

Reengineering of aerodynamic flow-field module

Tarik Chakkour

▶ **To cite this version:**

Tarik Chakkour. Reengineering of aerodynamic flow-field module. Engineering Mathematics Letters , 2018, 2018 (Article ID 4), http://scik.org/index.php/eml/article/view/3353. hal-01519521

HAL Id: hal-01519521
https://hal.science/hal-01519521

Submitted on 8 May 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Reengineering of aerodynamic flow-field module *

Tarik Chakkour

Univ. Bretagne-Sud, UMR 6205, LMBA, F-56000 Vannes, France

April 21, 2017

Abstract. In this paper, we reengineer the flow field computation module. This module generates a structured and an automatic mesh and in the next step it solves the potential equation of the flow field. The key concepts is to recall the use of the finite element method in solving Laplacian problem. We will show the feasibility of the full potential equation in the simulations of bi-dimensional steady compressible flows. Formally, we suggest an easiest version in order to generalize it in the future. This work is summarized by converting the old version which is writing in Fortran 77 to another programming language Fortran 90.

Keywords: Potential field, finite elements, software tool, Spaghetti code, mathematical model.

1 Introduction

Icing is due to the impact of supercooled water droplets on the wall of the aircraft. The supercooling of water is almost stable equilibrium. This water remains liquid when its temperature is negative. The drops that impact solidify instantly almost because of their unstable state. Frost is specific form associated to the ice which has a major disadvantage while phase of aircraft flight. Frost changes the profile plane (wings, rudders ...) and degrades the aerodynamic performance. The ice can block the rudders, measuring instruments, reduce the visibility through the windshield and it can also enter the engine. The paper [14] deals with the ice accretion on aircraft wings and with stabilizers that can cause some aerodynamic performance degradation. Weight increase, control and maneuver abilities difficulties that may reduce the aircraft operational safety margin.

The development of bi-dimensional numerical code which uses a predictor and corrector formulations is presented. At first the icing form has been predicted. After a first loop, we obtain a form of predictor of ice shape. To take into account the influence ice on the aerodynamic flow field, a new mesh is created assuming that the ice is normal to the surface of the body. A new iteration is then performed. The goal of reengineering this code (called module in the rest of the paper) is to reduce the overall time and memory required to compute physical variables (density, pressure, temperature) on the profile covered by the estimated shape.

Compressible fluid flow simulations needed for aerodynamic applications can be modeled with different degree of sophistication. The simplest model is the full potential equation which assumes inviscid, irrotational and isentropic flows. Numerical techniques for the solution of the full potential equation equation was developed respectively in the 1970s and in the 1980s [2, 11, 7, 9, 12]. Indeed, compressible flow around entire aircrafts have been simulated. For example, the full potential equation has been solved with wing, body, struts, and nacelles [15]. The main considerations addressed in this paper are the full potential. We use a fully structured finite element discretization for the full potential equation and the interface condition. 2D transonic flow simulations around airfoil are investigated. This paper first presents a short description of the module which computes the flow field around profile and generates

*This work is jointly funded by Airbus company (http://www.airbus.fr/)).

aerodynamic results. It solves the potential equation of the flow field.

It is easy to produce poor quality software in Fortran 77 than in any other language. Consequently, there is a need for software tools to aid the development of scientific and engineering applications in Fortran 90. Fortran 90 offers many significant advantages compared to previous standard Fortran 77. It brings dynamic memory management, abstract data types, data hiding, array notation and explicit subprogram interfaces. When we will use correctly these new features in the module, it allows us a better quality software and a perform application. In this paper, we investigate the software quality tools, covering Fortran 90 in order to re-engineer the module.

Since 1980, several numerical models for icing simulation were developed in the field of aeronautics. Several software packages have been designed by different groups of researchers around the world to simulate the accretion regime in local wet and dry on a wing in two dimensions (see table 1). The main objective of this paper is to take aerodynamic flow-field, explain this mathematical module, and propose a clear and easy version to be maintained in Fortran 90. The interest is to reduce the computational cost, to reduce the memory requirement and to improve the accuracy of the solution.

Name of code	Laboratory/country	year	Person developing code
Sic	Onera/France	1990	Gent
Trajice 2D	Englend	1992	Guffond
Canice	Canada	1994	Paraschivoiu
Lewice 2D	USA	1995	Wright
Cira	Italy	1996	Migione

Table 1: The numerical code developed by differents laboratories.

The organization of the rest of this paper is as follows. Section 2 describes the conservation of the potential field used the divergence equation which is discretized by the numerical method of finite element. The using of the numerical method for solving the linear system leads to accelerate fast this resolution. Section 3 deals with meshing the aerodynamic flow-field. The aim is to understand its functionality and propose in section 4 a clear and easy version in Fortran 90 to be maintained.

2 Computation of the flow field

This section is devoted to a numerical method for computing the flow field around profile. The numerical technique is based upon a general formulation for the potential field equation using Galerkin technique finite element approach. The conservation of the potential field uses the divergence equation which is discretized by the numerical method of finite element. The finite volume could be used to solve this equation, but using this method is greedy for computation time compared with the first one.

The computational domain is noted $\Omega =]a, b[\times]c, d[$ which is connected to open region in $\mathbb{R}^2$ whose boundary Γ and n is the outward pointing unit normal of surface element. Considering g the data function defined on Ω. On boundary Γ, we apply the surface flow tangency condition. Function g presents flux across the boundary Γ. The boundary conditions could be homogenous Dirichlet, Neumann type or mixt conditions depending on physical hypothesis of the problem. The full potential equation Neumann conditions is written in conservation form as:

$$\begin{cases} \operatorname{div}(\rho\nabla\phi) = 0, \text{in } \Omega \\ \rho\frac{\partial\phi}{\partial n} = g, \text{in } \Gamma \end{cases} \tag{1}$$

where, ϕ is the potential function, and ρ is the density gaz. Assuming that all the quantities are not dimensionalized by the free stream values. By appealing to the isentropic flow assumption we can write the density ρ as a nonlinear function of the potential, such as

$$\rho = \left(1 + \frac{\gamma - 1}{2} M_\infty^2 (1 - |\nabla \phi|^2)\right)^{\frac{1}{\gamma - 1}}, \tag{2}$$

where, M_∞ is the free stream Mach number and γ the ratio of specific heats. The isentropic assumption of the potential flow model leads to inaccurate physics for transonic flows with strong shocks. In what to follows, we give more details of the Finite Element Method (FEM) applied to the full Potential equation (1). There are many references [1, 10] which describe this method. The paper [1] is for FEM implementation, and the the paper [10] is for FEM theory. Let us define a space $H^1(\Omega)$ as function space where all the functions are bounded [quadratic integrable]:

$$H^1(\Omega) = \left\{ v : \Omega \to \mathbb{R} : \int_\Omega v^2, \int_\Omega v_x^2, \int_\Omega v_y^2 < \infty \right\}. \tag{3}$$

Assuming that all functions are regular, and supposing that the solution ϕ satisfies $\phi \in H^1(\Omega)$. Multiplying each terms of equation (1) by a given weight-function ψ. Next, integrating over Ω with using Green formula, we obtain:

$$\int_\Omega \rho \nabla \phi \nabla \psi = \int_\Gamma g\psi. \tag{4}$$

Indeed, the Green formula and the minimal regularity of ϕ and ψ are used in order to integrals get a sense, $\phi, \psi \in H^1(\Omega)$. For that, it is consistent to suppose that the function g is integrable over Ω. The weak form of the full potential equation is discretized by Galerkin Finite Element Method (FEM) [13, 3, 4]. Because this discretization scheme is well know. The weak formulation of problem (1) is given by:

$$\begin{gathered} \text{Find } \phi \in H^1(\Omega) \text{ such that} : \forall \psi \in H^1(\Omega) \\ \int_\Omega \rho \nabla \phi \nabla \psi = \int_\Gamma g\psi \ . \end{gathered} \tag{5}$$

Defining Hilbert spaces V_h and W_h of finite dimensional as follows:

$$V_h = \{\phi_h \in H^1(\Omega), \forall \rho_h \in L^2(\Omega), \rho_h \frac{\partial \phi_h}{\partial n} = g\}, \quad W_h = \{\psi_h \in H^1(\Omega), \psi_h = \text{Constant}\}. \tag{6}$$

In this context h is the maximum size of the cells or cells that make up the mesh. Galerkin method is applied in order to find weak formulation to approximate the defined problem in finite dimensional space. Indeed, the finite element approach consists in replacing space $H^1(\Omega)$ of test functions by subspace V_h of finite dimension

$$\begin{cases} \rho_h = \left(1 + \dfrac{\gamma - 1}{2} M_\infty^2 (1 - |\nabla \phi_h|^2)\right)^{\frac{1}{\gamma - 1}}, \text{in } \Omega \\ \displaystyle\int_\Omega \rho_h \nabla \phi_h \nabla \psi_h \, d\Omega = \int_\Gamma g\psi_h \, d\Gamma, \text{in } \Omega \end{cases} \tag{7}$$

Let us introduce Galerkin method with an abstract problem posed as a weak formulation on a Hilbert space V_h, namely,

$$\begin{gathered} \text{Find } \phi_h \in V_h \text{ such that } \forall \psi_k \in W_h \\ a(\phi_h, \psi_h) = l(\psi_h). \end{gathered} \tag{8}$$

Here, $a(.,.)$ is a bilinear form in the space $V_h \times W_h$, and l is a bounded linear functional on W_h, which are given by:

$$a(\phi_h, \psi_h) = \int_\Omega \rho_h \nabla\phi_h \nabla\psi_h \, d\Omega, \; l(\psi_h) = \int_\Gamma g\psi_h \, d\Gamma. \tag{9}$$

Let us be given two integers N_x and N_y. Assuming that the steps of mesh h_x and h_y following respectively two directions x and y are given by:

$$h_x = \frac{b-a}{N_x}, h_y \frac{d-c}{N_y}. \tag{10}$$

A rectangular mesh Q_k on Ω is defined by:

$$Q_k = \left\{ (x,y); ih_x \leq x \leq (i+1)h_x, jh_y \leq y \leq (j+1)h_y, i = 0, \ldots, N_x, j = 0, \ldots, N_y \right\}. \tag{11}$$

Denoting $N_t = (N_1 + 2) \times (N_2 + 2)$ the total number of the mesh nodes, including boundary nodes $N_b = 2(N_1 + N_2) + 4$, and internal nodes $N_I = N_1 \times N_2$. Let N_h be the number of rectangular elements, we may take as well $N_h = (N_1 + 1) \times (N_2 + 1)$. Now let us go back to weak form (8). Here we look for a function $\phi(x, y)$ which is continuously differentiable. We recall that basis functions $(f_i^h)_{1 \leq i \leq N_I}$ are all functions allowing to make an interpolation between the given points (at equal distance), and to make a piecewise continous linear polynomial. One characteristic for all these functions is the following:

$$f_i^h(x_j) = \delta_{ij}. \tag{12}$$

Every $f_i^h(x)$ is equal to 1 only at the i_{th} node and zero at all the other ones. Another characterisitc is that every $f_i^h(x)$ is only non-zero at element number i and $i + 1$, that is, elements that share node i. Note that all basis functions $(f_i^h)_{1 \leq i \leq N_I}$ are continuously differentiable. In our specific problem, we are in search of an approximate function $\phi_h(x, y)$ which is pieceswise linear on each element. As in 1D, we know that we can write it as a linear combination of basis functions $(f_i^h)_{1 \leq i \leq N_I}$:

$$\phi_h(x,y) = \sum_{i=1}^{N_I} \phi_i f_i^h(x,y). \tag{13}$$

And again in nodal basis,

$$\phi_h(x_j, y_j) = \sum_{i=1}^{N_I} \phi_i f_i^h(x_j, y_j). \tag{14}$$

Note that we are using ψ in terms of basis functions $(f_i^h)_{1 \leq i \leq N_I}$. We put these two expressions in weak form (8) and complete the expression:

$$a\left(\sum_{j=1}^{N_I} \psi_j f_j^h(x,y), \sum_{i=1}^{N_I} \phi_i f_i^h(x,y) \right) = l\left(\sum_{j=1}^{N_I} \psi_j f_j^h(x,y) \right). \tag{15}$$

By bilinearity of $a(.,.)$, and by linearity of l, we get

$$\sum_{j=1}^{N} \psi_j \sum_{i=1}^{N_I} a(f_j^h(x,y), f_i^h(x,y))\phi_i = \sum_{j=1}^{N_I} \psi_j l(f_j^h(x,y)). \tag{16}$$

Expression (16) is written in compact form as:

$$\psi^T \mathbf{A}\phi = \psi^T \mathbf{F} \Rightarrow \mathbf{A}\phi = \mathbf{F}. \tag{17}$$

Here,

$$\psi = \begin{pmatrix} \psi_1 & \psi_2 & \dots & \psi_{N_I} \end{pmatrix}^T. \tag{18}$$

$$\phi = \begin{pmatrix} \phi_1 & \phi_2 & \dots & \phi_{N_I} \end{pmatrix}^T. \tag{19}$$

$\mathbf{A}$ and $\mathbf{F}$ are given as follows:

$$\mathbf{A} = \begin{pmatrix} a(f_1^h, f_1^h) & a(f_1^h, f_2^h) & \dots & a(f_1^h, f_{N_I}^h) \\ a(f_2^h, f_1^h) & a(f_2^h, f_2^h) & \dots & a(f_2^h, f_{N_I}^h) \\ \vdots & \vdots & \vdots & \vdots \\ a(f_{N_I}^h, f_1^h) & a(f_{N_I}^h, f_2^h) & \dots & a(f_{N_I}^h, f_{N_I}^h) \end{pmatrix}. \tag{20}$$

$$\mathbf{F} = \begin{pmatrix} l(f_1^h) & l(f_2^h) & \dots & l(f_{N_I}^h) \end{pmatrix}^T. \tag{21}$$

Denoting by Q^1 the space of polynomials of partial degree less or equal to 1 from each of the variables x and y. The space Q^1 is generated by 1, x, y, xy, because it is generated by tensor products of linear functions in x and in y, which is described in terms of its canonical basis:

$$Q^1 = \{q; q(x) = a_0 + a_1 x + a_2 y + a_3 xy\}. \tag{22}$$

The numerical module uses reference elements which are elements with simple shapes and sizes. In a reference space, it is common to all elements of the same type. We will compute the four polynomials of type Q^1 associated to the unit square $\hat{Q}$ with using Lagrange condition:

$$q_i(\hat{A}_j) = \delta_{ij}. \tag{23}$$

Considering $\hat{x}$, $\hat{y}$ the coordinates of the unit square $\hat{Q}$ presented in Figure 1. Otherwise, since polynomial function q_1 is zero in both segments $[\hat{A}_3\hat{A}_4]$ and $[\hat{A}_2\hat{A}_3]$, the expression of q_1 is simplified:

$$q_1(\hat{x}, \hat{y}) = (1 - \hat{x})(1 - \hat{y}). \tag{24}$$

By the same way, the rest of polynomials are determined as follows:

$$q_j(\hat{x}, \hat{y}) = \begin{cases} \hat{x}(1 - \hat{y}), & \text{if } j = 2, \\ \hat{x}\hat{y}, & \text{if } j = 3, \\ (1 - \hat{x})\hat{y}, & \text{if } j = 4. \end{cases} \tag{25}$$

Since we have already computed the basic functions on the unit square $\hat{Q}$, we compute the local basic functions $(f_l^h)_{1 \le l \le 4}$ on each quadrangulation as follows:

$$\forall l \in \{1, \dots, 4\}, f_l^h(x, y) = q_l\left(\frac{x - x_i}{h_x}, \frac{y - y_j}{h_y}\right). \tag{26}$$

Let Q_k be an element of the mesh. There exists a unique affine bijective mapping F_k such that $F_k(\hat{Q}) = Q_k$. The affine transformation F_k permit to pass from unit square $\hat{Q}$ to square Q_k. In view of Figure 1, it is clearly enough to map the origin $\hat{A}_1(0,0)$ to point S of coordinates $(x(Q_k), y(Q_k))$, and then to multiply abscissae by h_x, and ordinates by h_y. This yields:

$$F_k \begin{pmatrix} \hat{x} \\ \hat{y} \end{pmatrix} = \begin{pmatrix} x(Q_k) + h_x \hat{x} \\ y(Q_k) + h_y \hat{y} \end{pmatrix}. \tag{27}$$

The inverse mapping is given by:

$$F_k^{-1} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{x - x(Q_k)}{h_x} \\ \frac{y - y(Q_k)}{h_y} \end{pmatrix}. \tag{28}$$

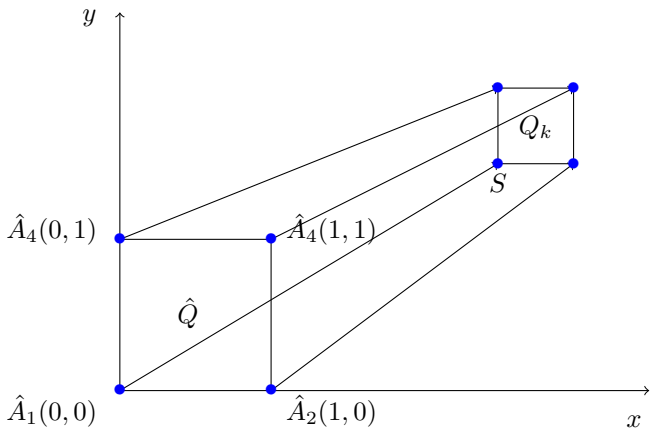


Figure 1: The affine change of variable from the reference element $\hat{Q}$ to the generic element Q_k.

The calculation of rigidity matrix $\mathbf{A}$ is done with node by node. We compute all contributions of each element that leads to similary calculation at each time. The matrix $(A_{i,j})_{\substack{1 \le i \le 4 \\ 1 \le j \le 4}}$ is defined on each grid $\Omega_{i,j}$, it is the elementary matrix 4×4, computed one time and defined with the following expression:

$$A_{i,j} = \int_{\Omega_{i,j}} \nabla f_i^h \nabla f_j^h \, d\Omega. \tag{29}$$

The rigidity matrix $\mathbf{A}$ is computed by assembly process, which is reduced to elementary contributions $A_{i,j}$ on each grid $\Omega_{i,j}$ of quadrangulation:

$$\mathbf{A}_{i,j} = \sum_{k=1}^{N_h} \rho_{i,j} A_{i,j}(Q_k), \tag{30}$$

We see that the coefficients $\mathbf{A}_{i,j}$ can thus be computed element-wise. The idea is that many of the elements $\mathbf{A}_{i,j}(Q_k)$ do not need to be computed, since it is known that they vanish as soon as the intersection of the supports of f_j^h and f_i^h does not meet Q_k. This vastly reduces the computer load. Likewise, the right-hand side of the linear system can be written as:

$$l_i = \sum_{k=1}^{N_h} l_i(Q_k). \tag{31}$$

Solve the system (8) is equivalent to solve the following linear system given by (17). The formulation of the FE approach give us a linear system of equations with dimension $N \times N$. This linear system shows that the matrix $\mathbf{A}$ is symmetric and positive definite whatever the value of the density ρ. We solve this linear system with iterative method by considering that ϕ is the limit of many sequences of ϕ_t. The convergence control is so fixed to find values of potential field ϕ at the nodes of our elliptic problem.

There are several iterative techniques designed to solve such matrices, and the method used here is method ICCG (Incomplete Cholesky Conjugate Gradient). It is based principally on preconditioning of factorisation of linear system Cholesky. The advantage to use the numerical method ICCG is to compute parameters include in the algorithm without any estimation. We use generalized mesh whose nodes are not aligned with the flow direction.

3 Meshing the physical domain

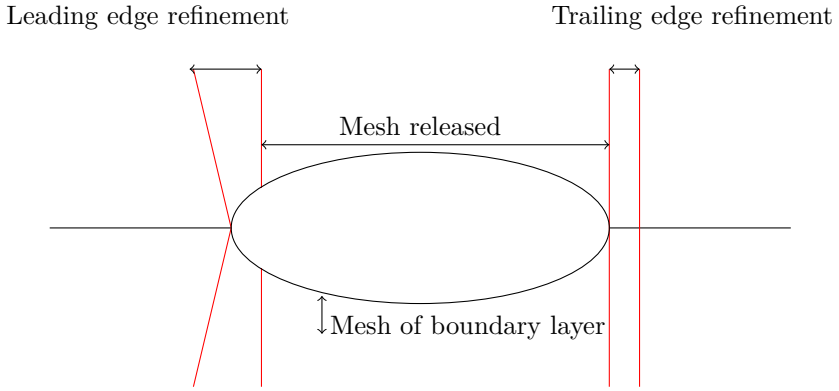


Figure 2: Identification of meshing regions.

When considering transonic flows over a wing, three regions can be identified: the boundary layer, the region around the shock, and the farfield. These regions are presented in Figure 2. The profile is extracted as a list of points. We show in Figure 3 the nodes in the standardized profile that are less than 2500. Noticing that an airfoil is divided into three main parts:

- The first part of the leading edge profile has a relatively high curvature. It is needed to have enough high discretization, it is not done only for the leading edge. However, the well discretization of break point is also necessary.

- The second part consists of the middle of the profile to the upper and lower part. We simulate flow field in transonic regime, so it must have a grid with points closer in order to capture the shock with accuracy. Indeed, the discretization being done on the skin will spread to the limits of the field.

- Finally, we arrive at the trailing edge. The mesh should tighten over the discretization in order to predict the passage of the flow in this section.

We mesh around the initial profile in order to build a new exterior mesh. From this profile, an automatic meshing is built with grid that size is $i_m \times j_m$. The new mesh is generated with form that takes C topology to surround the wing, and to better capture the wake of this mesh. The shape of the mesh was expanded from the cord and a widening of the trailing edge to infinity.

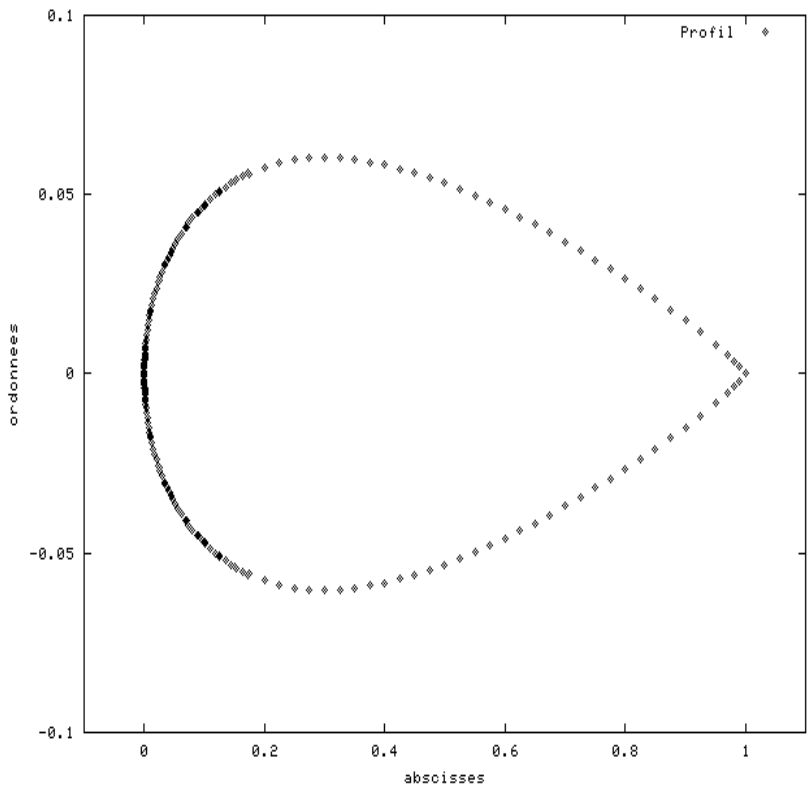


Figure 3: Profile of a given data.

The meshing zones for the two cases are presented in the following pictures of Figure 4. The flow around the profile is fairly simple. We are dealing with a 2D transonic case, thus develops a shock on the upper surface of the profile. Under these conditions, the only production of the trail is through the shock. The mesh is distinguished by two types parabolic and hyperbolic, the first is shown in the left picture of Figure 5, and the second is shown in the right picture of Figure 6. The hyperbolic mesh is very different from parabolic mesh. The first has specifically curvature almost in the form of a circle. Hyperbolic mesh is also aligned in comparison with the parabolic.

The generation of mesh meets all requirements of the flow fluid inviscid and anticipates the meshing of boundary layer. We deal the profile that has icing form. This case is very interesting because we introduce the tip form what explain the tighten discretization of some zones.

The nodes have to be distributed on perpendicular on profile. If we are placed near from the boundary, then the mesh evolves from small node near the profile to the big node. Then we need to improve the geometry in order to adapt this mesh. The node step evolves in length direction. On the contrary in width direction, it does not evolve because this boundary is considered the near part of trailing edge. Near of the trailing edge, the nodes should be small. The flare of mesh is realized in wake in order to have node near from boundary and the ratio of length and width is not large. Its target is to anticipate putting up the mesh of the boundary layer and wake in order to have no numerical stability problem.

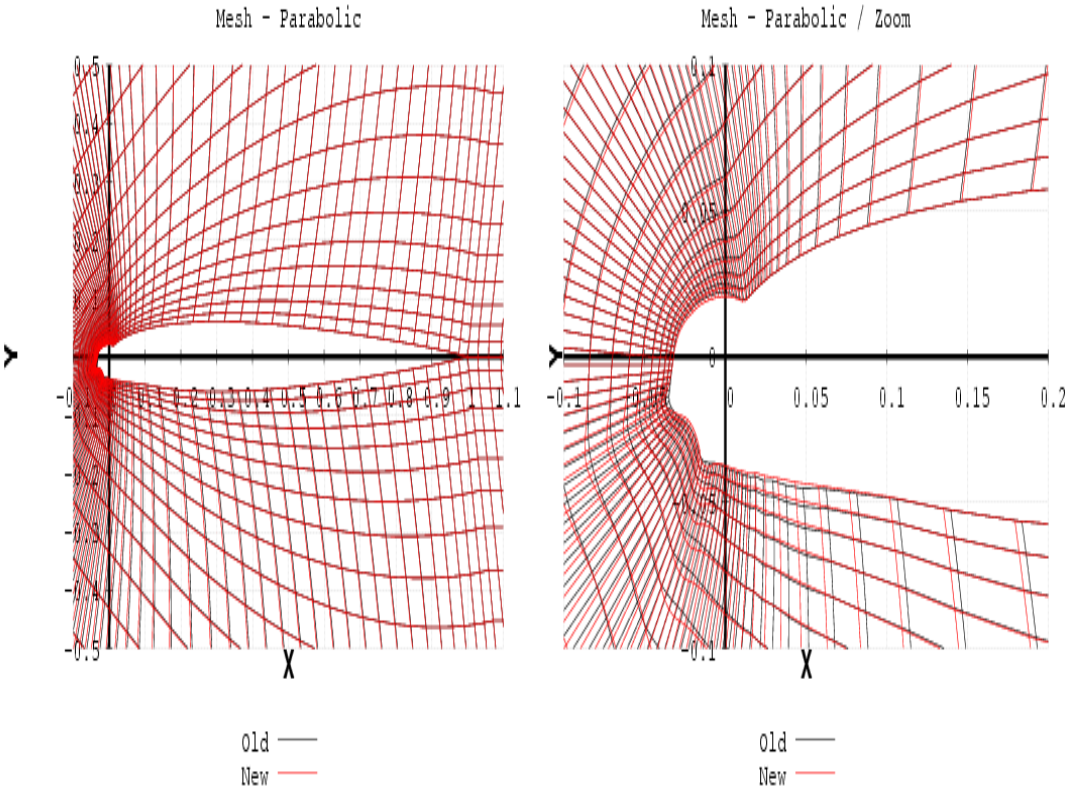


Figure 4: Meshing around profile given in Figure 3. The left picture presentes a parabolic mesh. The right picture presentes a zoom of the left picture to get a hyperbolic mesh

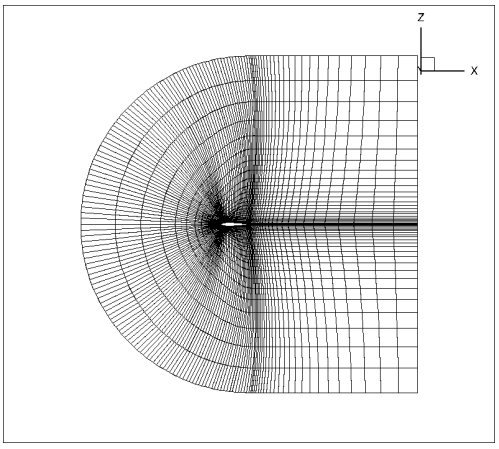


Figure 5: Parabolic mesh around the profile.

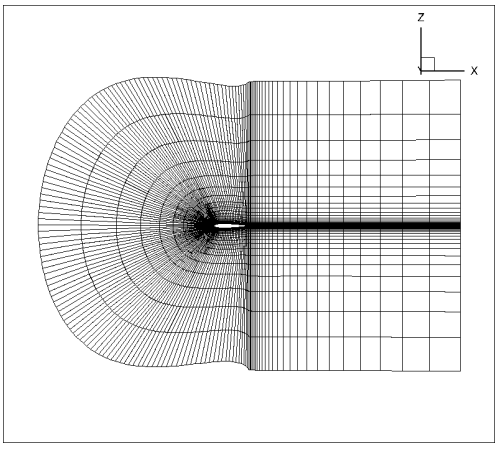


Figure 6: Hyperbolic mesh around the profile.

4 The reengineering of the flow field computation module

Much of the programming prior to 1970 was what is now considered unstructured which is a huge amount of Fortran code that is too valuable to throw away, but very expensive to maintain. It is often synonymous with the pejorative term spaghetti code, meaning source code with a complex and tangled control structure. The need for quality assurance is underlined by research such as that due to Les Hatton [5]. Hatton has analysed a large number of commercial scientific software packages written in Fortran 77. The Fortran 77 codes had an average of 12 statically detectable faults per 1000 executable statements. Another investigation of software quality in Fortran codes has been made by Hopkins [8]. He has investigated how the quality metrics of some well known public domain packages and published algorithms have changed with time.

There are many references that investigate the re-enginering physical codes in Fortran. In [6], the author proposes the use of software engineering metrics as an additional tool for the enchancement of quality in climate models.

We start to reengineer the flow field computation module by suppressing the command goto which is a statement found in many computer programming languages as like Basic and Fortran. It is a combination of the English words go and to. It performs a one-way transfer of control to another line of code; in contrast a function call normally returns control. The jumped-to locations are usually identified using labels, though some languages use line numbers. This command goto is a form of branch or jump statement. It has not been appreciated for modern programming since revolution of structured programming. In order to clarify and optimize efficiently the module, we will use classical structured programming. Consequently, we use structures such as conditional jumps (if ... then ... else) or loops (for, while, etc) in order to integrate the modern programming language.

For repeated execution of similar things, loops are used in the module with label such as do-loop presented in Listing 1. This standard loop is considered archaic in Fortran 77. They are modified to another without label depending on situation. The simple situation is given by Listing 2.

The common statement provides the combination of memory areas that can be shared by different program units (functions procedures). Command common is replaced by passing by arguments between procedures. In what to follows, we use dynamic allocation of tables. Indeed, the most of static tables are replaced with dynamic allocation to increase genericity of the module. Consequently, it fits the dimensions of any dimension mesh. In fact, the initial code could provide only meshes with predefined dimensions (and therefore not modifiable). The statements common are specification statements and have a general form defined in Listing 3. Here, blockname is a common block name. Variables var1 and var2 are a list of variable names, array names, and array declarators. To Suppress command common in this example, we use procedure such as Subroutine that name takes blockname, and arguments are var1 and var2. This example of reengineering code is presented in Listing 4.

Next, we suppress files that unnecessary output and binary temporary files. The initial code in Fortran 77 uses the large output files, which are most often unnecessary. For this reason, we are reengineering the code in order to conserve only useful files. The binary files defines the file that could be stocked all information that are presented in memory. Some variables of the module are not passed by routine's arguments. They are not saved in command common: these variables are directly in intermediate binary files. Consequently, the output files are been replaced by defining new variables with passing arguments between the routines in order to respect acutal programming standard.

All routines of reading/writing files are grouped in commune library in order to homogenize the functionality of the module. Now we justify how implicit declarations are suppressed. We declare explicitly all variables in order to prevent errors. That whay we use instruction implicit none in every blocs of declaration (see Listing 6). This instruction obligates the programmer to declare explicitly each variable.

```
1  DO 1 I1 = IV1, FV1, S1
2                  insrtuctions
3                  DO 2 I2 = IV2, FV2, S2
4                          instructions
5  2               CONTINUE
6                  DO 3 I3 = IV3, FV3, S3
7                          instructions
8  3               CONTINUE
9  1    CONTINUE
```

Listing 1: Loops with standard format in Fortran 77

```
1  DO 1 I1 = IV1, FV1, S1
2                  insrtuctions
3                  DO 2 I2 = IV2, FV2, S2
4                          instructions
5                  END DO
6                  DO 3 I3 = IV3, FV3, S3
7                          instructions
8                  END DO
9  END DO
```

Listing 2: Loops without label in Fortran 90

```
1  common/blockname/var1,var2
```

Listing 3: Implicit declarations in Fortran 77

```
1  Subroutine blockname(var1,var2)
```

Listing 4: Explicit declarations in Fortran 90

```
1  implicit real(a-h,o-z)
```

Listing 5: Implicit declarations in Fortran 77

```
1  implicit none
```

Listing 6: Explicit declarations in Fortran 90

The numerical module provides many physical results in order to give us the boundary layer properties (heat transfert coefficient and recovery temperature). These results are presented in Tecplot format. In particulary, the Figure 7 shows the density result. The red regions located at leading edge and at places near from the layer boundary have the large value of density. There exist low values of density in upper part. The average values of density are presented in green color.

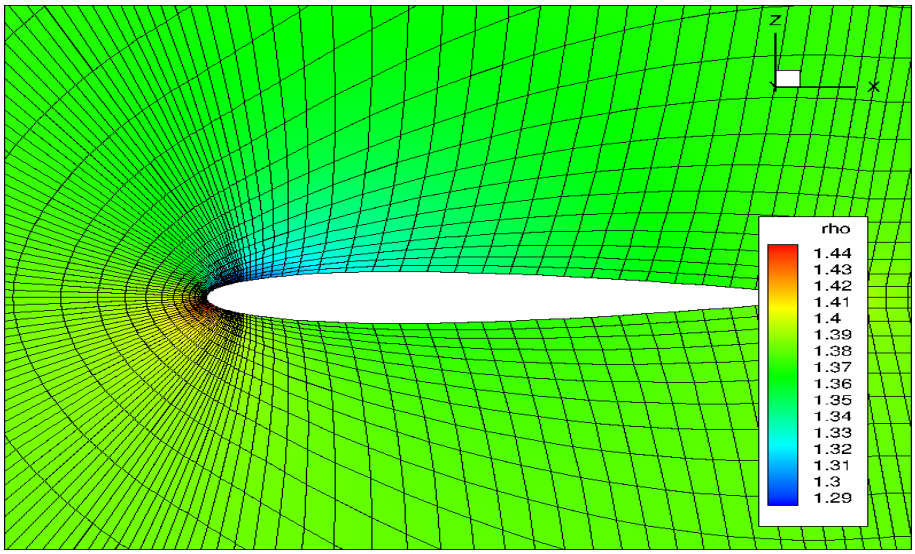


Figure 7: Density gaz around the profile.

References

[1] Klaus-Jürgen Bathe. *Finite element procedures*. Klaus-Jurgen Bathe, 2006.

[2] MO Bristeau, O Pironneau, R Glowinski, J Periaux, P Perrier, and G Poirier. On the numerical solution of nonlinear problems in fluid dynamics by least squares and finite element methods (ii). application to transonic flow simulations. *Computer Methods in Applied Mechanics and Engineering*, 51(1-3):363–394, 1985.

[3] H Deconinck and Ch Hirsch. Finite element methods for transonic blade-to-blade calculation in turbomachines. *ASME Journal of Engineering for Power*, 103(4):665–667, 1981.

[4] Wagdi G Habashi and Mohamed M Hafez. Finite element solutions of transonic flow problems. *Aiaa Journal*, 20(10):1368–1376, 1982.

[5] Les Hatton. The t-experiments: errors in scientific software. In *Quality of Numerical Software*, pages 12–31. Springer, 1997.

[6] A Henderson-Sellers, AJ Pitman, B Henderson-Sellers, D Pollard, and JM Verner. Applying software engineering metrics to land surface parameterization schemes. *Journal of climate*, 8(5):1043–1059, 1995.

[7] TL Holst, JW Slooff, H Yoshihara, WF Ballhaus Jr, and BM Spee. Applied computational transonic aerodynamics. Technical report, DTIC Document, 1982.

[8] TR Hopkins. Is the quality of numerical subroutine code improving? In *Modern software tools for scientific computing*, pages 311–324. Springer, 1997.

[9] Antony Jameson, T d Baker, and N Weatherill. Calculation of inviscid transonic flow over a complete aircraft. In *24th Aerospace Sciences Meeting*, 1986.

[10] Claes Johnson. *Numerical solution of partial differential equations by the finite element method*. Courier Corporation, 2012.

[11] J South M. Hafez and E Murman. Artificial compressibility methods for numerical solutions of transonic full potential equation. *Aiaa Journal*, 17(8):838–844, 1979.

[12] Richard B Pelz and A Jameson. Transonic flow calculations using triangular finite elements. *AIAA journal*, 23(4):569–576, 1985.

[13] Philip B Poll. *Full potential analysis and design of transonic propellers*. PhD thesis, Massachusetts Institute of Technology, 1991.

[14] Guilherme Silva, Otavio Silvares, and Euryale Zerbini. Simulation of an airfoil electro-thermal anti-ice system operating in running wet regime. In *43rd AIAA Aerospace Sciences Meeting and Exhibit*, page 1374, 2005.

[15] David P Young, Robin G Melvin, Michael B Bieterman, Forrester T Johnson, Satish S Samant, and John E Bussoletti. A locally refined rectangular grid finite element method: application to computational fluid dynamics and computational physics. *Journal of Computational Physics*, 92(1):1–66, 1991.