi}{T}\frac{1}{2}(-1)^{l-j}\csc\left[\frac{\pi}{2N+1}(l-j)\right] & \text{if } l \neq j \\ 0 & \text{if } l = j \end{cases} \tag{A.44}$$

Appendix B

Airfoil parametrization

Non-uniform rational basis spline (NURBS) is a mathematical model commonly used for generating and representing curves and surfaces. It offers great flexibility and precision for handling both analytic (surfaces defined by common mathematical formulae) and modelled shapes. The shape of the surface is determined by control points. Control points are always either connected directly to the curve/surface, or act as if they were connected by a rubber band. Depending on the type of user interface, editing can be realized via control points, which are most obvious and common for Bézier curves, or via higher level tools such as spline modelling or hierarchical editing.

An easy implementation of airfoil parametrization can be performed with two Ferguson splines, [24], whose simplicity lies in the fact that a significant amount of shape control can be exercised without having to introduce further interpolation points between the end points of the curve: it is possible here to simply adjust the tensions and the directions of the end-point tangents.

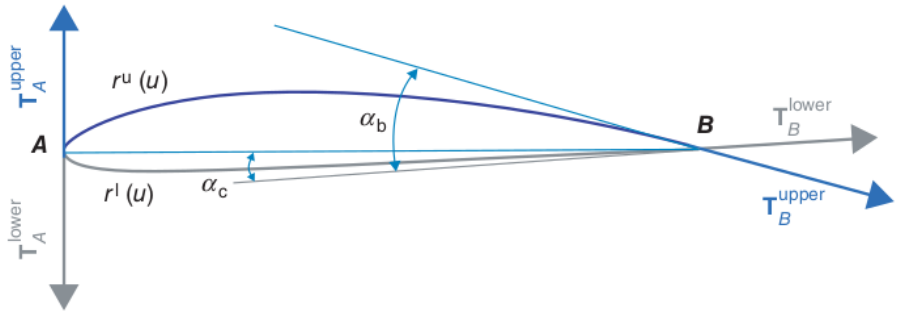


Figure B.1: Shape parameters

A Ferguson spline is a curve $\mathbf{r}(u)$ (with the parameter $u \in [0,1]$), connecting two points $\mathbf{r}(0) = \mathbf{A}$ and $\mathbf{r}(1) = \mathbf{B}$ in such a way that its tangent has a given value at these end points: $d\mathbf{r}/du|_{u=0} = \mathbf{T}_A$ and $d\mathbf{r}/du|_{u=1} = \mathbf{T}_B$. We define the curve as the cubic polynomial

$$\mathbf{r}(u) = \sum_{i=0}^{3} \mathbf{a_i} u^i, u \in [0,1] \tag{B.1}$$

We find the four sets of numbers required to define the curve by setting the endpoint conditions:

$$\mathbf{A} = \mathbf{a}_0 \tag{B.2}$$
$$\mathbf{B} = \mathbf{a}_0 + \mathbf{a}_1 + \mathbf{a}_2 + \mathbf{a}_3 \tag{B.3}$$
$$\mathbf{T}_A = \mathbf{a}_1 \tag{B.4}$$
$$\mathbf{T}_B = \mathbf{a}_1 + 2\mathbf{a}_2 + 3\mathbf{a}_3 \tag{B.5}$$

Rearranging in terms of the vectors:

$$\mathbf{a}_0 = \mathbf{A} \tag{B.6}$$
$$\mathbf{a}_1 = \mathbf{T}_A \tag{B.7}$$
$$\mathbf{a}_2 = 3[\mathbf{B} - \mathbf{A}] - 2\mathbf{T}_A - \mathbf{T}_B \tag{B.8}$$
$$\mathbf{a}_3 = 2[\mathbf{A} - \mathbf{B}] + \mathbf{T}_A + \mathbf{T}_B \tag{B.9}$$

Substituting back into B.1 we obtain:

$$\mathbf{r}(u) = \mathbf{A}(1 - 3u^2 + 2u^3) + \mathbf{B}(3u^2 - 2u^3) + \mathbf{T}_A(u - 2u^2 + u^3) + \mathbf{T}_B(-u^2 + u^3) \tag{B.10}$$

or in matrix form:

$$\mathbf{r}(u) = [1 \;\; u \;\; u^2 \;\; u^3] \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{A} \\ \mathbf{B} \\ \mathbf{T}_A \\ \mathbf{T}_B \end{pmatrix} \tag{B.11}$$

In the implementation of the parametrization, 6 parameters are given as input, T_{A_u}, T_{A_l}, T_{B_u}, T_{B_l}, α_c and α_b (it is necessary to remark the difference between the notation T_A and the bold $\mathbf{T}_A$: the first is the magnitude of the end tangent vector, so the parameter set as input, the second is the vector with the two components in x and z direction of the same end tangent vector.)

$$\mathbf{A} = [0, 0] \tag{B.12}$$
$$\mathbf{B} = [1, 0] \tag{B.13}$$
$$\mathbf{T}_{A_u} = \left[T_{A_u} \cos\left(-\frac{\pi}{2}\right), T_{A_u} \left|\sin\left(-\frac{\pi}{2}\right)\right|\right] \tag{B.14}$$
$$\mathbf{T}_{B_u} = \left[T_{B_u} \cos\left(-(\alpha_c + \alpha_b)\right), T_{B_u} \sin\left(-(\alpha_b + \alpha_c)\right)\right] \tag{B.15}$$
$$\mathbf{T}_{A_l} = \left[T_{A_l} \cos\left(-\frac{\pi}{2}\right), T_{A_l} \sin\left(-\frac{\pi}{2}\right)\right] \tag{B.16}$$
$$\mathbf{T}_{B_l} = \left[T_{b_l} \cos\left(-\alpha_c\right), T_{b_l} \sin\left(-\alpha_c\right)\right] \tag{B.17}$$

The leading edge end-point tangents must be vertical on both surfaces to ensure first-order continuity, their magnitudes can be used to control the shape of the nose of the airfoil and the boat tail angle and the camber, together with the curvatures of the rear sections of the two surfaces are controlled by the tangents at the trailing edge.

Ferguson curve parametrization scheme would be unable to reproduce airfoils with multiple inflections or relatively flat portions of either surfaces, like, for instance, supercritical airfoils. In fact, in following developments of the optimization code on a supercritical airfoil, the parametrization will change to a more flexible one.

In order to compute the sensitivities, it is sufficient to derive analytically the parameters with respect to each parameter (6 sensitivities will be obtained).

To better understand the weight that these parameters have in the shape of the airfoil, the six parameters are modified of the 50% (increased). The resulting shapes are illustrated below:

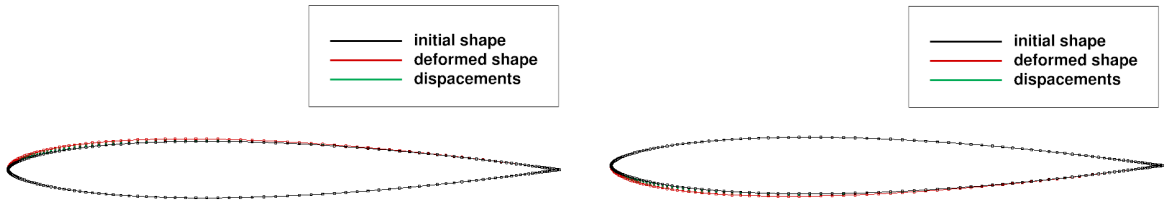


Figure B.2: T_A upper and T_A lower parameters.

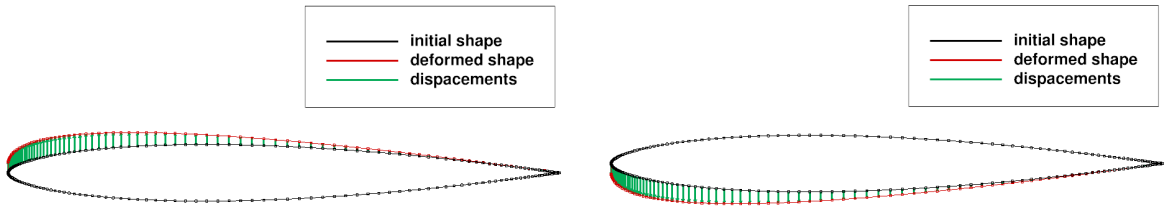


Figure B.3: T_B upper and T_B lower parameters.

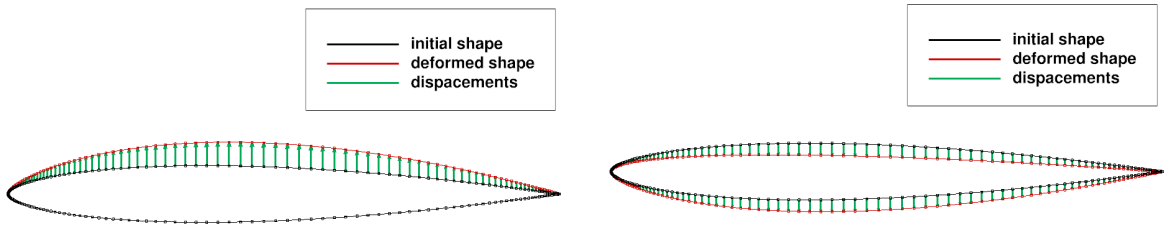


Figure B.4: α_b and α_c parameters.

At a first impression it is obvious how changes in T_{A_u} and T_{A_l} have not great influence in the shape (low sensitivities), and that changes in T_{B_u} and T_{B_l} need to act jointly to have influence on the middle and rear sections.

Appendix C

Mesh deformation

The mesh deformation method adopted in the code is based on the *Inverse Distance Weighting* method.

An analytic deformation of the points (indicated as P) of the mesh which don't belong to the aeroelastic interface Γ_w (indicated as W), that basically consists in the wall surface, or to the external boundary of the fluid domain Γ_b (indicated as B), is realized ([25]).

Knowing the surface displacements $\delta\mathbf{S}(W)$ from the NURBS module, the displacement vector $\delta\mathbf{X}(P)$ of a fluid point can be expressed as:

$$\delta\mathbf{X}(P) = \eta \frac{\displaystyle\int_{W\in\Gamma_w} \frac{1}{||\mathbf{x}(W)-\mathbf{x}(P)||^c}\delta\mathbf{S}(W)\,\mathrm{d}\mathbf{S}}{\displaystyle\int_{W\in\Gamma_w} \frac{1}{||\mathbf{x}(W)-\mathbf{x}(P)||^c}\,\mathrm{d}\mathbf{S}} = \eta \frac{\displaystyle\sum_{W\in\Gamma_w} \frac{1}{||\mathbf{x}(W)-\mathbf{x}(P)||^c}\delta\mathbf{X}(W)}{\displaystyle\sum_{W\in\Gamma_w} \frac{1}{||\mathbf{x}(W)-\mathbf{x}(P)||^c}} \tag{C.1}$$

where $||\mathbf{x}(W)-\mathbf{x}(P)||$ is the distance between surface points W and data points P, and c is a power parameter that in our code is tuned to $c = 4$.

$$\beta = 1 \tag{C.2}$$

$$\kappa = 0.15 \tag{C.3}$$

$$d_w = min(||\mathbf{x}(W)-\mathbf{x}(P)||) \tag{C.4}$$

$$d_b = min(||\mathbf{x}(B)-\mathbf{x}(P)||) \tag{C.5}$$

$\mathbf{x}(B)$ are the nodes corresponding to the boundaries, and $\mathbf{x}(W)$ are the nodes corresponding to the wall surface.

$$\eta = e^{-\kappa\left(\frac{d_w}{d_b}\right)^{\beta}} \tag{C.6}$$

η is an attenuation factor, varying from 0 to 1, and allows to ensure a physical weighting of displacements depending on the relative position of the current with respect to the aeroelastic interface and the external boundaries. d_w and d_b correspond respectively the minimal distance between the current point and the wall, and between the current point and the boundaries.

The c coefficient is chosen greater than 2. The coefficients κ and β are working parameters, tuned by the user.