An Edge Based Stabilized Finite Element Method For Solving Compressible Flows: Formulation and Parallel Implementation

Azzeddine Soulaimani*
Département de Génie Mécanique, École de technologie supérieure,
1100 Notre-Dame Ouest, Montréal (Québec), H3C 1K3, Canada

Yousef Saad
Department of Computer Science and Engineering,
University of Minnesota,4-192EE/CSci Building,
200 Union Street S.E., Minneapolis, MN 55455

Ali Rebaine
Département de Génie Mécanique, École de technologie supérieure,
1100 Notre-Dame Ouest, Montréal (Québec), H3C 1K3, Canada

Abstract: This paper presents a finite element formulation for solving multidimensional compressible flows. This method is inspired by our experience with the SUPG, Finite Volume and Discontinuous-Galerkin methods. Our objective is to obtain a stable and accurate finite element formulation for multidimensional hyperbolic-parabolic problems with particular emphasis on compressible flows. In the proposed formulation, the upwinding effect is introduced by considering the flow characteristics along the normal vectors to the element interfaces. This method is applied for solving inviscid, laminar and turbulent flows. The one-equation turbulence closure model of Spalart-Allmaras is used. Several numerical tests are carried out, and a selection of two and three-dimensional experiments is presented. The results are encouraging, and it is expected that more numerical experiments and theoretical analysis will lead to greater insight into this formulation. We also discuss algorithmic and parallel implementation issues.

Key words: Finite Element Method, Compressible Flows, Upwinding, Wings, Parallel Computing, Iterative Methods.

1 Introduction

This paper discusses the numerical solution of the compressible multidimensional Navier-Stokes and Euler equations using the finite element methodology. The standard Galerkin variational formulation is known to generate numerical instabilities for convective dominated flows. Many stabilization approaches have been proposed in the literature during the last two decades, each introducing in a different way an additional dissipation to the original centered scheme. For example, a popular class of finite element methods for compressible flows is based on the Lax-Wendroff/Taylor-Galerkin scheme proposed by Donéa [1]. However, these methods experience spurious oscillations for multidimensional hyperbolic systems, so that an artificial viscosity is introduced [2]. Another class of methods is based on the SUPG (Streamline Upwinding Petrov-Galerkin) formulation introduced by Brooks-Hughes [3] and was first applied by Hughes-Tezduyar [4] to compressible flows. These schemes also suffer from spurious oscillations in high gradient zones. Later work by Hughes and his co-workers [5] improved

*Corresponding author, email: asoulaimani@mec.etsmtl.ca

its stability through the use of a new set of variables, called entropy variables, and a shock capturing operator depending on the local residual of the finite element solution. These works led naturally to the introduction of the Galerkin-Least-Squares formulation to fluid flows [6]. In the same spirit, Soulaimani-Fortin [7] developed a Petrov-Galerkin formulation which used the conservative variables and a simplified design for the shock capturing operator and for the well known stabilization matrix (or the matrix of time scales). This formulation has also been applied to other types of independent variables [9]. In [8], a SUPG formulation is used with explicit schemes and adaptive meshing. SUPG methodology is indeed commonly used in finite element based formulations while the Roe-Muscl scheme is popular in the context of finite volume based methods ([11], [12] and [10]). Recent developments of the discontinuous-Galerkin formulation try to combine the underlying ideas behind the Galerkin method and stabilized finite volume methods (see for instance [13] and [14]).
In the present study, a new stabilized finite element formulation is introduced which lies between SUPG and finite volume methods. This formulation seems to embody the good properties of both of the above methods: high order accuracy and stability in solving high speed flows. As it uses continuous finite element approximations, it is relatively easier to implement than discontinuous-Galerkin formulation, either in combination of implicit or explicit time discretizations, and requires less memory for the same order of interpolations. Preliminary numerical results were presented in [15]. Here we present further developments, particularly for turbulent flows, and more numerical experiments in 3D. In the following, the SUPG and discontinuous Galerkin methods are briefly reviewed, followed by a description of EBS formulation. The implicit solver developed is based on the nonlinear version of the Flexible GMRES. For parallel computations, a Additive-Schwarz based domain decomposition algorithm is developed. A selection of numerical results is then presented.

2 Governing equations

Let Ω be a bounded domain of R^{nd} (with $nd = 2$ or $nd = 3$) and $\Gamma = \partial\Omega$ its boundary. The outward unit vector normal to Γ is denoted by $\boldsymbol{n}$. The nondimensional Navier-Stokes equations written in terms of the conservation variables $(\rho, \boldsymbol{U}, E)$ are given by

$$\begin{aligned}
&\frac{\partial \rho}{\partial t} + \operatorname{div} \boldsymbol{U} = 0 \\
&\frac{\partial \boldsymbol{U}}{\partial t} + \operatorname{div} (\boldsymbol{U} \otimes \boldsymbol{u}) + \operatorname{grad} p = \operatorname{div} \boldsymbol{\sigma} + \rho \boldsymbol{f} \\
&\frac{\partial E}{\partial t} + \operatorname{div} ((E+p)\boldsymbol{u}) = \operatorname{div} (\boldsymbol{\sigma}.\boldsymbol{u}) - \operatorname{div} \boldsymbol{q} + \boldsymbol{f}.\boldsymbol{U} + \rho r
\end{aligned} \tag{2.1}$$

In the above equations ρ is the density, $\boldsymbol{U}$ the momentum per unit volume, $\boldsymbol{u}$ the velocity, p the pressure, $\boldsymbol{\sigma}$ the viscous-stress tensor, $\boldsymbol{q}$ the heat flux, $\boldsymbol{f}$ the body force per unit mass , r the heat source per unit mass and E the total energy per unit volume. To close the above system of equations, the supplementary constitutive relations are adopted:
$\boldsymbol{u} = \frac{\boldsymbol{U}}{\rho}$,
$T = \frac{E}{\rho} + \frac{|\boldsymbol{U}|^2}{2\rho^2}$,
$p = (\gamma - 1)\rho T$,
$\boldsymbol{q} = -\frac{\mu\gamma}{R_e Pr} \operatorname{grad} T$ and
$\boldsymbol{\sigma} = \frac{\mu}{R_e}[\operatorname{grad} \boldsymbol{u} + (\operatorname{grad} \boldsymbol{u})^t - \frac{2}{3}(\operatorname{div} \boldsymbol{u})\boldsymbol{I}]$

where R_e is the Reynolds number, $Pr = 0.72$ the Prandtl number, $\boldsymbol{I}$ the identity tensor, and μ the nondimensianal laminar viscosity. In the case of turbulent regime,

$$\boldsymbol{q} = -\frac{\gamma}{R_e}(\mu/Pr + \mu_t/Pr_t) \operatorname{grad} T$$

and

$$\boldsymbol{\sigma} = \frac{(\mu + \mu_t)}{R_e}[\text{grad}\,\boldsymbol{u} + (\text{grad}\,\boldsymbol{u})^t - \frac{2}{3}(\text{div}\,\boldsymbol{u})\boldsymbol{I}]$$

with μ_t the nondimensional turbulent viscosity and $Pr_t = 0.9$ is the turbulent Prandtl number.

2.1 Turbulence closure model

The turbulent kinematic viscosity $\nu_t = \mu_t/\rho$ is computed using the Spalart-Allmaras (S-A) one-equation model [16]. This model consists of solving only one partial differential equation over the entire fluid domain. To be accurate in solving turbulent flows, as for all other models, the S-A model requires a fine grid near the wall where the first node from the wall must guarantee a value of $y^+ \leq 10$. In these conditions, computation becomes very expensive in terms of memory requirement and CPU time. One way to avoid this problem, as well as to reduce the need for a fine grid resolving the flow in the sublayer portion, is to use a wall function to model the inner region of the boundary layer by an analytical function which is matched with the numerical solution given by the S-A model in the outer region. In this case the S-A model can be reduced to its simplified high Reynolds number version:

$$\frac{\partial \nu_t}{\partial t} + \boldsymbol{u} \cdot \nabla \nu_t - \frac{1}{Re\sigma}\left[\nabla \cdot (\nu_t \nabla \nu_t) + C_{b2}(\nabla \nu_t)^2\right] - c_{b1}\omega\nu_t + \frac{C_{w1}}{R_e} f_w \left(\frac{\nu_t}{d}\right)^2 = 0 \tag{2.2}$$

ν_t is the kinematic turbulent viscosity, ω the vorticity and d the normal distance from the wall. The closure function f_w and constants are given by:

$$f_w = g\left[\frac{1 + C_{w3}^6}{g^6 + C_{w3}^6}\right]^{1/6},$$

$$g = r + C_{w2}(r^6 - r),$$

$$r = \frac{\nu_t}{R_e \omega \kappa^2 d^2},$$

$$C_{b1} = 0.1355, C_{b2} = 0.622, \sigma = 2/3,$$

$$C_{w1} = \frac{C_{b1}}{\kappa^2} + \frac{1 + C_{b2}}{\sigma}, C_{w2} = 0.3, C_{w3} = 2$$

and $\kappa = 0.41$ is the von Karman constant. The wall function used here consists of the law of the wall developed by Spalding , which models the inner laminar sublayer, the transition region and the intermediate logarithmic layer of the turbulent boundary layer:

$$y^+ = u^+ + e^{-\kappa B}[e^{\kappa u^+} - 1 - \kappa u^+ - \frac{(\kappa u^+)^2}{2} - \frac{(\kappa u^+)^3}{6}]$$

with $y^+ = R_e \frac{\rho y u_\tau}{\mu}, u^+ = \frac{||\boldsymbol{u}||}{u_\tau}, B = 5.5$, u_τ is the friction velocity and y is the normal distance from the wall.

In order to save more memory and CPU time when using the wall function, we adopt the following technique: the computational boundary is assumed to be positioned up from the real wall by a distance δ. A slip condition with friction is then imposed: $\boldsymbol{u} \cdot \mathbf{n} = 0$. The wall traction vector $\mathbf{t}_w$ and heat flux $\mathbf{q}_w$due to shear stresses are computed as:

$$\mathbf{t}_w = -C_f \rho \boldsymbol{u}||\boldsymbol{u}||; \tag{2.3}$$

$$\mathbf{q}_w = \mathbf{t}_w \cdot \boldsymbol{u}. \tag{2.4}$$

where C_f is a friction coefficient defined using the wall function as:

$$C_f = \left[y^+ - f(u^+)\right]^{-1} \tag{2.5}$$

with

$$f(u^+) = e^{-\kappa B}[e^{\kappa u^+} - 1 - \kappa u^+ - \frac{(\kappa u^+)^2}{2} - \frac{(\kappa u^+)^3}{6}]$$

The distance δ is chosen so that any node on the solid computational boundary falls within the logarithmic layer i.e. $30 \leq y^+ \leq 100$. In this case, the destruction term $\frac{C_{w1}}{R_e} f_w \left(\frac{\nu_t}{d}\right)^2$ due to the blocking effect of the wall can be neglected. The kinematic turbulent viscosity on the wall is computed as:

$$\nu_t = R_e \; u_\tau \; \kappa \; \delta$$

with

$$u_\tau = C_f \; ||\boldsymbol{u}||.$$

To ensure a positive turbulent viscosity throughout the entire domain and during all computation iterations, a change of variable is used as $\nu_t = e^{\tilde{\nu}}$. This change of variable can be seen as a way to stabilize the numerical solution of the viscosity equation. The Spalart-Allmaras equation is then written in terms of $\tilde{\nu}$ as:

$$\frac{\partial \tilde{\nu}}{\partial t} + \boldsymbol{u} \cdot \nabla \tilde{\nu} - \frac{1}{R_e \sigma}[\nabla \cdot (e^{\tilde{\nu}} \nabla \tilde{\nu}) + (1 + C_{b2}) e^{\tilde{\nu}} (\nabla \tilde{\nu})^2] - c_{b1} \omega = 0. \tag{2.6}$$

Remark 1:

- Equation (2.5) is a typical convection-diffusion scalar equation, with nonlinear terms, for which the proposed stabilization method can be applied.

- The averaged Navier-Stokes (2.1) and the turbulence equation (2.5) are solved in a coupled way according to the following algorithm:

1- Initialize the flow field and the turbulent viscosity with a given distance δ.
2- Loop over time steps.
3- Loop over Newton iterations.
4- Solve the wall function for u_τ.
5- Update ν_t and C_f on the wall.
6- Solve the coupled system of equations (N-S and S-A).
7- With the new solution repeat the algorithm from step 3 until convergence.
8- End of Newton iterations.
9- End of time advancing loop.

Remark 2:

Equations (1) and (4) can also be rewritten in terms of the vector $\boldsymbol{V} = (\rho, \boldsymbol{U}, E, \tilde{\nu}_t)^t$ in a compact and generic form as

$$\boldsymbol{V}_{,t} + \boldsymbol{F}^{adv}_{i,i}(\boldsymbol{V}) = \boldsymbol{F}^{diff}_{i,i}(\boldsymbol{V}) + \boldsymbol{\mathcal{F}} \tag{2.7}$$

where $\boldsymbol{F}^{adv}_i$ and $\boldsymbol{F}^{diff}_i$ are respectively the convective and diffusive fluxes in the ith-space direction, and $\boldsymbol{\mathcal{F}}$ is the source vector. Lower commas denote partial differentiation and repeated indices indicate summation. The diffusive fluxes can be written in the form:

$$\boldsymbol{F}^{diff}_i = \boldsymbol{K}_{ij} \boldsymbol{V}_{,j}$$

while the convective fluxes can be represented by diagonalizable Jacobian matrices $\boldsymbol{A}_i = \boldsymbol{F}^{adv}{}_{i,V}$. Note that any linear combination of these matrices has real eigenvalues and a complete set of eigenvectors.

3 Stabilization techniques

Throughout this paper, we consider a partition of the domain Ω into elements Ω^e where piecewise continuous approximations for the conservative variables are adopted. It is well known that the standard Galerkin finite element formulation often leads to numerical instabilities for convective dominated flows. Various stabilization finite element formulations have been proposed in the last two decades. Most of them can be cast in the generic form: find $\boldsymbol{V}$ such that for all weighting functions $\boldsymbol{W}$,

$$\sum_e \int_{\Omega^e} [\boldsymbol{W} \cdot (\boldsymbol{V}_{,t} + \boldsymbol{F}^{adv}_{i,i} - \boldsymbol{\mathcal{F}}) + \boldsymbol{W}_{,i} \boldsymbol{F}^{diff}_i] \, d\Omega \; - \int_\Gamma \boldsymbol{W}.\boldsymbol{F}^{diff}_i \; n_i \, d\Gamma \tag{3.1}$$

$$+ \sum_e \int_{\Omega^e} S(\boldsymbol{W}, \boldsymbol{V}) \, d\Omega = 0.$$

where $\boldsymbol{n}$ is the outward unit normal vector to the boundary Γ and $S(\boldsymbol{W}, \boldsymbol{V})$ is a bilinear form to add more stability to the Galerkin integral form. Note that $S(\boldsymbol{W}, \boldsymbol{V})$ is defined and intergated over elements interior. In its simplest and popular expressions, $S(\boldsymbol{W}, \boldsymbol{V})$ reduces for the one-dimensional system case to:

$$S(\boldsymbol{W}, \boldsymbol{V}) = \boldsymbol{W}_{,1} \frac{h}{2} |\boldsymbol{A}| \boldsymbol{V}_{,1} = \boldsymbol{A}^t \boldsymbol{W}_{,1} \boldsymbol{\tau} \boldsymbol{A} \boldsymbol{V}_{,1}$$

with h the element length and the matrix $\boldsymbol{\tau} = \frac{h}{2} |\boldsymbol{A}|^{-1}$. It is clear that $S(\boldsymbol{W}, \boldsymbol{V})$ is positive for symmetric $\boldsymbol{A}$. It can also be proven that $S(\boldsymbol{W}, \boldsymbol{V})$ is positive for $\boldsymbol{A}$ derived from Euler flux [17]. Thus, the obtained stabilized method is nothing but the classical first order upwinding scheme applied along flow characteristics. For the Navier-Stokes equations in multidimensions, there is an infinite number of flow characteristics inside any elements. However, for a prescribed space direction, there are only a finite number of characteristics along which upwinding techniques can, in principle, be applied.

3.1 SUPG formulation

In the SUPG method, the Galerkin variational formulation is modified to include an integral form depending on the local residual $\boldsymbol{\mathcal{R}}(\boldsymbol{V})$ of equation (2.7), i.e. $\boldsymbol{\mathcal{R}}(\boldsymbol{V}) = \boldsymbol{V}_{,t} + \boldsymbol{F}^{adv}_{i,i}(\boldsymbol{V}) - \boldsymbol{F}^{diff}_{i,i}(V) - \boldsymbol{\mathcal{F}}$, which is identically equal to zero for the exact solution. The SUPG formulation reads then as : find $\boldsymbol{V}$ such that for all weighting functions $\boldsymbol{W}$,

$$\sum_e \int_{\Omega^e} [\boldsymbol{W} \cdot (\boldsymbol{V}_{,t} + \boldsymbol{F}^{adv}_{i,i} - \boldsymbol{\mathcal{F}}) + \boldsymbol{W}_{,i} \boldsymbol{F}^{diff}_i] \, d\Omega \; - \int_\Gamma \boldsymbol{W}.\boldsymbol{F}^{diff}_i \; n_i \, d\Gamma \tag{3.2}$$

$$+ \sum_e \int_{\Omega^e} (\boldsymbol{A}^t_i \boldsymbol{W}_{,i}) \cdot \boldsymbol{\tau} \cdot \boldsymbol{\mathcal{R}}(\boldsymbol{V}) \, d\Omega = 0.$$

In this case, $S(\boldsymbol{W}, \boldsymbol{V}) = (\boldsymbol{A}^t_i \boldsymbol{W}_{,i}) \cdot \boldsymbol{\tau} \cdot \boldsymbol{\mathcal{R}}(\boldsymbol{V})$, and the matrix $\boldsymbol{\tau}$ is commonly referred to as *the matrix of time scales.* The SUPG formulation is built as a combination of the standard Galerkin integral form and a perturbation-like integral form depending on the local residual vector. The objective is to reinforce the stability *inside the elements.* The SUPG formulation involves two important ingredients: First, it is a residual method in the sense that the exact continuous regular solution of the original physical problem is still a solution of the variational problem (3.1). This is a requirement for optimal accuracy. Second, it contains the following integral term: $\sum (\int_{\Omega^e} (\boldsymbol{A}^t{}_i . \boldsymbol{W}_{,i}) \boldsymbol{\tau} (\boldsymbol{A}_j \boldsymbol{V}_{,j}) d\Omega)$, which is of elliptic type provided that the matrix $\boldsymbol{\tau}$ is appropriately designed. However, for multidimensional systems, it is difficult to define $\boldsymbol{\tau}$ in such a way as to introduce the additional stability in the characteristic directions. This property is desired to reduce artificial cross-wind diffusion. Indeed, for multidimensional Navier-Stokes, the convection matrices are not simultaneously diagonalizable. A choice of $\boldsymbol{\tau}$ matrix proposed in [17] reads $\boldsymbol{\tau} = (\boldsymbol{B}_i \boldsymbol{B}_i)^{-1/2}$, where $\boldsymbol{B}_i = \frac{\partial \eta_i}{\partial x_j} \boldsymbol{A}_j$ and $\frac{\partial \eta_i}{\partial x_j}$ are

the components of the element Jacobian matrix. Thus, $\boldsymbol{\tau}$ is defined using a combination of advection matrices computed in the local element frame. In [7] a simplified formula is proposed to analytically compute $\boldsymbol{\tau}$ as

$$\boldsymbol{\tau} = (\sum_i |\boldsymbol{B}_i|)^{-1}.$$

The above expressions of $\boldsymbol{\tau}$ reproduce exactly the one-dimensional case.

3.2 Discontinuous Galerkin method

The discontinuous-Galerkin (DG) method is usually applied to purely hyperbolic PDEs. It is obtained by applying the standard Galerkin method to each element. That is, a finite-dimensional basis set is selected for each element, the solution in each element is approximated in terms of an expansion on that basis, and the governing equations are then interpreted in a weak form as

$$\int_{\Omega^e} \boldsymbol{W} \cdot \boldsymbol{V}_{,t}\, d\Omega \; - \int_{\Omega^e} \boldsymbol{W}_{,i} \cdot \boldsymbol{F}_i^{adv}\, d\Omega \; + \int_{\Gamma^e} \boldsymbol{W} \cdot \boldsymbol{F}_i^{adv,R}(\boldsymbol{V}, \boldsymbol{V}^{'})\, n_i\, d\Gamma = 0 \tag{3.3}$$

where $\boldsymbol{n}$ is the outward unit normal vector to Γ^e, $\boldsymbol{V}$ is the approximate solution in element Ω^e, and $\boldsymbol{V}^{'}$ denotes the approximate solution in the neighboring elements to Ω^e and computed on the element boundary Γ^e. Because the global solution is discontinuous across element interfaces, the discontinuities are resolved through the use of approximate Riemann flux vector $\boldsymbol{F}_i^{adv,R}(\boldsymbol{V}, \boldsymbol{V}^{'})\, n_i$. This flux provides the *upwinding effect* that is required to ensure stability. For instance, Roe type approximate Riemann flux is given by

$$\boldsymbol{F}_i^{adv,R}(\boldsymbol{V}, \boldsymbol{V}^{'})\, n_i = \frac{(\boldsymbol{F}_i^{adv}(\boldsymbol{V}) + \boldsymbol{F}_i^{adv}(\boldsymbol{V})^{'})\, n_i}{2} - |\tilde{\boldsymbol{A}}_n| \frac{(\boldsymbol{V} - \boldsymbol{V}^{'})}{2}$$

where $\boldsymbol{A}_n = n_i \boldsymbol{A}_i$ and $\tilde{\boldsymbol{A}}_n$ is the well-known Roe matrix. Taking an approximation of order zero for the weighting $\boldsymbol{W}$ and trial functions, i.e. a constant for each element, one can recover the classical cell-centered finite volume Roe scheme. For higher order approximations, the number of degrees of freedom can however increase rapidly which may be a serious drawback. The higher-order DG method may also be somewhat complex to implement for implicit time schemes. The first term in the above approximate Riemann flux generates a centered scheme, while the last term introduces the upwind bias in the flow characteristics and along *the normal direction to the element boundary*. In the above scheme, stability is introduced by resolving the discontinuity in this direction of the solution field $\boldsymbol{V}$. This stabilization approach is also known as stabilization or upwinding by discontinuity. Note that cell centered finite volume (FV) schemes can be derived from (3.3) by choosing zero order weighting functions. It is believed that the success of FV schemes in solving high speed flows is related to the fact that artificial dissipation is primarily introduced along the flow characteristics which are computed along the normal directions to the element edges (or faces in 3D). It is desirable to keep this property in the framework of the finite element methodology using simple continuous interpolations.

4 The Edge Based Stabilized method (EBS)

Let us first take another look at the SUPG formulation. Using integration by parts in (3.1), the integral

$$\int_{\Omega^e} (\boldsymbol{A}_i^t \boldsymbol{W}_{,i}) \cdot \boldsymbol{\tau} \cdot \boldsymbol{\mathcal{R}}(\boldsymbol{V})\, d\Omega$$

can be transformed into

$$\int_{\Gamma^e} \boldsymbol{W} \cdot (\boldsymbol{A}_n \boldsymbol{\tau} \cdot \boldsymbol{\mathcal{R}}(\boldsymbol{V}))\, d\Gamma - \int_{\Omega^e} \boldsymbol{W} \cdot (\boldsymbol{A}_i\, \boldsymbol{\tau} \cdot \boldsymbol{\mathcal{R}}(\boldsymbol{V}))_{,i}\, d\Omega \tag{4.1}$$

where Γ^e is the boundary of the element Ω^e, $\boldsymbol{n}^e$ the outward unit normal vector to Γ^e and $\boldsymbol{A}_n = n^e{}_i \boldsymbol{A}_i$. If one neglects the second integral above, then

$$\sum_e \int_{\Omega^e} (\boldsymbol{A}_i^t \boldsymbol{W}_{,i}) \cdot \boldsymbol{\tau} \cdot \boldsymbol{\mathcal{R}}(\boldsymbol{V}) \cong \sum_e \int_{\Gamma^e} \boldsymbol{W} \cdot (\boldsymbol{A}_n\, \boldsymbol{\tau} \cdot \boldsymbol{\mathcal{R}}(\boldsymbol{V})\, d\Gamma.$$

The above equation suggests that $\boldsymbol{\tau}$ could be defined explicitly only at the element boundary. Since in practice a numerical integration is usually employed, it is then sufficient to compute $\boldsymbol{\tau}$ at a few Gauss points on Γ^e. Hence, a natural choice for $\boldsymbol{\tau}$ is given by

$$\boldsymbol{\tau} = \frac{h}{2}|\boldsymbol{A}_n|^{-1} \tag{4.2}$$

Since the characteristic lines are well defined on Γ^e for the given direction $\boldsymbol{n}^e$, then the above definition of $\boldsymbol{\tau}$ is *not completely arbitrary*. It defines $\boldsymbol{\tau}$ using the eigenvalues of $\boldsymbol{A}_n$. Using (4.2), the stabilizing contour integral term in (4.1) becomes

$$\sum_e \int_{\Gamma^e} \frac{h}{2} \boldsymbol{W} \cdot (sign(\boldsymbol{A}_n) \cdot \boldsymbol{\mathcal{R}}(\boldsymbol{V}))\, d\Gamma.$$

For a one dimensional hyperbolic system, one can recognize the upwinding effect introduced by the EBS added term

$$\sum_e \int_{\Gamma^e} \frac{h}{2} \boldsymbol{W} \cdot (|\boldsymbol{A}_n| \boldsymbol{V}_{,1})\, d\Gamma.$$

which has a strong similarity with $\sum_e \int_{\Omega^e} \boldsymbol{W}_{,1} \frac{h}{2} |\boldsymbol{A}| \boldsymbol{V}_{,1} d\Omega$.

Remarks 3:

- Equation (4.2) provides an appropriate definition of the matrix of time scales for multidimensional systems. Thus, the standard SUPG formulation can still be used along with the new design of $\boldsymbol{\tau}$ as given in (4.2). However, to easily compute the stabilizing integral term, integration points have to be chosen on the element contour.

- For linear finite element interpolations, the first integral term in (4.1) introduces the most dominant stabilizing effect. The resulting Edge Based Stabilized finite element formulation clearly shares some roots with SUPG and DG methods.

Here we would like to show how more upwinding effect can naturally be introduced in the framework of EBS formulation. Consider the eigen-decomposition of $\boldsymbol{A}_n$,

$$\boldsymbol{A}_n = S_n \Lambda_n {S_n}^{-1}.$$

Let $Pe_i = \lambda_i h / 2\nu$ be the local Peclet number for the eigenvalue λ_i, h a measure of the element size on the element boundary, ν the physical viscosity, $\beta_i = min(Pe_i/3, 1.0)\beta$ and $0 \leq \beta \leq 1$ a positive parameter. We define the matrix $\boldsymbol{B}_n$ by

$$\boldsymbol{B}_n = S_n L {S_n}^{-1} \tag{4.3}$$

where L is a diagonal matrix whose entries are given by $L_i = (1 + \beta_i)$ if $\lambda_i > 0$; $L_i = -(1 - \beta_i)$ if $\lambda_i < 0$ and $L_i = 0$ if $\lambda_i = 0$. This means that more weight is introduced for the upwind element. The proposed EBS formulation can now be summarized as follows: Find $\boldsymbol{V}$ such that for all weighting functions $\boldsymbol{W}$,

$$\sum_e \int_{\Omega^e} [\boldsymbol{W} \cdot (\boldsymbol{V}_{,t} + \boldsymbol{F}^{adv}_{i,i} - \boldsymbol{\mathcal{F}}) + \boldsymbol{W}_{,i} \boldsymbol{F}^{diff}_i]\, d\Omega - \int_\Gamma \boldsymbol{W}.\boldsymbol{F}^{diff}_i n_i\, d\Gamma \tag{4.4}$$

$$+ \sum_e \int_{\Gamma^e} \boldsymbol{W} \cdot \boldsymbol{\tau}^{ed}_n \cdot \boldsymbol{\mathcal{R}}(\boldsymbol{V})\, d\Gamma = 0$$

with $\boldsymbol{\tau}^{ed}_n$ the matrix of intrinsic length scales given by

$$\boldsymbol{\tau}^{ed}_n = \frac{h}{2} \cdot \boldsymbol{B}_n. \tag{4.5}$$

We now point out the following important remarks:

Remarks 4:

- As in the case of SUPG method, EBS formulation is a residual method in the sense that if the exact solution is sufficiently regular then it is also a solution of (2.7). Thus, one may expect high-order accuracy. Note that the only assumption made on the finite element approximations for the trial and weighting functions is that they are piecewise continuous. Equal-order or mixed approximations can in principle be employed. Further theoretical analysis is required to give a clearer answer.

- A stabilization effect is introduced by computing the difference between the residuals on element interfaces while considering the direction of the characteristics. In this approach, the stabilization is introduced using *the jumps of the residuals across element boundaries*. A higher jump is in fact an indication of an irregular solution or of a mesh-related difficulty in solving the PDE.

- For a purely hyperbolic scalar problem, one can see some analogy between the proposed formulation and the discontinuous-Galerkin method and also with the finite volume formulation.

- The function of the EBS formulation is to add an amount of artificial viscosity in the characteristic directions. Since EBS formulation leads to a high-order scheme, high frequency oscillations in the vicinity of shocks or stagnation points can occur. A shock capturing viscosity depending on the discrete residual $\boldsymbol{\mathcal{R}}(\boldsymbol{V})$ is used. More dissipation is then added in high gradient zones to avoid any undesirable local oscillations. Two formulations for the shock capturing viscosity were used. The first one is identical to that proposed in [7], $\mu_{cc1} = C_{k1}\ h\ min(||\tau\boldsymbol{\mathcal{R}}(\boldsymbol{V})||, ||\boldsymbol{u}||)/2$ with C_{k1} a tuning parameter. The second formulation is obtained by multiplying the artificial viscoisty ν_d proposed in [18] by a tuning paremeter C_{k2}, $\mu_{cc2} = C_{k2}\ \nu_d$. From extensive tests, we observed that when μ_{cc2} is used with $C_{k2} = 1.$, an excessively smeared solution is obtained. Better results are given with a smaller value up to 0.25. The parameter C_{k1} usually takes a value between 1. and 10. depending on the grid resoution. As a general observation, EBS formulation behaves better with μ_{cc2} than with μ_{cc1}. We usullay start the computations with a higher value of C_{k1} or C_{k2} and we gradually decrease it as long as the convergence is guaranteed.

- The parameter β_i is introduced to give more weight to the element situated in the upwind characteristic direction. The formulation given above for the parameter β_i is introduced in order to make it vanish rapidly in regions dominated by the physical diffusion such as the boundary layers. It is also possible to choose β_i as a function of the local Mach number Ma, for instance by choosing $\beta = Ma$ by analogy with finite volume schemes [13].

- The length scale h introduced above is computed as the distance between the centroid of the element and its edges (faces in 3D).

- Numerical experiments showed that a higher order integration quadrature should be used to evaluate the element-contour integrals. For instance, in the case of 3D computations using tetraedral elements, stable and accurate results have been obtained using three Gauss points for each face.

4.1 An illustrative example

The Edge Based Stabilized finite element formulation (4.4) reads in the case of the multidimensional scalar advection-diffusion equation (2.6) as:

$$\sum_e \int_{\Omega^e} [\boldsymbol{W} \cdot (\frac{\partial \tilde{\nu}}{\partial t} + \boldsymbol{u} \cdot \nabla \tilde{\nu} - \frac{1}{R_e \sigma}((1 + C_{b2}) e^{\bar{\nu}} (\nabla \tilde{\nu})^2 - c_{b1}\omega) + \frac{1}{R_e \sigma} \nabla \boldsymbol{W} \cdot (e^{\bar{\nu}} \nabla \tilde{\nu})] d\Omega$$

$$+ \sum_e \int_{\Gamma^e} [\boldsymbol{W} \cdot \boldsymbol{\tau}_n^{ed} \cdot (\frac{\partial \tilde{\nu}}{\partial t} + \boldsymbol{u} \cdot \nabla \tilde{\nu} - \frac{1}{R_e \sigma}(\nabla \cdot (e^{\bar{\nu}} \nabla \tilde{\nu}) + (1 + C_{b2}) e^{\bar{\nu}} (\nabla \tilde{\nu})^2) - c_{b1}\omega)] d\Gamma$$

with $\boldsymbol{\tau}_n^{ed}$ a scalar having a dimension of length. It reads in its simplest expression as $\boldsymbol{\tau}_n^{ed} = \frac{h}{2} \cdot \boldsymbol{B}_n$ with $\boldsymbol{B}_n = (1 + \beta)$ if $\boldsymbol{u} \cdot \boldsymbol{n}^e > 0$ and $\boldsymbol{B}_n = -(1 - \beta)$ if $\boldsymbol{u} \cdot \boldsymbol{n}^e < 0$. Note that the residual of equation (2.6) is weighted

and integrated over the element boundary. The difference of two gradients $\boldsymbol{u} \cdot \nabla \tilde{\nu}$ computed along an edge (or a face) of two adjacent elements results in a discrete Laplacian operator along the streamline, thus introducing a stablization effect. It is worth noting that the time derivative term in the contour integral has a beneficial effect on the conditioning of the global system.

5 Solution algorithms

An implicit solution algorithm is used based on a time marching procedure combined with an inexact-Newton algorithm and variants of GMRES [19]. Stabilization methods naturally introduce more nonlinearities in the original PDEs equations. These nonlinearities could be very strong so that robust solution methods are required. Specifically, it was observed that in the case of turbulent flows [20], standard preconditioned GMRES algorithm failed to converge in some situations, especially for turbulent flows. However, convergence was achieved by using ILUT preconditioner with the Flexible GMRES. This robust combination of techniques has also been successful with Additive Schwarz domain decomposition method, leading to a fairly efficient parallel version of the code, as will be seen later. The proposed preconditioners are briefly presented in this section. More details are given in the references. We begin with a review of the right-preconditioned GMRES algorithm [19] described here for solving a linear system of the form

$$Ax = b$$

where A is the coefficient matrix and b the right-hand-side. The algorithm requires a preconditioning matrix M in addition to the original matrix A. The preconditioner M is typically a certain matrix that is close to A but is easily invertible in the sense that solving linear systems with it is inexpensive. The algorithm also requires an initial guess x_0 to the solution.

ALGORITHM **5.1** *Right preconditioned GMRES*

1. *Compute* $r_0 = b - Ax_0$, $\beta := ||r_0||_2$, $v_1 := r_0/\beta$.
2. *Define the* $(m+1) \times m$ *matrix* $\overline{H}_m =$
   $\{h_{i,j}\}_{1 \le i \le m+1, 1 \le j \le m}$. *Set* $\overline{H}_m = 0$.
3. *For* $j = 1, 2, ..., m$ *Do:*
4. *Compute* $z_j := M^{-1} v_j$
5. *Compute* $w_j := A z_j$
6. *For* $i = 1, ..., j$ *Do:*
7. $h_{ij} := (w_j, v_i)$
8. $w_j := w_j - h_{ij} v_i$
9. *EndDo*
10. $h_{j+1,j} = ||w_j||_2$. *If* $h_{j+1,j} = 0$ *set* $m := j$, *goto* 13
11. $v_{j+1} = w_j / h_{j+1,j}$
12. *EndDo*
13. *Define* $V_m = [v_1, ..., v_m]$, *and* $H_m = \{h_{i,j}\}_{1 \le i \le m+1, 1 \le j \le m}$.
14. *Compute* y_m *the minimizer of*
    $||\beta e_1 - \overline{H}_m y||_2$ *and* $x_m = x_0 + V_m y_m$.
15. *If satisfied Stop, else set* $x_0 := x_m$ *goto* 1.

We now make a few comments on the nonlinear version of GMRES. When the above algorithm is used in the context of Newton's method, the matrix A represents the Jacobian matrix of a certain nonlinear function F, which in our case is the discretized version of the Navier-Stokes equations. However, it is not always possible, or it may simply be very expensive, to compute the Jacobian matrix analytically. It may be preferable to compute an approximation of the Jacobian and freeze it for a prescribed number of time steps and Newton iterations. This matrix will then be used to construct the preconditioning matrix M. On the other hand, the action of the Jacobian on a vector (i.e. the matrix-by-vector product in line 5 of the above algorithm) can be approximated using a finite difference quotient such as:

$$\frac{F(x_0 + \epsilon z_j) - F(x_0)}{\epsilon} \ , . \tag{5.1}$$

where ϵ is an appropriate small number. All the problems considered in this paper are solved using this *non-linear version* of GMRES or its variant FGMRES. This general approach is also termed 'inexact Newton' method, since the linear system in Newton's method is solved approximately, or *inexactly.*

5.1 Incomplete LU factorizations and ILUT

One of the most common ways to define the preconditioning matrix M is through Incomplete LU factorizations. In essence, an ILU factorization is simply an approximate Gaussian elimination. When Gaussian Elimination is applied to a sparse matrix A, a large number of nonzero elements may appear in locations originally occupied by zero elements. These fill-ins are often small elements and may be dropped to obtain Incomplete LU factorizations. So ILU is in essence a Gaussian Elimination procedure in which fill-ins are dropped.

The simplest of these procedures is ILU(0) which is obtained by performing the standard LU factorization of A and dropping all fill-in elements that are generated during the process. In other words, the L and U factors have the same pattern as the lower and upper triangular parts of A (respectively). More accurate factorizations denoted by ILU(k) and IC (k) have been defined which drop fill-ins according to their 'levels'. Level-1 fill-ins for example are generated from level-zero fill-ins (at most). So, for example, ILU(1) consists of keeping all fill-ins that have level zero or one and dropping any fill-in whose level is higher.

Another class of preconditioners is based on dropping fill-ins according to their numerical values. One of these methods is ILUT (ILU with Threshold). This procedure uses basically a form of Gaussian elimination which generates the rows of L and U one by one. Small values are droped during the elimination using a parameter τ. A second parameter, p, is then used to keep the largest p entries in each of the rows of L and U. This procedure is denoted by $ILUT(\tau, p)$ of A. More details on ILUT and other preconditioners can be found in [19].

5.2 Flexible GMRES

Recall from what was said above that the role of the preconditioner M is to solve the linear system $Ax = b$ *approximately and inexpensively.* At one extreme, we can find a preconditioner M that is very close to A, leading to a very fast convergence of GMRES, possibly in just one iteration. However, in this situation it is likely that M will require too much memory and be too expensive to compute. At the other extreme, we can compute a very inexpensive preconditioner such as one obtained with ILU(0) – but for realistic problems, convergence is unlikely to be achieved.

If the goal of the preconditioner is to solve the linear system approximately, then one may think of using a full-fledged iterative procedure, utilizing whatever preconditioner is available. The resulting overall method will be an inner-outer method, which includes two nested loops: an outer GMRES loop as defined earlier, and an inner GMRES loop in lieu of a preconditioning operation. In other words, we wish to replace the simple preconditioning operation in line 4 of Algorithm 5.1 by an iterative solution procedure. The result of this is that each step the preconditioner M is actually defined as some complex iterative operation – and we can denote the result by $z_j = M_j^{-1} v_j$. Therefore, the effect of this on Algorithm 5.1 is that the preconditioner M varies at every step j. However, Algorithm 5.1 works only for constant preconditioners M. It would fail for the variable preconditioner case. To remedy this, a variant of GMRES called Flexible GMRES (FGMRES) has been developed, see [19]. For the sake of brevity, we will not sketch the method. The main difference between FGMRES and Algorithm 5.1 is that the vectors z_j generated in line 4 must now be saved. These vectors are then used again in Line 13, in which they replace the vectors v_i of the basis V_m used to compute the approximation x_m. This gives the following algorithm

ALGORITHM **5.2** *FGMRES*

1. *Compute* $r_0 = b - Ax_0$, $\beta := ||r_0||_2$, $v_1 := r_0/\beta$.
2. *Define the* $(m+1) \times m$ *matrix* $\overline{H}_m =$
   $\{h_{i,j}\}_{1 \le i \le m+1, 1 \le j \le m}$. *Set* $\overline{H}_m = 0$.
3. *For* $j = 1, 2, ..., m$ *Do:*
4. *Compute* $z_j := M_j^{-1} v_j$
5. *Compute* $w_j := A z_j$

6. For $i = 1, ..., j$ *Do:*
7. $h_{ij} := (w_j, v_i)$
8. $w_j := w_j - h_{ij} v_i$
9. *EndDo*
10. $h_{j+1,j} = ||w_j||_2$. *If* $h_{j+1,j} = 0$ *set* $m := j$, *goto* 13
11. $v_{j+1} = w_j / h_{j+1,j}$
12. *EndDo*
13. *Define* $Z_m := [z_1, ..., z_m]$, *and* $H_m = \{h_{i,j}\}_{1 \leq i \leq m+1, 1 \leq j \leq m}$.
14. *Compute* y_m *the minimizer of*
$||\beta e_1 - \overline{H}_m y||_2$ *and* $x_m = x_0 + V_m y_m$.
15. *If satisfied Stop, else set* $x_0 := x_m$ *goto* 1.

A non-linear version of FGMRES is obtained, similar to the standard case of GMRES, by computing the matrix-by-vector product Az_j in line 5 via the finite difference formula (5.1).

6 Parallel implementation issues

Domain decomposition methods have recently become a general, simple, and practical means for solving partial differential equations on parallel computers. Typically, a domain is partitioned into several sub-domains and a technique is used to recover the global solution by a succession of solutions of independent subproblems associated with the entire domain. Each processor handles one or several subdomains in the partition and then the partial solutions are combined, typically over several iterations, to deliver an approximation to the global system. All domain decomposition methods (d.d.m.) rely on the fact that each processor can do a big part of the work independently. In this work, a decomposition based approach is employed using an Additive Schwarz algorithm with one layer of overlapping elements. The general solution algorithm used is based on a time marching procedure combined with the inexact-Newton and the matrix-free version of FGMRES or GMRES algorithms. The MPI library is used for communication among processors and PSPARSLIB [22] is used for preprocessing the parallel data structures.

6.1 Data structure for Additive Schwarz d.d.m. with overlapping

In order to implement a domain decomposition approach we need a number of numerical and non-numerical tools for performing the preprocessing tasks required to decompose a domain and map it into processors, as well as to set up the various data structures, and solving the resulting distributed linear system. PSPARSLIB [16, 23], a portable library of parallel sparse iterative solvers, is used for this purpose. The first task is to partition the domain using a partitioner such as METIS [24]. PSPARSLIB assumes a vertex-based partitioning (a given row and the corresponding unknowns are assigned to a certain domain). However, it is more natural and convenient for FEM codes to partition according to elements. The conversion can easily be done by setting up a dual graph which shows the coupling between elements. Assume that each subdomain is assigned to a different processor. We then set up a local data structure in each processor to perform the basic operations such as computing local matrices and vectors, assembling interface coefficients, and preconditioning operations. The first step in setting up the local data-structure mentioned above is to have each processor determine the set of all other processors with which it must exchange information when performing matrix-vector products, computing global residual vector or assembling matrix components related to interface nodes. When performing a matrix-by-vector product or computing a residual global vector (as actually done in the present FEM code), neighboring processors must exchange values of their adjacent interface nodes. In order to perform this exchange operation efficiently, it is important to determine the list of nodes that are coupled with nodes in other processors. These local interface nodes are grouped processor by processor and are listed at the end of the local nodes list. Once the boundary exchange information is determined, the local representations of the distributed linear system must be built in each processor. If it is needed to compute the global residual vector or the global preconditioning matrix, we first compute their local representation to a given processor and move the interface components from remote processors for the operation to complete. The assembly of interface components for

the preconditioning matrix is a non trivial task. A special data structure for the interface local matrix is built to facilitate the assembly operation, in particular when using the Additive Schwarz algorithm with *geometrically non-overlapping* subdomains. The boundary exchange information contains the following items:
1. nproc - The number of all adjacent processors.
2. proc(1:nproc) - List of the nproc adjacent processors.
3. ix - List of local interface nodes, i.e. nodes whose values must be exchanged with neighboring processors. The list is organized processor by processor using a pointer-list data structure.
4. va_{send} - The trace of the preconditioning matrix at the local interface which is computed using local elements. This matrix is organized in a CSR format, each element of which can be retrieved using arrays ia_{send} and ja_{send}. Rows of matrix va_{send} are sent to the adjacent subdomains using arrays proc and ix.
5. ja_{send} and ia_{send} - The Compressed-Sparse-Row arrays for the local interface matrix va_{send} , i.e. ja_{send} is an integer array to store the column positions in *global numbering* of the elements in the interface matrix va_{send} and ia_{send} a pointer array, the i-th entry of which points to the beginning of the i-th row in ja_{send} and va_{send}.
6. va_{recv} - The assembled interface matrix, i.e. each subdomain assembles in va_{recv} interface matrix elements received from adjacent subdomains. va_{recv} is also stored in a CSR format using two arrays ja_{recv} and ia_{recv}.

Additional details on the data structure used as well on the general organization of PSPARSLIB can be found in [22, 23].

6.2 Algorithmic aspects

The general solution algorithm employs a time marching procedure with local time-stepping for steady state solutions. At each time step, a nonlinear system is solved using a quasi-Newton method and the matrix-free GMRES or FGMRES algorithm. The preconditioner used is the block-Jacobian matrix computed and factorized using ILUT algorithm, at each 10 time steps. Interface coefficients of the preconditoner are computed by assembling contributions from all adjacent elements and subdomains, i.e. the va_{recv} matrix is assembled with the local Jacobian matrix. Another aspect worth mentioning is the fact that the FEM formulation requires a continuous state vector $\boldsymbol{V}$ in order to compute a *consistent* residual vector. However, when applying the preconditoner (i.e. multiplication of the factorized preconditioner by a vector) or at the end of Krylov-iterations, a discontinuous solution at the subdomain interface is obtained. To circumvent this inconsistency, a simple averaging operation is applied to the solution interface coefficients.

ALGORITHM **6.1** *Parallel Newton-GMRES*

1. *Get a mesh decomposition using Metis partitioner*
2. *Preprocess the parallel data structures*
3. *Loop over time steps, For* $Is = 1, nsteps$ *Do:*
4. *Compute and factorize the local preconditioning matrix* M *at every* N *time steps*
5. *Loop over Newton iterations, For* $In = 1, niterations$ *Do:*
6. *Compute Initial residual* $r_0 = F(x_0)$ *and* $\beta := ||r_0||_2$, $v_1 := r_0/\beta$.
7. *Define the* $(m+1) \times m$ *matrix* $\overline{H}_m =$
   $\{h_{i,j}\}_{1 \le i \le m+1, 1 \le j \le m}$. *Set* $\overline{H}_m = 0$.
8. *For* $j = 1, 2, ..., m$ *Do:*
9. *Compute* $z_j := M^{-1} v_j$
10. *Compute local representation of the perturbed solution* $x_0 + \epsilon z_j$*and average interface components to obtain the global representation, then compute* $F(x_0 + \epsilon z_j)$
11. *Compute* $w_j := \frac{F(x_0+\epsilon z_j) - F(x_0)}{\epsilon}$
12. *For* $i = 1, ..., j$ *Do:*
13. *Compute local coefficient* $h_{ij} := (w_j, v_i)$ *and sum over subdomains*
14. $w_j := w_j - h_{ij} v_i$
15. *EndDo*
16. *Compute local coefficient* $h_{j+1,j} = ||w_j||_2$ *and sum over subdomains.*
    If $h_{j+1,j} = 0$ *set* $m := j$, *goto* 19
17. $v_{j+1} = w_j / h_{j+1,j}$

18. EndDo
19. Define $V_m = [v_1, ..., v_m]$, *and* $H_m = \{h_{i,j}\}_{1 \leq i \leq m+1, 1 \leq j \leq m}$.
20. Compute y_m *the minimizer of* $||\beta e_1 - \overline{H}_m y||_2$,
get local representation $x_m = x_0 + V_m y_m$ *and average interface components to obtain continuous global solution.*
21. If satisfied EndDo, else set $x_0 := x_m$ *goto* 6.
22. EndDo
23. EndDo

For compressible flows, the following parameters are generally used: $m = 10$, $N = 10$, $\epsilon = 10^{-6}$, $niterations = 1$, $p = NNZ/NEQ + lfil$ and $\tau = 10^{-3}$; with NNZ the number of nonzero enteries, NEQ the number of local equations and $(-10 lelfille 10)$. The parallelized Newton-FGMRES is similar to Algorithm 6.1 but step (9) is replaced by an inner preconditioned-GMRES loop for which $m = 2$.

7 Numerical results

The EBS formulation has been implemented in 2D and 3D, and tested for computing viscous and inviscid compressible flows. Also, EBS results are compared with those obtained using SUPG formulation (the definition of the stabilization matrix employed is given by $\boldsymbol{\tau} = (\sum_i |\boldsymbol{B}_i|)^{-1}$) and with some results obtained using a Finite Volume code developed in INRIA (France). All tests have been performed on a SUN Enterprise 6000 parallel machine with 165 MHz processors. The objective of the numerical tests is to assess the stability and accuracy of EBS formulation as compared to SUPG and FV methods. Linear finite element approximations over tetrahedra are used for 3D calculations and mixed interpolations over triangles for 2D (quadratic for momentum and linear for density, pressure, temperature and energy). A time-marching procedure is used, second order accurate for unsteady solutions and first order Euler scheme with nodal time steps for steady solutions.

7.1 Two dimensional tests

Several benchmark tests were carried out. We first present results for subsonic and transonic flows around a NACA0012 airfoil at, respectively, the following conditions: (inviscid, Ma=0.50 and angle of attack =0), (inviscid, Ma=0.80 and angle of attack =1.25 degree) and (viscous $Re = 10000$, $Ma = 0.80$ and angle of attack =0). A symmetric mesh of 8150 triangular elements is used. For EBS formulation, the parameter β was set to 0.5. For the inviscid computations, all the formulations gave similar results although the shocks are steeper for EBS (Figures 1 and 2). For the viscous case the results (Figures 3) clearly show a strong vortex shedding phenomenon at the trailing edge. Again, the shock is steeper for EBS formulation. These results compare quite well with those obtained in [25] and [26] where refined and adapted meshes were used. A second problem consists of solving a two-dimensional viscous flow at $Re = 1000$, $Ma = 3$ and zero angle of attack over a flat plate. A structured mesh of $2 \times (28 \times 16)$ triangular elements was employed. Figure 4 shows the isomachs. The boundary layer obtained with SUPG and the new design for $\boldsymbol{\tau}$ seems a little thinner. A smooth solution is obtained using EBS. A thinner boundary layer could be obtained by decreasing β to 0.3 for instance, as has actually been observed in our tests.

7.2 Three dimensional tests

Three-dimensional tests have been carried out for computing viscous flows over a flat plate, inviscid as well as turbulent flows around the ONERA-M6 wing and inviscid flows around the AGARD-445.6 wing [27].
For the flat plate, flow conditions are set to ($Re = 100$ and $Ma = 1.9$) and ($Re = 400$ and $Ma = 1.9$). A coarse and unadapted mesh is used for this test. Figures 5 and 6 show the Mach number contours (at a vertical plane). It is clearly shown that SUPG and FV solutions are more diffusive than EBS solution.
For the ONERA-M6 wing, a Euler solution is computed for $Ma = 0.8447$ and an angle of attack of 5.06 degrees. The mesh used has 15460 nodes and 80424 elements. Figures 7 show the Mach number contours at the root section for EBS, SUPG, FV methods respectively. It is clearly shown that EBS method is stable and less diffusive than SUPG method. The shock is well captured as in the 2nd order FV solution. Under the same conditions (Mach number, angle of attack and mesh), a turbulent flow is computed for a Reynolds number of

$R_e = 11.710^6$ and for a distance $\delta = 10^{-4}$. These are the same flow conditions used in [21]. However, in [21] a much finer mesh on the wall is employed. Figures 8 present respectively the Mach contours obtained with EBS, SUPG, first-order FV and second-order FVmethods. The Finite Volume code uses the κ-ϵ turbulence model. These results show clearly that SUPG and first-order FV codes give a smeared shock. It is fairly well captured by EBS method. However, the use of the second-order FV method results in a much stronger shock. It is also observed from these figures that the positions of the shock obtained respectively with the EBS-SA and FV-κ-ϵ codes are quite different. However, the results obtained with the EBS-based code are comparable to those obtained in [21]. It seems that the way the nonpenetration condition is implemented for the trailing edge nodes is responsible for these discrepancies. In our code a unit normal vector is computed for every node and the condition $\boldsymbol{u} \cdot \boldsymbol{n}$ is enforced precisely. In the used finite volume code, this condition is obtained in a weak form. Figure 9 shows the isocountours for the turbulent viscosity obtained with EBS and SUPG methods using S-A turbulence model.
The AGARD-445.6 is a thin swept-back and tapered wing with a symmetrical NACA 65A004 airfoil section. This wing is popular in aeroelastic studies as experimental results exist. An unstructured grid is employed for Euler computations which has 84946 nodes and 399914 and generates 388464 coupled equations (Figure 10). A flow at a free-stream Mach number of 0.96 and zero angle of attack is computed and results are compared with those of [28] where a structured and fine mesh is used. Figure 10 shows a comparison of pressure coefficient contours on the upper surface obtained using our SUPG and EBS methods and results of [28]. It can be observed that the results obtained with EBS are qualitatively similar to [28] while those obtained with SUPG are more diffusive. A comparison of Cp coefficients at the root section is also presented. In this plot the results of reference [28] actually correspond to Navier-Stokes computations at a high Reynolds number since those corresponding to the Euler case are not reported. Discrepancies at the trailing part are likely due to the boundary layer and shock interactions in Navier-Stokes solution.

Tables 1 and 2 show speed-up results obtained in the case of Euler computations around the Onera-M6 and AGARD 445.6 wings using the parallel version of the code. Figure 11 shows the convergence history for the case of the Euler flow around the Onera-M6 wing using EBS formulation and a different number of processors. Identical convergence is then ensured for any number of processors. For the small-scale problem, efficiency is of order of 90%. However, it drops to 70% for the large-scale problem. The performance drop is mainly caused by the increase of the total number of GMRES iterations. The additive Schwarz algorithm, with only one layer of overlapping elements, along with ILUT factorization and the FGMRES/GMRES algorithm, seems to provide reasonable numerical tools for parallel solutions of compressible flows. However, there is still room for improvement, using for instance more overlapping layers or a more sophisticated preconditioner such as the distributed Schur complement [[29] and [30]].
Another Euler test has been performed on the Onera-M6 wing using a frequently studied parameter combination of $Ma = 0.8395$ and an angle of attack of 3.06 degrees. This transonic case gives rise to a characteristic lambda-shock. A relatively fine mesh was used (187248 nodes, 905013 element and 936240 degrees of freedom). Comparisons of the C_p coeficients with the experimental data [31] show reasonable agreement for an inviscid model and for the mesh used (Figure 12). Figure 13 shows the C_p contours for EBS and SUPG formulations which are used along with a final value of $\mu_{cc2=.25}$. For EBS formulation, the lambda shock seems to start appearing. A comparison with the numerical results obtained in [32] are shown in Figure 14.

Conclusion

A new stabilization finite element formulation (EBS) is proposed in this study and applied to multidimensional systems, in particular Euler and Navier-Stokes equations. Also, a new design for the time scale matrix $\boldsymbol{\tau}$ is proposed for the classical SUPG formulation. These formulations need more stabilization for stagnation points and shocks. To do this, the shock capturing operator of Le Beau et al [18] is used. In the framework of EBS formulation, it is possible to add more stabilization to the upwind element as it is usually done in the FV formulations. Numerical tests in 2D and 3D show that the EBS formulation combines well with the shock capturing operator of [18] while SUPG seems very diffusive. On the other hand, SUPG formulation is robust and has a good convergence for Euler and turbulent flows using standard iterative solvers such as the ILUT preconditioned GMRES. However, some convergence difficulties are encountered for EBS formulation, especially in the case of

turbulent flows. To solve these difficulties, the ILUT preconditioned FGMRES has been used as the default iterative solver for turbulet flows. In terms of CPU time, EBS formulation is in general as twice consuming as SUPG (using the old designs of τ and only one Gauss quadrature point). This trend is similar to what is generally observed when comparing first and higher order methods for compressible flows. We also discussed some parallel implementation issues. An Additive Schwarz domain decomposition method with *algebraically* one layer of overlapping elements is implemented along with the ILUT-FGMRES/GMRES algorithms. Numerical results show that the parallel code offers reasonable performance for a number of processors less than 16.

Acknowledgments

This research has been funded by the Natural Sciences and Engineering Research Council of Canada (NSERC), PSIR research program of ETS and by la Fondation Bombardier. The second author acknowledges support from the National Science Foundation and from the Minnesota Supercomputer Institute. The authors would like to thank Professor Bruno Koobus (Universite de Montpellier) for providing the results obtained with his finite volume code and Alexandre Forest for his help in generating the three-dimensional meshes.

References

[1] J. Donea. "A Taylor-Galerkin method for conservative transport problems", *Internat. J. Numer. Methods Engrg.*, **20**, 101-120 (1984).

[2] R. Lohner, K. Morgan and O.C. Zienkiewicz. "Adaptive finite element procedure for high speed flows", *Computer Methods in Applied Mechanics and Engineering*, **51** , 441-465 (1980).

[3] A.N. Brooks and T.J.R. Hughes. "Streamline upwind/Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations", *Computer Methods in Applied Mechanics and Engineering*, **32**, 199-259 (1982).

[4] T.J.R. Hughes and T.E. Tezduyar. "Finite element methods for first-order hyperbolic systems with particular emphasis on the compressible Euler equations". *Computer Methods in Applied Mechanics and Engineering*, **45**, 217-284 (1984).

[5] T.J.R. Hughes, L.P. Franca and M. Mallet. "A new finite element formulation for computational fluid dynamics: I. Symmetric forms of the compressible Euler and Navier-Stokes equations and the second law of thermodynamics", *Computer Methods in Applied Mechanics and Engineering*, **54**, 223-234 (1986).

[6] T.J.R. Hughes, L.P. Franca and Hulbert. "A new finite element formulation for computational fluid dynamics: VIII. The Galerkin/least-squares method for advective-duffusive equations", *Computer Methods in Applied Mechanics and Engineering,* **73**, 173-189 (1989).

[7] A. Soulaimani and M. Fortin. "Finite Element Solution of Compressible Viscous Flows Using Conservative Variables", *Computer Methods in Applied Mechanics and Engineering* , **118**, 319-350 (1994).

[8] P.Hansbo. "Explicit streamline diffusion finite element methods for the compressible Euler equations in conservation variables", *J. Comput. Phys.*, **109**, 274-288 (1993).

[9] N.E. ElKadri, A. Soulaimani and C. Deschene. "A finite element formulation for compressible flows using various sets of independent variables", *Computer Methods in Applied Mechanics and Engineering*, **181**, 161-189 (2000).

[10] A. Dervieux. "Steady Euler simulations using unstructured meshes",*Von Karman lecture note series 1884-04*. March 1980.

[11] P.L. Roe. Approximate Riemann solvers, parameter vectors and difference schemes", *J. Comput. Phys.*, **43**, 357-371 (1981).

[12] B. Van Leer. "Towards the ultimate conservative difference scheme V: a second-order sequel to Goudonov's method", *J. Comput. Phys.*, **32**, 361-370 (1979).

[13] C.B. Laney. Computational Gasdynamics. Cambridge University Press, (1998).

[14] B. Cockburn, S. Hou, and C.W. Shu. "The Runge-Kutta Local Projection Discontinuous Galerkin Finite Element Method for Conservation Laws IV: The Multidimensional Case", *Mathematics of Computation"*,**62** No. 190, 545-581 (1990).

[15] A. Soulaimani and C. Farhat. "On a finite element method for solving compressible flows", *Proceedings of the ICES-98 Conference: Modeling and Simulation Based Engineering*. Atluri and O'Donoghue editors, 923-928, October (1998).

[16] P.R. Spalart and S.R. Allmaras. "A one-equation turbulence model for aerodynamic flows", *La Recherche Aerospatiale*, **1**, 5-21 (1994).

[17] T.J.R. Hughes and Mallet. " A new finite element formulation for computational fluid dynamics: III. The generalized streamline operator for multidimensional advective-diffusive systems", *Computer Methods in Applied Mechanics and Engineering*, **58**, 305-328 (1986).

[18] G.J. Le Beau, S.E. Ray, S.K. Aliabadi and T.E. Tezduyar. "SUPG finite element computation of compressible flows with the entropy and conservative variables formulations", *Computer Methods in Applied Mechanics and Engineering*, **104**, 397-422, (1993).

[19] Y. Saad. "Iterative Methods For Sparse Linear Systems", PWS Publishing Company, 1996.

[20] A. Soulaimani, G. Da Ponte and N. Ben Salah. "Acceleration of GMRES convergence for three-dimensional compressible, incompressible and MHD flows", AIAA-99-3381 (1999).

[21] N. T. Frink. "Tetrahedral Unstructured Navier-Stokes Method for Turbulent Flows", AIAA Journal, **36**, No. 11, November (1998).

[22] Y. Saad and A. Malevsky. "PSPARSLIB: A portable library of distributed memory sparse iterative solvers". In V. E. Malyshkin et al., editor, *Proceedings of Parallel Computing Technologies (PaCT-95), 3-rd international conference, St. Petersburg, Russia, Sept. 1995*, 1995.

[23] S. Kuznetsov, G. C. Lo, and Y. Saad. "Parallel solution of general sparse linear systems using PSPARSLIB". In Choi-Hong Lai, Petter Bjorstad, Mark Cross, and Olof B. Widlund, editors, *Domain Decomposition XI*, pages 455–465, Domain Decomposition Press, Bergen, Norway, (1999).

[24] G. Karypis and V. Kumar. "A fast and high-quality multi-level scheme for partitioning irregular graphs", *SIAM Journal on Scient. Comput.*, vol. 20, pp. 359-392 (1998).

[25] L. Cambier. "Computation of viscous transonic flows using an unsteady type method and a nozal grid refinement technique", report O.N.E.R.A., France, 1985.

[26] Y. Bourgault. "Méthodes d'éléments finis en mécanique des fluides, conservation et autres propriétés", Ph.D. thesis, Université Laval, 1996.

[27] E.C. Yates. "AGARD Standard Aeroelastic Configuration for Dynamics Response, Candidate Configuration I.-Wing 445.6", NASA TM 100492, 1987.

[28] E.M. Lee-Rausch and J.T. Batina. "Calculation of AGARD wing 445.6 flutter using Navier-Stokes aerodynamics", AIAA paper 93-3476, 1993.

[29] Y. Saad and M. Sosonkina. Distributed Schur complement techniques for general sparse linear systems. Technical Report umsi-97-159, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 1997. To appear.

[30] Y. Saad, M. Sosonkina, and J. Zhang. Domain decomposition and multi-level type techniques for general sparse linear systems. In *Domain Decomposition Methods 10*, Providence, RI, 1998. American Mathematical Society.

[31] V. Shmitt and F. Charpin. Pressure distributions on the ONERA M6 wing at transonic Mach numbers. Technical Report AR-138, AGARD, May 1979.

[32] W. D. Gropp, D. E. Keyes, L. C. MCINNES and M. D. Tidriri. Globalized Newton-Krylov-Schwarz algorithms and software for parallel implicit CFD. Int. J. High Performance Computing Applications, 14:102-136, 2000.

Appendix: A computer program for computing the EBS tau matrix.

```
      subroutine tau3d-ebs(taub,v1,v2,v3,pres,dens,gama,ci,
     1                          beta0,iel,ndim,hel,vmu)
c=====================================================================
c     Stabilisation matrix TAU calculation in three dimensions
c  Input : - velocity components:  v1,v2, v3.
c          - density : dens
c          - gama: specfici heat ratio
c          - unit normal vector: ci
c          - viscoity: vmu
c          - upwinding parameter: beta (see paper)
c  Output:   tau EBS matrix
c                                  Author:  Azzeddine SOULAIMANI
c                                    asoulaimani@mec.etsmtl.ca
c=====================================================================
      double precision t(5,5),ti(5,5),vm(5,5),vmi(5,5),sol(5,5),
     1        d(5),taub(5,5),ci(3),sol1(5,5),soli1(5,5),
     2       v1,v2,v3,pres,dens,gama,gama1,beta,hel,vmu,Pec,
     3       rtc,cel,rc,vv,c1,c2,c3,pl1,pl2,pl3,plmax,
     4       rt2,ct1,ct2,ct3,zero,un,deux,eps,beta0
      integer iel,ndim,i,j
      data zero/0.d0/, un/1.d0/, deux/2.d0/, eps/1.d-15/
      gama1 = gama - un
      rt2 = sqrt(deux)
      do i = 1,5
      do j = 1,5
      taub(i,j) = zero
      enddo
      enddo
c---- sound speed
      if(pres.le.zero.or.dens.le.zero)then
      write(*,*)'    Problem : negative pressure or density in element:',iel
      stop
      endif
      cel = sqrt(gama*pres/dens)
      rc= dens/(cel*rt2)
c---- velocity norm square
      vv = v1*v1 + v2*v2 + v3*v3 + eps*eps
c====== TAU matrix computation ========================================
      c1 = ci(1)
      c2 = ci(2)
      c3 = ci(3)
      rtc = sqrt(c1*c1 + c2*c2 + c3*c3)
c----- eigenvalues of the Gradient matrices as given by Warming et al.
      pl1 = (c1*v1 + c2*v2 + c3*v3) + eps
      pl2 = (c1*v1 + c2*v2 + c3*v3 + cel*rtc) + eps
```

```
      pl3 = (c1*v1 + c2*v2 + c3*v3 - cel*rtc) + eps
c----- diagonal matrix
      d(1) = pl1
      d(2) = pl1
      d(3) = pl1
      d(4) = pl2
      d(5) = pl3
c----- update the diagonal matrix
      do i=1,5
      Pec= (abs(d(i))*hel)/(6.d0*vmu)
      beta= dmin1(Pec,1.d0)*beta0
      if(d(i).ge.eps) then
       d(i) = (1.d0 + beta)
      else
           if(dabs(d(i)).gt.eps) then
           d(i) = - (1.d0 - beta)
           else
           d(i) = 0.d0
           endif
      endif
      enddo
c-----------------------------------------
      ct1 = c1/rtc
      ct2 = c2/rtc
      ct3 = c3/rtc
c------ Initialization
      do i=1,5
      do j=1,5
      vm(i,j)=zero
      vmi(i,j)=zero
      ti(i,j)=zero
      t(i,j)=zero
      enddo
      enddo
c------ T matrix as given in Warming et al.
      t(1,1) = ct1
      t(1,2) = ct2
      t(1,3) = ct3
      t(1,4) = rc
      t(1,5) = rc
      t(2,2) = - ct3
      t(2,3) = ct2
      t(2,4) = ct1/rt2
      t(2,5) = - ct1/rt2
      t(3,1) = ct3
      t(3,3) = - ct1
      t(3,4) = ct2/rt2
      t(3,5) = - ct2/rt2
      t(4,1) = - ct2
      t(4,2) = ct1
      t(4,4) = ct3/rt2
      t(4,5) = - ct3/rt2
      t(5,4) = dens*cel/rt2
```

```
      t(5,5) = dens*cel/rt2
c------ TI invers matrix as given in Warming et al.
      ti(1,1) = ct1
      ti(1,3) = ct3
      ti(1,4) = - ct2
      ti(1,5) = - ct1/(cel*cel)
      ti(2,1) = ct2
      ti(2,2) = - ct3
      ti(2,4) = ct1
      ti(2,5) = - ct2/(cel*cel)
      ti(3,1) = ct3
      ti(3,2) = ct2
      ti(3,3) = - ct1
      ti(3,5) = - ct3/(cel*cel)
      ti(4,2) = ct1/rt2
      ti(4,3) = ct2/rt2
      ti(4,4) = ct3/rt2
      ti(4,5) = un/(dens*cel*rt2)
      ti(5,2) = - ct1/rt2
      ti(5,3) = - ct2/rt2
      ti(5,4) = - ct3/rt2
      ti(5,5) = un/(dens*cel*rt2)
c------ M matrix as given in Warming et al.
      vm(1,1) = un
      vm(2,1) = v1
      vm(2,2) = dens
      vm(3,1) = v2
      vm(3,3) = dens
      vm(4,1) = v3
      vm(4,4) = dens
      vm(5,1) = vv/deux
      vm(5,2) = dens*v1
      vm(5,3) = dens*v2
      vm(5,4) = dens*v3
      vm(5,5) = un/gama1
c------ MI invers matrix as given in Warming et al.
      vmi(1,1) = un
      vmi(2,1) = - v1/dens
      vmi(2,2) = un/dens
      vmi(3,1) = - v2/dens
      vmi(3,3) = un/dens
      vmi(4,1) = - v3/dens
      vmi(4,4) = un/dens
      vmi(5,1) = vv*gama1/deux
      vmi(5,2) = - gama1*v1
      vmi(5,3) = - gama1*v2
      vmi(5,4) = - gama1*v3
      vmi(5,5) = gama1
c---- product M.T
      call mpro(sol1,vm,t,5)
c---- product TI.MI
      call mpro(soli1,ti,vmi,5)
c----- product M.T.D
```

```
      do i = 1,5
         do j = 1,5
          sol1(i,j) = sol1(i,j)*d(j)
         enddo
      enddo
c----- product M.T.D.TI.MI
      call mpro(sol,sol1,soli1,5)
c----- TAU matrix
      do i = 1,5
         do j = 1,5
           taub(i,j) = taub(i,j) + sol(i,j)
         enddo
      enddo
      return
      end
```

Table 1: Parallel Performance of Euler flow around Onera-M6 wing

Number of	SUPG		EBS	
processors	Speedup	Efficiency	Speedup	Efficiency
2	1.91	0.95	1.86	0.93
4	3.64	0.91	3.73	0.93
6	5.61	0.94	5.55	0.93
8	7.19	0.90	7.30	0.91
10	9.02	0.90	8.79	0.88
12	10.34	0.86	10.55	0.88

Table 2: Parallel Performance of an Euler flow calculation around AGARD wing 445.6 using SUPG, GMRES and 300 pseudo-time steps at CFL=20

Number of	GMRES		
processors	Speedup	Efficiency	iterations
1	1	1.00	1004
4	2.95	0.74	1285
6	4.24	0.71	1399
8	5.86	0.73	1357
10	6.53	0.65	1457
12	8.76	0.73	1342

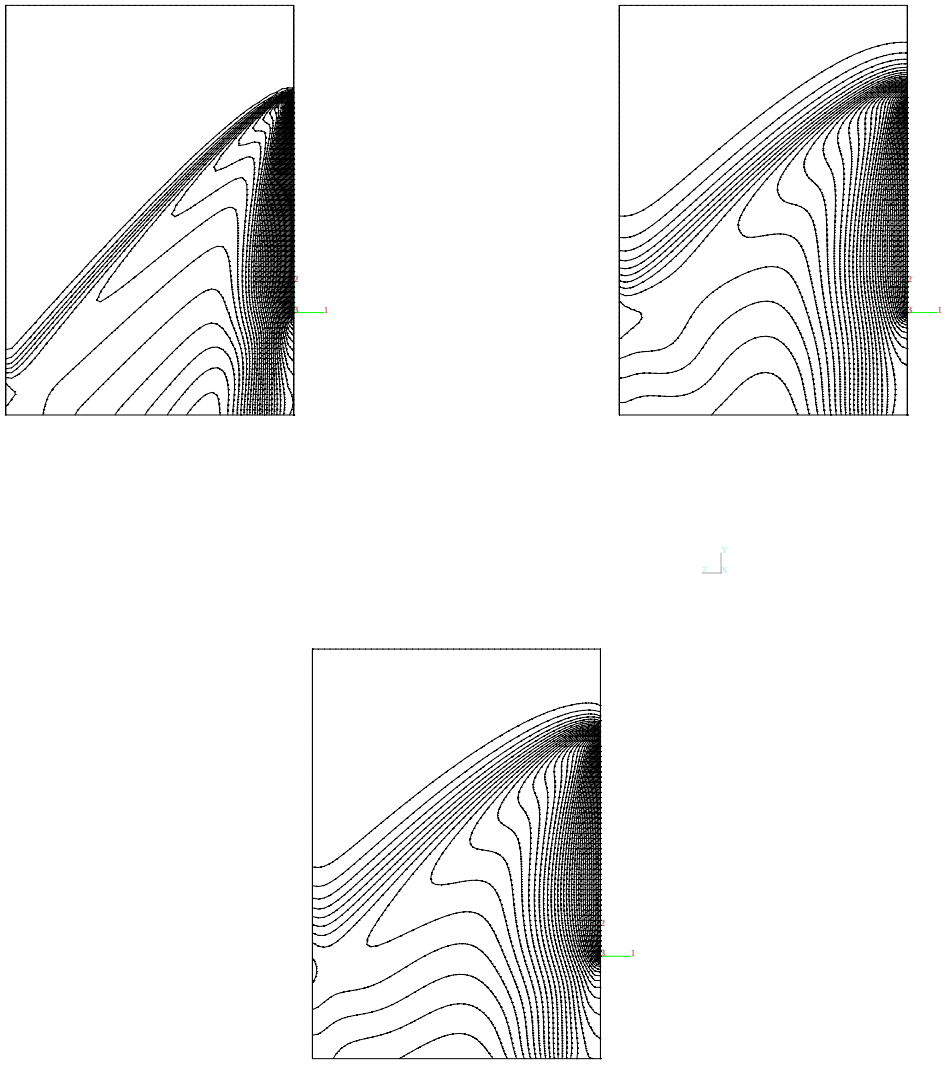


Figure 5: 3D viscous flow at Re = 100 and M = 1.9. Mach contours for EBS, SUPG and FV methods.

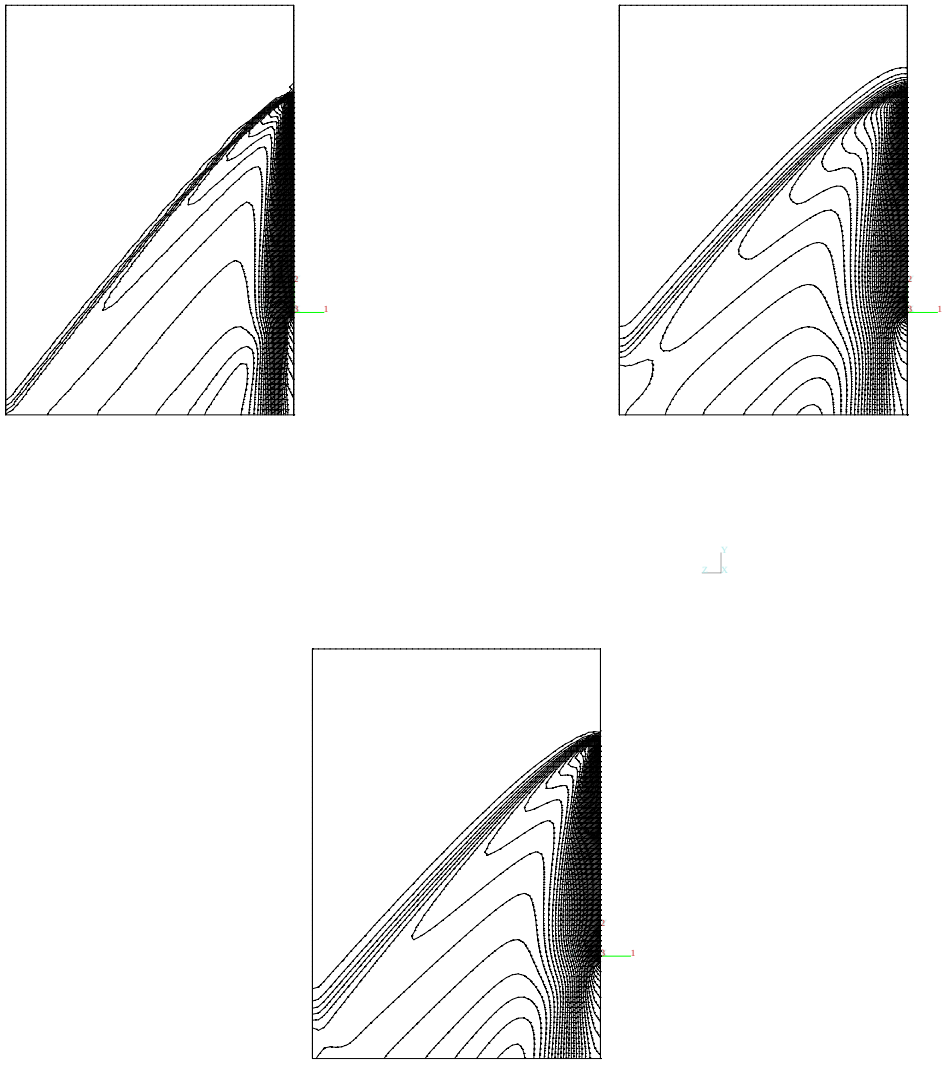


Figure 6: 3D viscous flow at Re = 400 and M = 1.9. Mach contours for EBS, SUPG and FV methods.

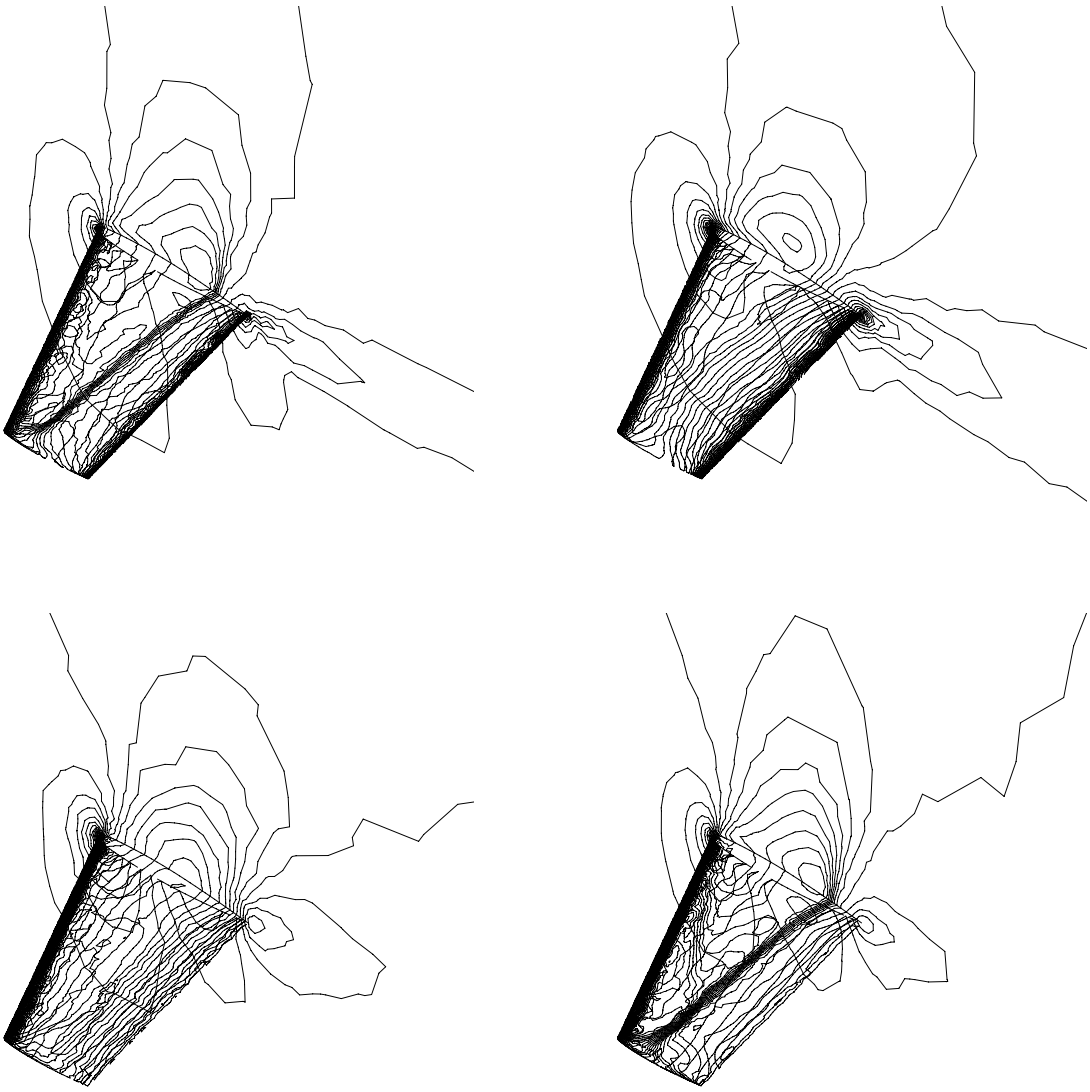


Figure 7: Euler flow around Onera-M6 wing. Mach contours for EBS, SUPG and 1st order FV and 2nd order FV methods.

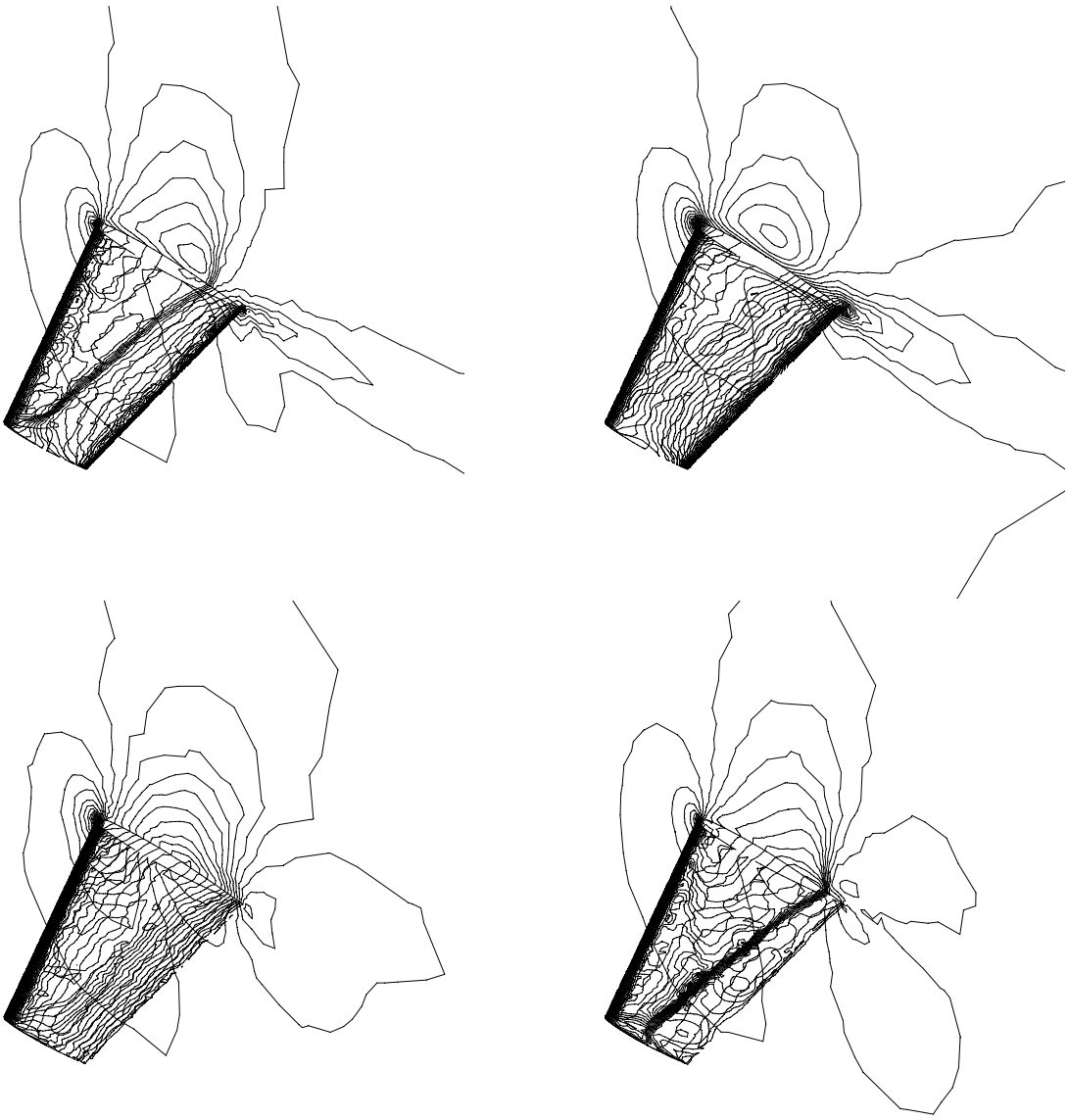
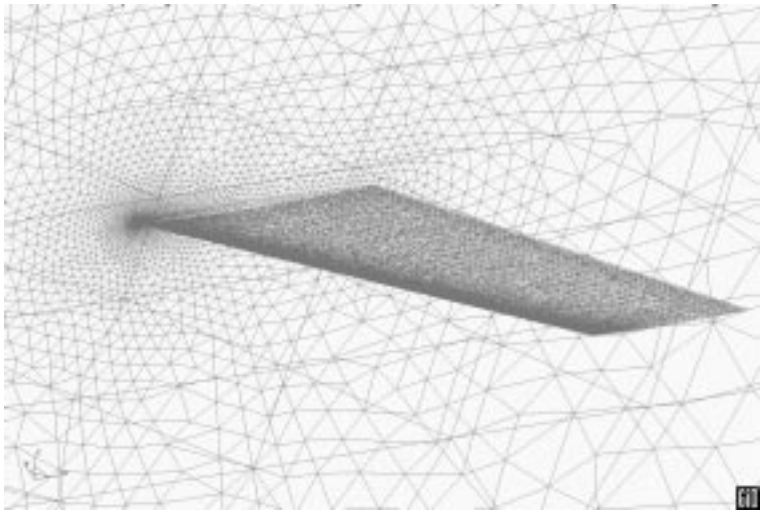
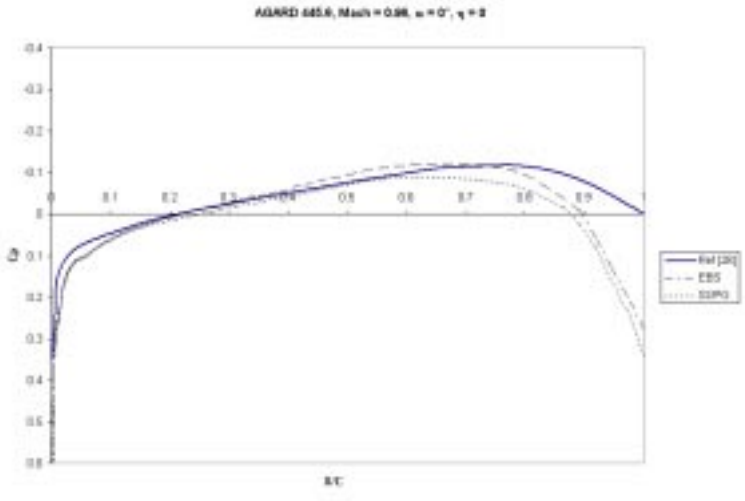
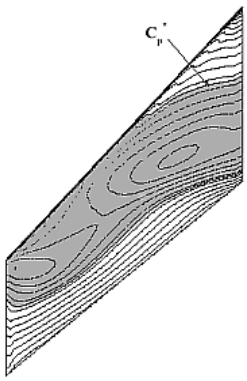


Figure 8: Turbulent flow around Onera-M6 wing. Mach contours for EBS, SUPG and 1st order FV and 2nd order FV methods.

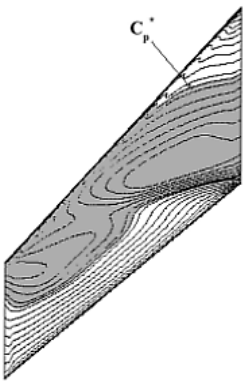
Figure 9: Turbulent flow around Onera-M6 wing. Turbulent viscosity contours for EBS and SUPG methods.



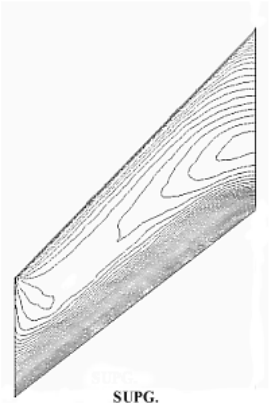




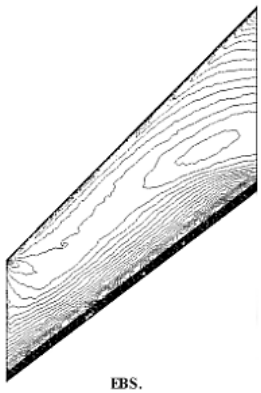
Navier-Stokes.



Euler.



SUPG.



EBS.

Figure 10: Euler flow around Agard wing 445.6. EBS and SUPG methods.

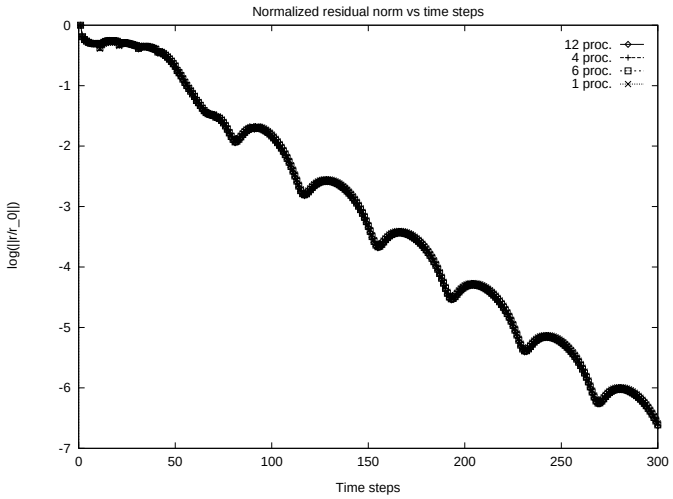


Figure 11: Euler flow around Onera-M6 wing. Convergence history with EBS.