# Application de l'algorithme GMRES a la resolution des equations de Navier-Stokes compressible. Etude de divers preconditionnements

L.C. Dutto

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

N° 1234

***Programme 5***
*Automatique, Productique,*
*Traitement du Signal et des Données*

***Programme 7***
*Calcul Scientifique,*
*Logiciels Numériques et Ingénierie Assistée*

# APPLICATION DE L'ALGORITHME GMRES A LA RESOLUTION DES EQUATIONS DE NAVIER-STOKES COMPRESSIBLE. ETUDE DE DIVERS PRECONDITIONNEMENTS

**Laura C. DUTTO**

**Juin 1990**

*RR_1234*

# APPLICATION DE L'ALGORITHME GMRES A LA RESOLUTION DES EQUATIONS DE NAVIER–STOKES COMPRESSIBLE. ETUDE DE DIVERS PRECONDITIONNEMENTS.

# APPLICATION OF GMRES ALGORITHM TO SOLVE THE COMPRESSIBLE NAVIER–STOKES EQUATIONS. STUDY OF PRECONDITIONERS.

Laura C. Dutto

*AMD/BA (DGT/DEA-MN), 78 quai Marcel Dassault, 92214 ST CLOUD Cedex, France*

*INRIA, B.P. 105, 78153 LE CHESNAY CEDEX, France*

# APPLICATION DE L'ALGORITHME GMRES A LA RESOLUTION DES EQUATIONS DE NAVIER-STOKES COMPRESSIBLE. ETUDE DE DIVERS PRECONDITIONNEMENTS.

**Résumé:**

On étudie la simulation numérique de l'écoulement d'un fluide visqueux compressible autour d'un obstacle dans $I\!R^2$, modélisé par les équations de Navier–Stokes, écrites sous forme non conservative.

On utilise une approche instationnaire des équations, même quand il s'agit d' obtenir une solution stationnaire. La discrétisation en espace est réalisée au moyen de méthodes d'éléments finis conformes en utilisant, pour les différentes variables, des espaces d'approximation compatibles.

La discrétisation implicite en temps (Euler rétrograde) demande la résolution de problèmes non linéaires. On le fait au moyen d'une méthode de sous–espaces de Krylov (l'algorithme de GMRES non linéaire). Pour préconditionner la fonction non linéaire à étudier on utilise des formes approchées du jacobien de celle-ci; on tient compte de sa structure par blocs. On présente une famille de préconditionneurs non symétriques obtenue par des factorisations de type $LDU$.

**Mots clé:**

Equations de Navier–Stokes compressible, méthode d'éléments finis; méthode de GMRES; préconditionnement linéaire; matrices par blocs; factorisation de Gauss (complète et incomplète).

# APPLICATION OF GMRES ALGORITHM TO SOLVE THE COMPRESSIBLE NAVIER–STOKES EQUATIONS. STUDY OF PRECONDITIONERS.

**Abstract:**

We discuss in this paper the numerical simulation of compressible viscous flows around airfoils in $I\!R^2$, modelled by the Navier–Stokes equations in the non conservative form.

We use an unsteady model of the equations, even when we want to capture a steady solution. We choose the finite element methods for the space discretization (conforming Lagrange elements) with the compatible finite element approximations for the different variables.

The implicit time discretization (backward Euler) requests to solve nonlinear problems. We do this by an (iterative) Krylov subspace method (the nonlinear GMRES algorithm). To precondition the nonlinear mapping under study, we use approximations of his jacobian; in doing this we benefit also of the block structure of the jacobian. We develop nonsymmetric preconditioners based upon the *LDU* factorization.

**Key words:**

Compressible Navier–Stokes equations, finite element methods; GMRES method; linear preconditioning; block matrix; (complete and uncomplete) Gaussian elimination.

## Table des matières:

## 1. Introduction

La simulation numérique d'écoulements de fluides visqueux compressibles modélisés par les équations de Navier–Stokes est basée sur différents schémas de discrétisation en temps (explicite, implicite linéarisé, implicite), combinés à des discrétisations en espace adéquates (cf. [P3]). Dans le présent travail, nous avons choisi un schéma implicite du premier ordre en temps, associé à des espaces d'éléments finis compatibles pour les différentes variables (cf. [B5]).

Après discrétisation en temps, le problème discret qui se pose à chaque pas de temps, consiste à résoudre un système non linéaire de dimension finie. On le résout par une méthode de sous-espaces de Krylov (la méthode de GMRES non linéaire; cf. [S1], [B3]). Nous présentons les propriétés de cette méthode.

Pour accélérer la convergence locale de notre solveur, nous développons des préconditionneurs non symétriques basés sur la factorisée par l'algorithme de Gauss de la matrice jacobienne de la fonction associée au problème à résoudre. Nous utilisons aussi la structure par blocs creuse de cette matrice, qui provient du groupement naturel des inconnues par nœud lors de la discrétisation par éléments finis. Des résultats numériques illustrent les avantages et les inconvénients de tels opérateurs de préconditionnement.

Dans §2, on décrit les équations de Navier–Stokes régissant un fluide visqueux compressible, écrites en variables non conservatives, ainsi que les discrétisations en temps et en espace choisies. Dans §3, on présente la méthode de GMRES, utilisée pour résoudre le problème non linéaire qui se pose à chaque pas de temps. Dans §4, on présente les opérateurs de préconditionnement utilisés. Le §5 est consacré à développer les outils nécessaires à la définition de nos opérateurs de préconditionnement, à savoir, divers algorithmes conduisant à des décompositions plus ou moins complètes de type $LDU$ d'une matrice par blocs. Ils ont leur origine dans la factorisation de Gauss usuelle d'une matrice. Les résultats numériques qui montrent la performance des opérateurs de préconditionnement choisis sont présentés dans §6, et les conclusions sont exposées dans §7. Finalement, l'annexe 1 explicite le calcul du jacobien associé au système non linéaire à résoudre.

# 2. Présentation du problème physique

## 2.1. Les équations de Navier–Stokes compressible

On rappelle ci-dessous les équations Navier-Stokes qui régissent l'écoulement d'un fluide visqueux compressible. Pour plus de détails, on renvoie par exemple, à Landau-Lifchitz [L2].

On utilisera les notations suivantes: $\mathbf{u}$, $\rho$ et $T$ désignent respectivement la vitesse, la densité et la température du fluide; $\gamma := C_p/C_v$ désigne le rapport des chaleurs spécifiques du fluide, à pression et à volume constant notées respectivement $C_p$ et $C_v$ ($\gamma = 1.4$ pour l'air); Pr, $M$ et Re désignent le nombre de Prandtl, de Mach et de Reynolds respectivement (Pr = .72 pour l'air).

On utilisera, en fait, le logarithme de la densité:

$$\sigma = \ln \rho \tag{2.1}$$

comme nouvelle variable (ce changement de variable n'est pas nécessaire ici, mais on l'a effectué pour des raisons historiques; cf. [B1]). Les équations de Navier-Stokes pour un fluide compressible, écrites en variables non conservatives adimensionnées, deviennent:

$$\frac{\partial \sigma}{\partial t} + \nabla \cdot \mathbf{u} + \mathbf{u} \cdot \nabla \sigma = 0 \tag{2.2}$$

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} + (\gamma - 1)(T\nabla\sigma + \nabla T) - \frac{e^{-\sigma}}{\mathrm{Re}}\Big(\Delta \mathbf{u} + \frac{1}{3}\nabla(\nabla \cdot \mathbf{u})\Big) = \mathbf{0} \tag{2.3}$$

$$\frac{\partial T}{\partial t} + \mathbf{u} \cdot \nabla T + (\gamma - 1)T(\nabla \cdot \mathbf{u}) - \frac{e^{-\sigma}}{\mathrm{Re}}\Big(\frac{\gamma}{\mathrm{Pr}}\Delta T + F(\nabla \mathbf{u})\Big) = 0. \tag{2.4}$$

Le terme $F(\cdot)$ dans (2.4) est donné dans le cas bidimensionnel par:

$$F(\nabla \mathbf{u}) = \frac{4}{3}\Big(|\frac{\partial u}{\partial x}|^2 + |\frac{\partial v}{\partial y}|^2 - \frac{\partial u}{\partial x}\frac{\partial v}{\partial y}\Big) + (\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x})^2. \tag{2.5}$$

On se place dans le cas particulier (important) de l'écoulement d'un fluide autour d'un profil d'aile. Soit $d$ la dimension de l'espace physique ($d = 2$ dans notre cas). Le domaine de calcul est décrit par la figure 2.1, où

$$\Gamma_\infty^- = \{\mathbf{x} \in I\!R^d \mid \mathbf{x} \in \Gamma_\infty, \mathbf{u}_\infty \cdot \mathbf{n} < 0\}$$
$$\Gamma_\infty^+ = \{\mathbf{x} \in I\!R^d \mid \mathbf{x} \in \Gamma_\infty, \mathbf{u}_\infty \cdot \mathbf{n} \geq 0\}.$$

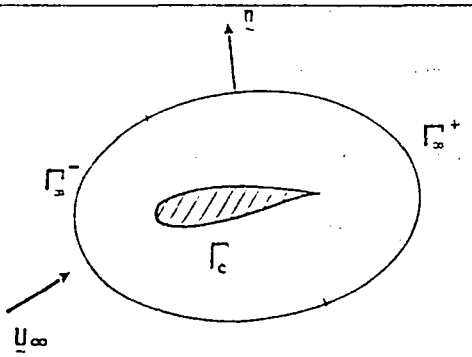



Figure 2.1

Sur la frontière amont $\Gamma_\infty^-$, on impose des conditions de type Dirichlet à toutes les variables:

$$\mathbf{u} = \mathbf{u}_\infty = \begin{pmatrix} \cos\alpha \\ \sin\alpha \end{pmatrix}, \quad \alpha = \text{angle d'attaque } (d = 2) \tag{2.6}$$

$$\rho = \rho_\infty = 1 \Rightarrow \sigma = \ln\rho = 0 \tag{2.7}$$

$$T = T_\infty = \frac{1}{\gamma(\gamma-1)M_\infty^2}. \tag{2.8}$$

Sur la frontière aval $\Gamma_\infty^+$, on impose des conditions de type Neumann à la vitesse et à la température (on ne doit pas imposer des conditions à toutes les variables pour que le problème soit bien posé; cf. [H1]):

$$\frac{\partial \mathbf{u}}{\partial \mathbf{n}} = 0, \quad \frac{\partial T}{\partial \mathbf{n}} = 0. \tag{2.9}$$

Finalement on impose sur le corps une condition d'adhérence et une condition de paroi isotherme:

$$\mathbf{u} = \mathbf{0} \tag{2.10}$$

$$T = T_C = T_\infty\left(1 + \frac{(\gamma-1)}{2}M_\infty^2\right), \tag{2.11}$$

avec $T_C$ la température d'arrêt. Dans (2.6)–(2.11), $\mathbf{u}_\infty$, $T_\infty$ et $M_\infty$ désignent la vitesse, la température et le Mach pour l'écoulement uniforme correspondant (celui qui prévaut loin de $\Gamma_C$).

Finalement, il faut ajouter aux équations (2.2)–(2.4), les conditions initiales suivantes:

$$\sigma(\mathbf{x},0) = \sigma_0(\mathbf{x}) = \ln\rho_0(\mathbf{x}), \quad \mathbf{u}(\mathbf{x},0) = \mathbf{u}_0(\mathbf{x}), \quad T(\mathbf{x},0) = T_0(\mathbf{x}). \tag{2.12}$$

Classiquement, un cas de calcul correspond à la donnée du nombre de Mach à l'infini $M_\infty$, du nombre de Reynolds Re, de l'angle d'incidence $\alpha$ (et, pour les écoulements tridimensionnels, de l'angle de dérapage $\beta$).

## 2.2. Discrétisation en temps

Dans beaucoup de cas on s'intéresse à des écoulements stationnaires, et l'on pourrait éliminer dans les équations de la mécanique des fluides les dérivées par rapport au temps. Cependant, même dans ce cas, la marche en temps reste l'approche naturelle pour résoudre ces équations car la résolution directe des équations stationnaires, bien qu'elle ne soit pas impossible, demeure très difficile.

Si on pose $\mathbf{U} = (\sigma, \mathbf{u}, T)^T$, le problème instationnaire à résoudre est de la forme:

$$\frac{\partial \mathbf{U}}{\partial t} + \mathbf{G}(\mathbf{U}) = \mathbf{0}. \tag{2.13}$$

On introduit un schéma totalement implicite (Euler rétrograde), de degré un en temps. Soit $\Delta t$ $(> 0)$ le pas de discrétisation en temps. D'après (2.12) on se donne comme solution initiale la valeur de $\mathbf{U}$ à l'instant $t = 0$:

$$\sigma^0 = \sigma_0 = \ln \rho_0, \quad \mathbf{u}^0 = \mathbf{u}_0, \quad T^0 = T_0.$$

Soit $\mathbf{U}^n = (\sigma^n, \mathbf{u}^n, T^n)^T$ la solution du système à l'instant $t = t^n$. A l'instant $t^{n+1} = t^n + \Delta t$, on approche $\mathbf{U}^{n+1}$ en résolvant:

$$\frac{\mathbf{U}^{n+1} - \mathbf{U}^n}{\Delta t} + \mathbf{G}(\mathbf{U}^{n+1}) = \mathbf{0}. \tag{2.14}$$

Cela nécessite la résolution d'un système non linéaire, ce qui est très coûteux. Par contre, le schéma défini par (2.14) est en général inconditionnellement stable, c'est-à-dire stable pour tout pas de temps. Ceci dit, en pratique, le coût de la résolution du système non linéaire impose des restrictions sur les choix raisonnables du pas de temps.

Quand on calcule la solution stationnaire, on peut aussi utiliser un pas de temps local.

On peut bien sûr utiliser d'autres schémas de discrétisation en temps: des schémas totalement implicites d'ordres plus élévés (par exemple, le schéma de Gear qui a besoin de la solution à deux instants successifs, cf. [B5]), ou des schémas basés sur une technique de décomposition d'opérateurs, de façon à découpler les problèmes de la convection et de la diffusion (par exemple, cf. [R1] par un $\theta$-schéma).

## 2.3. Une formulation variationnelle

Nous considérons la formulation non conservative des équations de Navier-Stokes (2.2)–(2.4). A chaque pas de temps du schéma (2.14) nous devons résoudre un problème non linéaire de la forme:

$$\alpha\sigma + \nabla \cdot \mathbf{u} + \mathbf{u} \cdot \nabla\sigma = g \tag{2.15}$$

$$\alpha\mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u} + (\gamma-1)(T\nabla\sigma + \nabla T) - \frac{e^{-\sigma}}{\mathrm{Re}}\Big(\Delta\mathbf{u} + \frac{1}{3}\nabla(\nabla\cdot\mathbf{u})\Big) = \mathbf{f} \tag{2.16}$$

$$\alpha T + \mathbf{u}\cdot\nabla T + (\gamma-1)T(\nabla\cdot\mathbf{u}) - \frac{e^{-\sigma}}{\mathrm{Re}}\Big(\frac{\gamma}{\mathrm{Pr}}\Delta T + F(\nabla\mathbf{u})\Big) = h \tag{2.17}$$

où $F$ est donné par (2.5), $\alpha$ est un paramètre positif et $g, \mathbf{f}$ et $h$ sont des fonctions données. Les variables $\sigma$, $\mathbf{u}$ et $T$ satisfont les conditions aux limites (2.6)–(2.11).

Nous introduisons les espaces fonctionnels de type Sobolev suivants:

$$R_r = \{\phi \in H^1(\Omega) \mid \phi = r \text{ sur } \Gamma_\infty^-\}$$
$$V_{\mathbf{z}} = \{\mathbf{v} \in (H^1(\Omega))^d \mid \mathbf{v} = \mathbf{z} \text{ sur } \Gamma_c \cup \Gamma_\infty^-\}$$
$$W_s = \{\theta \in H^1(\Omega) \mid \theta = s \text{ sur } \Gamma_c \cup \Gamma_\infty^-\}.$$

Si $r$ (respectivement $\mathbf{z}$, $s$) est assez régulière, alors $R_r$ (réspectivement $V_{\mathbf{z}}$, $W_s$) est non vide.

Une formulation variationnelle équivalente des équations (2.15)–(2.17) est:

$$\alpha\int_\Omega \sigma\phi\, d\mathbf{x} + \int_\Omega (\nabla\cdot\mathbf{u})\phi\, d\mathbf{x} + \int_\Omega (\mathbf{u}\cdot\nabla\sigma)\phi\, d\mathbf{x} = \int_\Omega g\phi\, d\mathbf{x}$$

$$\alpha\int_\Omega \mathbf{u}\cdot\mathbf{v}\, d\mathbf{x} + \int_\Omega (\mathbf{u}\cdot\nabla)\mathbf{u}\cdot\mathbf{v}\, d\mathbf{x} + (\gamma-1)\int_\Omega (T\nabla\sigma + \nabla T)\cdot\mathbf{v}\, d\mathbf{x}$$
$$- \int_\Omega \frac{e^{-\sigma}}{\mathrm{Re}}\Big(\Delta\mathbf{u} + \frac{1}{3}\nabla(\nabla\cdot\mathbf{u})\Big)\cdot\mathbf{v}\, d\mathbf{x} = \int_\Omega \mathbf{f}\cdot\mathbf{v}\, d\mathbf{x}$$

$$\alpha\int_\Omega T\theta d\mathbf{x} + \int_\Omega (\mathbf{u}\cdot\nabla T)\theta d\mathbf{x} + (\gamma-1)\int_\Omega T(\nabla\cdot\mathbf{u})\theta d\mathbf{x}$$
$$- \int_\Omega \frac{e^{-\sigma}}{\mathrm{Re}}\Big(\frac{\gamma}{\mathrm{Pr}}\Delta T + F(\nabla\mathbf{u})\Big)\theta d\mathbf{x} = \int_\Omega h\theta\, d\mathbf{x}$$

$$\forall\ (\phi, \mathbf{v}, \theta) \in R_0 \times V_0 \times W_0,\ (\sigma, \mathbf{u}, T) \in R_r \times V_{\mathbf{z}} \times W_s,$$

où $r$, $\mathbf{z}$ et $s$ sont choisies telles qu'elles satisfont les conditions aux limites (2.6)–(2.11).

Les conditions aux limites (2.9) sont traitées au sens faible, i.e., elles sont incluses dans la formulation (à chaque fois qu'on fait une intégration par parties, on les utilise pour négliger certains termes).

## 2.4. Discrétisation en espace par la méthode des éléments finis

L'espace $\Omega$ est le domaine borné de $\mathbb{R}^d$ (avec $d = 2, 3$) occupé par le fluide. On désigne par $\Omega_h$ une approximation polyédrique (polygonale si $d = 2$) de ce domaine. Soit $\mathcal{T}_h$ une discrétisation de $\Omega_h$ en éléments finis, i.e. $\mathcal{T}_h$ est un ensemble d'éléments $T$ ($T$ étant un triangle si $d = 2$ ou un tétraèdre si $d = 3$) recouvrant le domaine $\Omega_h$ (i.e. $\overline{\Omega}_h = \cup T$) tels

que si on prend deux éléments quelconques de $\mathcal{T}_h$, alors soit leur intersection est vide, soit ils ont en commun un sommet (nœud de la triangulation), ou une arête, ou (dans le cas de $d = 3$) une face.

Soit $\mathcal{P}_k$ l'espace des polynômes en $d$ variables de degré inférieur ou égal à $k$.

On définit les espaces discrets suivants:

$$R_{rh} = \{\phi_h \in C^0(\overline{\Omega}_h) : \phi_h|_T \in \mathcal{P}_1 \ \ \forall T \in \mathcal{T}_h, \quad \phi_h = r_h \text{ sur } \Gamma_\infty^-\} \tag{2.18}$$

$$W_{sh} = \{\theta_h \in C^0(\overline{\Omega}_h) : \theta_h|_T \in \mathcal{P}_1 \ \ \forall T \in \mathcal{T}_h, \quad \theta_h = s_h \text{ sur } \Gamma_c \cup \Gamma_\infty^-\}, \tag{2.19}$$

où $r_h$ et $s_h$ sont des approximations convenables de $r$ et de $s$ respectivement.

Soit maintenant la triangulation $\mathcal{T}_{h/2}$ obtenue à partir de la triangulation $\mathcal{T}_h$ par subdivision de chaque élément $T \in \mathcal{T}_h$ en $l$ sous-éléments ($l = 4$ pour des triangles et 8 pour des tétraèdres) obtenus en joignant les points milieux de chaque arête, comme on peut le voir sur la figure 2.2 dans le cas $d = 2$ (cf. Bercovier–Pironneau [B6]).


$T_h \in \mathcal{T}_h$ $\qquad$ $T_i \in \mathcal{T}_{h/2}$, $i = 1, \ldots, 4$

Figure 2.2

On peut alors définir l'espace suivant:

$$V_{zh} = \{\mathbf{v}_h \in (C^0(\overline{\Omega}_h))^d : \mathbf{v}_h|_T \in (\mathcal{P}_1)^d \ \ \forall T \in \mathcal{T}_{h/2}, \quad \mathbf{v}_h = \mathbf{z}_h \text{ sur } \Gamma_c \cup \Gamma_\infty^-\}, \tag{2.20}$$

où $\mathbf{z}_h$ est une approximation convenable de $\mathbf{z}$.

Pour les cas de calcul considérés ici, on se place toujours dans $\mathbb{R}^2$ ($d = 2$).

Le choix des éléments finis utilisés pour la simulation de l'écoulement d'un fluide compressible est basé sur les résultats obtenus par Bristeau et al. [B5].

Il est bien connu que pour les équations de Navier–Stokes incompressible, la pression et la vitesse ne peuvent pas être approchées indépendamment (cf. [G1]), mais les espaces d'approximation doivent satisfaire une condition de "*compatibilité*", appelée couramment condition "*Inf–Sup*" (obtenue paralellement par Brezzi et Babuška; cf. [B7] et [B8]).

Dans [B5] on trouve des résultats numériques montrant que dans le cas d'un schéma centré pour la résolution des équations de Navier–Stokes modélisant un fluide compressible,

l'utilisation des espaces d'éléments finis "compatibles" est appropriée là aussi. On ne peut pas se contenter d'utiliser les mêmes espaces d'approximation pour les différentes variables, même en raffinant le maillage utilisé.

Enfin, quelques résultats théoriques concernant cette condition "*Inf-Sup*" ont été démontrés dans le cas de problèmes modèles simplifiés pour les fluides stationnaires adiabatiques par Pironneau–Rappaz [P2] et par Fortin–Soulaimani [F1].

Pour l'utilisation d'éléments finis "$P_1 - P_1 +$ *bulle*" se référer aux travaux de G. Rogé [R1].

Après discrétisation en temps et en espace des équations de Navier–Stokes (2.2)–(2.4), on obtient bien *un système non linéaire de dimension finie* (cf. §2.5) que nous résolvons par la méthode de GMRES, décrite dans la section §3. Pour préconditionner cette méthode on utilise le fait que la matrice jacobienne associée au système non linéaire à résoudre est non symétrique (cf. annexe 1), aspect qui sera consideré dans les sections §4 et §5.

## 2.5. Définition du systéme non linéaire à résoudre

Comme dans §2.4, on fixe la dimension $d$ de l'espace physique égal à 2 (i.e., $\Omega$, le domaine occupé par le fluide, est inclus dans $\mathbb{R}^2$).

Comme dans §2.2, on notera avec un indice $n$ les différentes variables à l'instant $t = t^n$; par exemple $(\mathbf{u}^n)^T = (u^n, v^n)$ est la vitesse du fluide au $n$-ième pas de temps. Les variables *non indicées* vont indiquer soit l'inconnue du problème *continu*, soit l'inconnue à l'instant $t = t^{n+1} = t^n + \Delta t$ (quand la notation sera ambiguë, on signalera explicitement l'indice $n+1$). On notera par $\alpha$ l'inverse du pas de temps: $\alpha = 1/\Delta t$.

Si $\mathbf{U}^T = (\sigma, \mathbf{u}, T)$ est la solution du problème (2.15)–(2.17), on notera par $H_i$ les différents termes du membre de gauche qui composent le dit système, i.e.

$$H_1(\mathbf{U}) = \alpha\sigma + u\frac{\partial\sigma}{\partial x} + v\frac{\partial\sigma}{\partial y} + \left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right) \tag{2.21}$$

$$H_2(\mathbf{U}) = \alpha u + \left(u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}\right) + (\gamma - 1)\left(T\frac{\partial\sigma}{\partial x} + \frac{\partial T}{\partial x}\right) - \frac{e^{-\sigma}}{\mathrm{Re}}\left(\Delta u + \frac{1}{3}\frac{\partial}{\partial x}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right)\right) \tag{2.22}$$

$$H_3(\mathbf{U}) = \alpha v + \left(u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y}\right) + (\gamma - 1)\left(T\frac{\partial\sigma}{\partial y} + \frac{\partial T}{\partial y}\right) - \frac{e^{-\sigma}}{\mathrm{Re}}\left(\Delta v + \frac{1}{3}\frac{\partial}{\partial y}\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right)\right) \tag{2.23}$$

$$H_4(\mathbf{U}) = \alpha T + \left(u\frac{\partial T}{\partial x} + v\frac{\partial T}{\partial y}\right) + (\gamma - 1)T\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right)$$

$$-\frac{e^{-\sigma}}{\mathrm{Re}}\Big(\frac{\gamma}{\mathrm{Pr}}\Delta T + F(\nabla \mathbf{u})\Big) \tag{2.24}$$

Après avoir fait notre discrétisation en espace, on dira qu'un nœud $i$ appartient à la triangulation $\mathcal{T}_h$ (resp. à $\mathcal{T}_{h/2}$) et on notera $i \in \mathcal{T}_h$ (resp. $i \in \mathcal{T}_{h/2}$) si $i$ est le sommet d'un triangle $T$ appartenant à $\mathcal{T}_h$ (resp. à $\mathcal{T}_{h/2}$). On notera de 1 jusqu'à $N_1$ les nœuds associés à la triangulation $\mathcal{T}_h$, et de $N_1 + 1$ à $N_3 = N_1 + N_2$ les nœuds associés à la triangulation $\mathcal{T}_{h/2}$ qui *ne sont pas de nœuds* de la triangulation $\mathcal{T}_h$.

Soit $N_{\sigma_i}$ (resp. $N_{T_i}$) la fonction de base associée à la variable $\sigma$ (resp. $T$) et au nœud $i$ (avec $i \in \mathcal{T}_h$). Elle appartient à $R_{rh}$ (resp. à $W_{sh}$) et on a $N_{\sigma_i}(j) = N_{T_i}(j) = \delta_{ij}$, pour tout nœud $j$ appartenant à la triangulation grossière $\mathcal{T}_h$.

D'une façon analogue on définit les fonctions de base $N_{u_i}$ et $N_{v_i}$ associées aux composantes de la vitesse et au nœud $i$, avec $i$ un nœud de la triangulation $\mathcal{T}_{h/2}$. $(N_{u_i}, N_{v_j})^T$ appartient à $V_{2h}$ et on vérifie $N_{u_i}(j) = N_{v_i}(j) = \delta_{ij}$, pour tout nœud $j$ appartenant à la triangulation fine $\mathcal{T}_{h/2}$.

Bien que la notation soit équivalente, il ne faut pas confondre entre une fonction base associée à l'une des compossantes de la vitesse et une fonction base associée à $\sigma$ ou à $T$, *même si elles sont associées au même nœud.* On peut voir schématiquement dans la figure 2.3 le support associé à $N_{\sigma_i}$ (resp. à $N_{T_i}$) et celui associé à $N_{u_i}$ (resp. à $N_{v_i}$), avec $i$ un nœud de la triangulation $\mathcal{T}_h$.

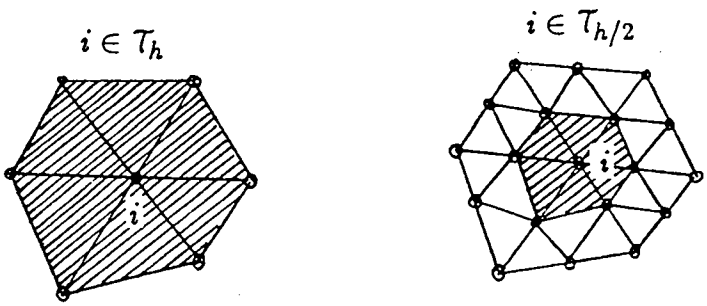



Figure 2.3

Dans la discrétisation du problème (2.15)–(2.17), nous utilisons le groupement naturel des inconnues par nœud. Donc si $\mathbf{U}_i$ est la variable (discrète) associée au nœud $i$, elle peut s'écrire:

$$\mathbf{U}_i = \begin{pmatrix} \sigma_i N_{\sigma_i} \\ u_i N_{u_i} \\ v_i N_{v_i} \\ T_i N_{T_i} \end{pmatrix} \quad \text{pour } i \in \mathcal{T}_h \qquad \text{et} \qquad \mathbf{U}_i = \begin{pmatrix} u_i N_{u_i} \\ v_i N_{v_i} \end{pmatrix} \quad \text{pour } i \in \mathcal{T}_{h/2}, i \notin \mathcal{T}_h.$$

Alors, une variable du système complet est noté par $\mathbf{U}$, où $\mathbf{U}^T = (\mathbf{U}_1^T, \ldots, \mathbf{U}_{N_1}^T, \mathbf{U}_{N_1+1}^T, \ldots, \mathbf{U}_{N_3}^T)$.

Après la discrétisation en espace, à chaque pas de temps il faut calculer les inconnues

$$\{\sigma_i, T_i \mid i \in \mathcal{T}_h\} \cup \{u_i, v_i \mid i \in \mathcal{T}_{h/2}\}$$

en résolvant le système non linéaire (de dimension finie):

$$\mathbf{F}(\mathbf{U}) = \mathbf{0},$$

avec

$$\mathbf{F}_i = \begin{pmatrix} f_1^{(i)} \\ f_2^{(i)} \\ f_3^{(i)} \\ f_4^{(i)} \end{pmatrix} = \begin{pmatrix} \int_\Omega H_1(\mathbf{U}) N_{\sigma_i} dx \\ \int_\Omega H_2(\mathbf{U}) N_{u_i} dx \\ \int_\Omega H_3(\mathbf{U}) N_{v_i} dx \\ \int_\Omega H_4(\mathbf{U}) N_{T_i} dx \end{pmatrix} - \begin{pmatrix} \alpha \sigma_i^n \int_\Omega N_{\sigma_i} dx \\ \alpha u_i^n \int_\Omega N_{u_i} dx \\ \alpha v_i^n \int_\Omega N_{v_i} dx \\ \alpha T_i^n \int_\Omega N_{T_i} dx \end{pmatrix} \quad \text{si} \quad i \in \mathcal{T}_h \qquad (2.25)$$

et

$$\mathbf{F}_i = \begin{pmatrix} f_1^{(i)} \\ f_2^{(i)} \end{pmatrix} = \begin{pmatrix} \mathrm{R} \\ \int_\Omega H_3(\mathbf{U}) N_{u_i} dx \\ \int_\Omega H_3(\mathbf{U}) N_{v_i} dx \end{pmatrix} - \begin{pmatrix} \alpha u_i^n \int_\Omega N_{u_i} dx \\ \alpha v_i^n \int_\Omega N_{v_i} dx \end{pmatrix} \quad \text{si} \quad i \in \mathcal{T}_{h/2}, i \notin \mathcal{T}_h; \qquad (2.26)$$

$H_1, \ldots, H_4$ données par (2.21)–(2.24).

L'intégration numérique des polynômes linéaires et quadratiques a été calculé de façon exacte à l'aide des formules suivantes:

$$\int_T p(\mathrm{x}) dx = \frac{\mathrm{aire}(T)}{3} \sum_{i=1}^{3} p(M_i) \qquad \text{si } p \in P_1$$

$$\int_T q(\mathrm{x}) dx = \frac{\mathrm{aire}(T)}{3} \sum_{i=1}^{3} q(m_i) \qquad \text{si } q \in P_2,$$

où le point $M_i$ est un sommet du triangle $T$ et le point $m_i$ est le point milieux d'une arête du dit triangle, pour $1 \leq i \leq 3$. Les fonctions non linéaires ont été approchées par les premiers termes du polynôme de Taylor associé, pour se reporter ensuite à l'un des cas précedents.

## 3. La méthode de GMRES pour la résolution des systèmes non linéaires

Après une brève introduction sur les méthodes des sous-espaces de Krylov, dans la présente section nous explicitons en détail la méthode de GMRES, due à Saad et Schultz (cf. [S1]) dans le cas linéaire, et à Brown et Saad (cf. [B3]) dans le cas général. Cette méthode est spécialement adaptée pour la résolution des problèmes non linéaires. Elle est robuste et d'usage simple, ce qui permet son utilisation sur une large palette de problèmes.

### 3.1. Quelques rappels sur les méthodes de sous-espaces de Krylov

La résolution des grands systèmes linéaires creux a toujours été un sujet de recherche qui a attiré l'attention de nombreux chercheurs, d'abord pour l'intérêt que ce sujet comporte en lui même, et ensuite, comme étant un outil indispensable pour la résolution des problèmes non linéaires.

On veut résoudre le système linéaire:

$$A\mathbf{x} = \mathbf{b} \tag{3.1}$$

où $A$ est une matrice $N \times N$ inversible, et où $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ et $\mathbf{b} = (b_1, b_2, \ldots, b_N)$ désignent respectivement l'inconnue et le second membre du système.

On peut classer la difficulté du problème en quatre catégories:

(1) $A$ est symétrique définie positive;

(2) $A$ est symétrique indéfinie;

(3) $A$ est non symétrique positive réelle, c.à.d., sa partie symétrique $(A + A^T)/2$ est définie positive;

(4) $A$ est non symétrique non positive réelle, c.à.d. sa partie symétrique est non définie. On dit alors que $A$ est non définie.

De nos jours, les problèmes de la première classe sont bien compris, et en conséquence, les autres classes concentrent une bonne partie de la recherche numérique en algèbre linéaire. Les travaux de Saad [S3] et de Chan–Jackson [C1] (pour ne citer que quelques uns) concernent les méthodes de sous-espaces de Krylov considérant, en particulier, le cas des matrices $A$ non symétriques. L'idée commune à toutes ces méthodes est que la solution approchée $\mathbf{x}_k$ appartient à l'espace affine $\mathbf{x}_0 + K_k$, où $K_k$, défini ci-dessous (voir définition 3.1), est le *sous-espace de Krylov* de dimension $k$ associé à la matrice $A$ et au résidu initial $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$.

**Définition 3.1.** *On appelle sous-espace de Krylov $K_k$ associé au vecteur $\mathbf{r}$ (non nul) et à la matrice $A$ (non singulière) le sous-espace de $\mathbb{R}^N$ de dimension $k$ engendré par les vecteurs $\mathbf{r}, A\mathbf{r}, \ldots, A^{k-1}\mathbf{r}$.* ∎

Leur principe est de construire une solution approchée $\mathbf{x}_k$ telle que le vecteur résidu $\mathbf{r}_k$ associé à $\mathbf{x}_k$ soit orthogonale à quelque sous-espace $L_k$. Dans la pratique, $L_k$ peut être différent de $K_k$ (cf. [S4]). On peut regarder ce processus comme une méthode de projection de Galerkin sur $K_k$, où le problème original (de dimension $N$) est remplacé par un problème de dimension $k$, associé à l'opérateur linéaire $A_k = P_k A|_{K_k}$, où $P_k$ est l'opérateur de projection sur $K_k$ selon la direction $L_k$.

Un bon nombre des méthodes sur les sous-espaces de Krylov developpées dans la littérature supposent que la partie symétrique de la matrice $A$ est définie positive, i.e. qu'il s'agit de problèmes de la classe (3). Comme nous sommes principalement interessés par les problèmes de la classe (4), nous allons décrire en détail la méthode de GMRES, qui a été spécialement conçue pour cette sorte de problèmes, et qui, en utilisant ce savoir-faire, a été généralisée pour la rendre apte à la résolution des problèmes non linéaires. C'est justement cela ce qui nous interesse, comme on l'a déjà dit dans la section §2.

### 3.2. La méthode de GMRES linéaire

Dans ce paragraphe on présente la méthode de GMRES (Generalised Minimal RESidual) proposée par Y. Saad et M. Schultz (voir [S1]) pour la résolution d'un système linéaire quelconque.

Une propriété remarquable de GMRES est que, en arithmétique exacte, la méthode donne la solution en un nombre fini de pas. En plus, on peut l'étendre avec plusieurs avantages, pour résoudre des systèmes non linéaires (cf. §3.3).

L'algorithme de GMRES est essentiellement une méthode des résidus minima: il consiste à minimiser, à chaque étape, la norme euclidienne du résidu dans un certain sous-espace affine de $\mathbb{R}^N$. Plus précisément, si l'inconnue $\mathbf{x}$ est mise sous la forme $\mathbf{x}_0 + \mathbf{z}$ où $\mathbf{x}_0$ est une première estimation de $\mathbf{x}$ et où $\mathbf{z}$, la nouvelle inconnue, appartient à un sous-espace $K_k$ convenablement choisi, GMRES cherche $\mathbf{z}_k \in K_k$ tel que:

$$\|\mathbf{b} - A(\mathbf{x}_0 + \mathbf{z}_k)\|_2 = \min_{\mathbf{z} \in K_k} \|\mathbf{b} - A(\mathbf{x}_0 + \mathbf{z})\|_2.$$

Pour $K_k$ on choisira le sous-espace de Krylov de dimension $k$ associé au résidu initial $\mathbf{r}_0$ $(= \mathbf{b} - A\mathbf{x}_0)$ et à la matrice $A$ (voir définition 3.1). Pour les détails présentés dans la suite sur la méthode de GMRES linéaire, voir [S1].

Dans l'algorithme de GMRES, on ne construira pas une base quelconque de l'espace de Krylov mais plutôt (moyennant l'algorithme d'Arnoldi, lequel utilise un processus d'orthogonalisation de type Gram-Schmidt) une base orthonormée de $\mathrm{K}_k$. On fera l'ortho-

normalisation par rapport à la norme euclidienne de $\mathbb{R}^N$

$$\|\mathbf{x}\|_2 = \left(\sum_{i=1}^{N} x_i^2\right)^{1/2}, \tag{3.2}$$

induite par le produit scalaire

$$(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{N} x_i y_i. \tag{3.3}$$

**Algorithme 3.1.** Algorithme d'Arnoldi

Etape 0 : Choisir $\mathbf{v}_1$ : $\|\mathbf{v}_1\|_2 = 1$.

Etape 1 : Pour $j = 1, 2, \ldots, k$ faire :

$$h_{ij} = (A\mathbf{v}_j, \mathbf{v}_i) \qquad i = 1, \ldots, j$$

$$\hat{\mathbf{v}}_{j+1} = A\mathbf{v}_j - \sum_{i=1}^{j} h_{ij}\mathbf{v}_i$$

$$h_{j+1,j} = \|\hat{\mathbf{v}}_{j+1}\|_2$$

$$\mathbf{v}_{j+1} = \hat{\mathbf{v}}_{j+1}/h_{j+1,j}. \quad \blacksquare$$

**Proposition 3.1.** *On se place dans le cas où $h_{j+1,j}$ est positif pour $j = 1, \ldots, k$. Alors, les vecteurs obtenus par l'algorithme d'Arnoldi $\mathbf{v}_1, \ldots, \mathbf{v}_k$ sont deux à deux orthogonaux de norme 1.*

Preuve: (Par récurrence sur $j$)

Soit $j \in [1, k-1]$, $k \geq 2$ donné. Montrons que

$$(\mathbf{v}_{j+1}, \mathbf{v}_i) = 0 \qquad \forall\ i = 1, \ldots, j. \tag{3.4}$$

Si $j = 1$, nous avons $\mathbf{v}_2 = \frac{1}{h_{21}}(A\mathbf{v}_1 - h_{11}\mathbf{v}_1)$ avec $h_{21} \neq 0$ (sinon $\mathbf{v}_2$ ne pourrait être construit et l'algorithme se serait terminé).

$$(\mathbf{v}_2, \mathbf{v}_1) = \frac{1}{h_{21}}\{(A\mathbf{v}_1, \mathbf{v}_1) - h_{11}(\mathbf{v}_1, \mathbf{v}_1)\} = \frac{1}{h_{21}}\{(A\mathbf{v}_1, \mathbf{v}_1) - (A\mathbf{v}_1, \mathbf{v}_1)\|\mathbf{v}_1\|_2^2\} = 0.$$

Supposons la relation (3.4) vraie jusqu'à $j - 1 \in [1, k-1]$. On fixe $i$ entre 1 et $j$. Pour $j$ nous avons:

$$(\mathbf{v}_{j+1}, \mathbf{v}_i) = \frac{1}{h_{j+1,j}}\left(A\mathbf{v}_j - \sum_{l=1}^{j} h_{lj}\mathbf{v}_l, \mathbf{v}_i\right)$$

$$= \frac{1}{h_{j+1,j}}(A\mathbf{v}_j, \mathbf{v}_i) - \sum_{l=1}^{j} \frac{h_{lj}}{h_{j+1,j}}(\mathbf{v}_l, \mathbf{v}_i).$$

Or l'hypothèse de récurrence entraîne que

$$\sum_{l=1}^{j} h_{lj}(\mathbf{v}_l, \mathbf{v}_i) = h_{ij}(\mathbf{v}_i, \mathbf{v}_i) = h_{ij},$$

*donc*

$$(\mathbf{v}_{j+1}, \mathbf{v}_i) = \frac{1}{h_{j+1,j}}\{(A\mathbf{v}_j, \mathbf{v}_i) - h_{ij}\} = 0,$$

puisque $h_{ij} = (A\mathbf{v}_j, \mathbf{v}_i)$. Par conséquent les vecteurs $\mathbf{v}_1, \ldots, \mathbf{v}_k$ sont deux à deux orthogonaux et, par construction, leur norme est 1. ∎

A la fin du processus on obtient une matrice $(k+1) \times k$, $\overline{H}_k$, à partir des $h_{ij}$ en posant:

$$(\overline{H}_k)_{ij} = \begin{cases} h_{ij} & 1 \leq j \leq k,\ 1 \leq i \leq j+1; \\ 0 & \text{ailleurs.} \end{cases}$$

Nous désignons par $H_k$ la matrice d'Hessenberg supérieure $k \times k$ obtenue à partir de $\overline{H}_k$ en supprimant sa dernière ligne, i.e.

$$\overline{H}_k = \begin{pmatrix} & & H_k & \\ 0 & \cdots & 0 & h_{k+1,k} \end{pmatrix}.$$

**Proposition 3.2.** *Les vecteurs $\mathbf{v}_1, \ldots, \mathbf{v}_k$ générés par l'algorithme d'Arnoldi forment une base du sous-espace de Krylov $K_k$ associé au vecteur initial $\mathbf{v}_1$ et à la matrice $A$, i.e.*

$$< \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k > = < \mathbf{v}_1, A\mathbf{v}_1, \ldots, A^{k-1}\mathbf{v}_1 > = K_k \tag{3.5}$$

Preuve:

(i) On montre $< \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_j > \subseteq < \mathbf{v}_1, A\mathbf{v}_1, \ldots, A^{j-1}\mathbf{v}_1 >$ pour $j = 1, \ldots, k$, par récurrence sur $j$. Pour $j = 1$, il n'y a rien à démontrer. Supposons donc $k \geq 2$, et supposons l'inclusion vérifiée pour $j$ dans l'intervalle $1 \leq j \leq k-1$. Soit $\mathbf{u} \in < \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{j+1} >$. Il s'écrit:

$$\mathbf{u} = \sum_{i=1}^{j} c_i \mathbf{v}_i + c_{j+1}\mathbf{v}_{j+1} \quad \text{pour certains } c_1, \ldots, c_{j+1},$$

$$= \sum_{i=1}^{j} b_i A^{i-1}\mathbf{v}_1 + c_{j+1}\mathbf{v}_{j+1} \text{ (par hypothèse de récurrence).}$$

Or

$$\mathbf{v}_{j+1} = \frac{\hat{\mathbf{v}}_{j+1}}{h_{j+1,j}} = \frac{1}{h_{j+1,j}}(A\mathbf{v}_j - \sum_{i=1}^{j} h_{ij}\mathbf{v}_i),$$

donc

$$\begin{aligned}\mathbf{u} &= \sum_{i=1}^{j} b_i A^{i-1}\mathbf{v}_1 + \frac{c_{j+1}}{h_{j+1,j}}[A\mathbf{v}_j - \sum_{i=1}^{j} h_{ij}\mathbf{v}_i] \\ &= \sum_{i=1}^{j} b_i A^{i-1}\mathbf{v}_1 + \frac{c_{j+1}}{h_{j+1,j}}\Big(A(\sum_{i=1}^{j} d_i A^{i-1}\mathbf{v}_1) - \sum_{i=1}^{j} k_{ij} A^{i-1}\mathbf{v}_1\Big) \\ &= \sum_{i=1}^{j+1} l_i A^{i-1}\mathbf{v}_1\end{aligned}$$

avec $l_i = \sum_{r=1}^{j} m_{ir} c_i$ ($m_{ir}$ connus), ce qui démontre (i).

(ii) Les calculs précédents prouvent aussi l'inclusion inverse puisque si on se donne $\mathbf{w} = \sum_{i=1}^{j} b_i A^{i-1}\mathbf{v}_1$, alors

$$\mathbf{w} = \sum_{i=1}^{j} c_i \mathbf{v}_i$$

où les $c_i$ sont les solutions du système linéaire

$$b_i = \sum_{r=1}^{j} m_{ir} c_i. \quad \blacksquare$$

**Proposition 3.3.** *Si $V_k$ est la matrice $N \times k$ dont les colonnes sont les vecteurs de la base orthonormale de $K_k$, $\mathbf{v}_1, \ldots, \mathbf{v}_k$, on a la relation:*

$$AV_k = V_{k+1}\overline{H}_k, \tag{3.6}$$

*et aussi*

$$H_k = V_k^T A V_k. \tag{3.7}$$

Preuve: (Par récurrence sur $k$)

Si $k = 1$ : $V_1 = \mathbf{v}_1$, $V_2 = [\mathbf{v}_1, \mathbf{v}_2]$ et $\overline{H}_1 = \begin{pmatrix} h_{11} \\ h_{21} \end{pmatrix}$, donc $V_2\overline{H}_1 = h_{11}\mathbf{v}_1 + h_{21}\mathbf{v}_2$.

Or $\mathbf{v}_2 = \frac{1}{h_{21}} A\mathbf{v}_1 - \frac{h_{11}}{h_{21}}\mathbf{v}_1$, ce qui donne

$$V_2\overline{H}_1 = h_{11}\mathbf{v}_1 + A\mathbf{v}_1 - h_{11}\mathbf{v}_1 = A\mathbf{v}_1 = AV_1$$

donc l'égalité (3.6) est vraie pour $k = 1$.

Supposons la relation (3.6) vraie pour $k$, i.e. $AV_k = V_{k+1}\overline{H}_k$. Pour $k+1$, nous avons:

$$V_{k+2} = [V_{k+1}, \mathbf{v}_{k+2}], \quad \overline{H}_{k+1} = \begin{pmatrix} & H_{k+1} & \\ 0 & \cdots \; 0 & h_{k+2,k+1} \end{pmatrix} = \begin{pmatrix} H_{k+1} \\ \mathbf{g}_{k+1}^T \end{pmatrix}$$

avec $\mathbf{g}_{k+1}^T = (0, \ldots, 0, h_{k+2,k+1}) \in I\!R^{k+1}$ et $H_{k+1} = [\overline{H}_k, \overline{\mathbf{h}}_k]$, où $\overline{\mathbf{h}}_k$, le $(k+1)$-ième vecteur colonne de $H_{k+1}$, est

$$\overline{\mathbf{h}}_k = \begin{pmatrix} h_{1,k+1} \\ \cdots \\ h_{k+1,k+1} \end{pmatrix}.$$

On peut écrire:

$$V_{k+2}\overline{H}_{k+1} = [V_{k+1}, \mathbf{v}_{k+2}]\begin{pmatrix} H_{k+1} \\ \mathbf{g}_{k+1}^T \end{pmatrix} = V_{k+1}H_{k+1} + \mathbf{v}_{k+2}\mathbf{g}_{k+1}^T.$$

On a

$$V_{k+1}H_{k+1} = V_{k+1}[\overline{H}_k, \overline{\mathbf{h}}_k] = [V_{k+1}\overline{H}_k, V_{k+1}\overline{\mathbf{h}}_k] = [AV_k, V_{k+1}\overline{\mathbf{h}}_k]$$

$$\mathbf{v}_{k+2}\mathbf{g}_{k+1}^T = [0, h_{k+2,k+1}\mathbf{v}_{k+2}],$$

donc

$$V_{k+2}\overline{H}_{k+1} = [AV_k, h_{k+2,k+1}\mathbf{v}_{k+2} + V_{k+1}\overline{\mathbf{h}}_k].$$

Montrons que

$$h_{k+2,k+1}\mathbf{v}_{k+2} + V_{k+1}\overline{\mathbf{h}}_k = A\mathbf{v}_{k+1}. \tag{3.8}$$

Comme

$$\mathbf{v}_{k+2} = \frac{1}{h_{k+2,k+1}}A\mathbf{v}_{k+1} - \sum_{i=1}^{k+1}\frac{h_{i,k+1}}{h_{k+2,k+1}}\mathbf{v}_i,$$

on a

$$h_{k+2,k+1}\mathbf{v}_{k+2} = \frac{h_{k+2,k+1}}{h_{k+2,k+1}}A\mathbf{v}_{k+1} - \sum_{i=1}^{k+1}\frac{h_{i,k+1}}{h_{k+2,k+1}}h_{k+2,k+1}\mathbf{v}_i$$

$$= A\mathbf{v}_{k+1} - \sum_{i=1}^{k+1}h_{i,k+1}\mathbf{v}_i.$$

D'autre part, $V_{k+1}\overline{\mathbf{h}}_k = \sum_{i=1}^{k+1}h_{i,k+1}\mathbf{v}_i$, ce qui implique (3.8). Par conséquent,

$$V_{k+2}\overline{H}_{k+1} = [AV_k, A\mathbf{v}_{k+1}] = AV_{k+1},$$

ce qui prouve la relation (3.6).

De ce qui précede on déduit que

$$AV_k = [V_k, \mathbf{v}_{k+1}]\begin{pmatrix} H_k \\ \mathbf{g}_k^T \end{pmatrix} = V_kH_k + \mathbf{v}_{k+1}\mathbf{g}_k^T.$$

Comme les colonnes de $V_{k+1}$ sont orthonormales on a

$$V_k^T v_{k+1}g_k^T = 0$$

et on obtient, alors, la relation (3.7). ∎

Ainsi nous voulons résoudre le problème: Trouver $\mathbf{z}_k \in K_k$ tel que

$$\|\mathbf{b} - A(\mathbf{x}_0 + \mathbf{z}_k)\|_2 = \min_{\mathbf{z}\in K_k} \|\mathbf{b} - A(\mathbf{x}_0 + \mathbf{z})\|_2 = \min_{\mathbf{z}\in K_k} \|\mathbf{r}_0 - A\mathbf{z}\|_2. \tag{3.9}$$

Cela revient à minimiser la fonctionnelle $J(\mathbf{y}) = \|\beta\mathbf{v}_1 - AV_k\mathbf{y}\|_2$ sur $\mathbb{R}^k$, où $\beta = \|\mathbf{r}_0\|_2$ et $\mathbf{v}_1 = \mathbf{r}_0/\beta$. En effet, grâce à la relation (3.5), tout $\mathbf{z} \in K_k$ s'écrit $\mathbf{z} = V_k\mathbf{y}$, pour un certain $\mathbf{y} \in \mathbb{R}^k$, et on a $\mathbf{r}_0 = \beta\mathbf{v}_1$ par définition.

D'après l'égalité (3.6) nous avons: $J(\mathbf{y}) = \|\beta\mathbf{v}_1 - V_{k+1}\overline{H}_k\mathbf{y}\|_2$. Or $\mathbf{v}_1 = V_{k+1}\mathbf{e}_1$, où $\mathbf{e}_1 = (1,0,\ldots,0)^T \in \mathbb{R}^{k+1}$, donc $J(\mathbf{y})$ s'écrit encore $\|V_{k+1}(\beta\mathbf{e}_1 - \overline{H}_k\mathbf{y})\|_2$. Comme les colonnes de $V_{k+1}$ sont orthonormales, on obtient finalement

$$J(\mathbf{y}) = \|\beta\mathbf{e}_1 - \overline{H}_k\mathbf{y}\|_2. \tag{3.10}$$

Ainsi pour $k$ donné nous pouvons construire la solution approchée

$$\mathbf{x}_k = \mathbf{x}_0 + V_k\mathbf{y}_k \qquad \text{où} \qquad J(\mathbf{y}_k) \leq J(\mathbf{y}) \quad \forall\ \mathbf{y} \in \mathbb{R}^k,$$

étant $J$ donnée par la formule (3.10).

Lorsque $k = N$ nous obtenons la solution exacte $\mathbf{x}$ de notre problème, parce qu'on minimise sur tout $\mathbb{R}^N$. On peut dire alors que la méthode de GMRES est essentiellement *une méthode directe* en ce sens qu'il existe $k_{\max} \leq N$ tel que, en l'absence d'erreurs d'arrondi, $\mathbf{z} = \mathbf{x} - \mathbf{x}_0$ appartienne à $K_{k_{\max}}$.

Maintenant nous pouvons donner l'algorithme complet.

**Algoritme 3.2.** Algorithme de GMRES linéaire

Etape 0: Choisir $\mathbf{x}_0$ et calculer $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$, $\mathbf{v}_1 = \mathbf{r}_0/\|\mathbf{r}_0\|_2$. Poser $k = 1$.

Etape 1: Faire:

$$h_{ik} = (A\mathbf{v}_k, \mathbf{v}_i) \qquad i = 1,\ldots,k$$
$$\hat{\mathbf{v}}_{k+1} = A\mathbf{v}_k - \textstyle\sum_{i=1}^{k} h_{ik}\mathbf{v}_i$$
$$h_{k+1,k} = \|\hat{\mathbf{v}}_{k+1}\|_2$$
$$\mathbf{v}_{k+1} = \hat{\mathbf{v}}_{k+1}/h_{k+1,k}.$$

Etape 2: Calculer la solution approchée $\mathbf{x}_k = \mathbf{x}_0 + V_k\mathbf{y}_k$, où $\mathbf{y}_k$ minimise $J$ (donnée par (3.10)) sur $\mathbb{R}^k$.

Si $\mathbf{x}_k$ est satisfaisant, s'arrêter. Sinon, faire $k = k + 1$ et retourner à l'étape 1. ∎

D'un point de vue pratique, on comprend facilement que, plus $k$ est grand, plus l'algorithme est coûteux, puisqu'il demande le stockage de $V_{k+1}$ et que le nombre de multiplications augmente comme $k^2N/2$. Pour remédier à cette difficulté, on peut transformer l'algorithme en une méthode itérative. On fixe un nombre $k_{\max}$. On construit l'espace de Krylov de dimension $k_{\max}$ associé à $\mathbf{v}_1$ et à $A$, et après la construction de la solution approchée $\mathbf{x}_{k_{\max}}$, on réinitialise l'algorithme avec $\mathbf{x}_{k_{\max}}$. Le nouvel algorithme sera appelé GMRES($k_{\max}$).

**Algoritme 3.3.** Algorithme de GMRES($k$) linéaire

Etape 0: Choisir $\mathbf{x}_0$ et calculer $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$, $\mathbf{v}_1 = \mathbf{r}_0/\|\mathbf{r}_0\|_2$.

Etape 1: Pour $j = 1, 2, \ldots, k$ faire:

$$h_{ij} = (A\mathbf{v}_j, \mathbf{v}_i) \qquad i = 1, \ldots, j$$
$$\hat{\mathbf{v}}_{j+1} = A\mathbf{v}_j - \textstyle\sum_{i=1}^{j} h_{ij}\mathbf{v}_i$$
$$h_{j+1,j} = \|\hat{\mathbf{v}}_{j+1}\|_2$$
$$\mathbf{v}_{j+1} = \hat{\mathbf{v}}_{j+1}/h_{j+1,j}.$$

Etape 2: Former la solution approchée $\mathbf{x}_k = \mathbf{x}_0 + V_k\mathbf{y}_k$, où $\mathbf{y}_k$ minimise $J$ (donnée par (3.10)) sur $\mathbb{R}^k$.

Etape 3: Calculer $\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k$. Si $\mathbf{r}_k$ est satisfaisant, s'arrêter. Sinon, faire $\mathbf{x}_0 = \mathbf{x}_k$, $\mathbf{v}_1 = \mathbf{r}_k/\|\mathbf{r}_k\|_2$, et aller à l'étape 1. ∎

Ayant présenté les deux versions de GMRES, la seule difficulté d'utilisation réside dans la minimisation de $J(\mathbf{y}) = \|\beta\mathbf{e}_1 - \overline{H}_k\mathbf{y}\|_2$; or minimiser $J(\mathbf{y})$ revient à résoudre un système linéaire triangulaire supérieur d'ordre $k+1$. En effet, l'idée est de factoriser $\overline{H}_k$ en $Q_kR_k$, où $Q_k$ est une matrice unitaire d'ordre $k+1$, et $R_k$ une matrice triangulaire supérieure d'ordre $(k+1) \times k$ dont la dernière ligne est nulle. Ces calculs sont simples grâce à la structure particulière de $\overline{H}_k$. On profite de la factorisation de $\overline{H}_{k-1}$ pour obtenir celle de $\overline{H}_k$. Cette manière de factoriser nous donne directement la norme résiduelle $\|\mathbf{r}_k\|_2$ de la solution approchée $\mathbf{x}_k$ *sans calculer* $\mathbf{x}_k$. On ne calcule $\mathbf{x}_k$ que lorsque l'algorithme est terminé.

Remarque 3.1. Pour rendre plus efficace GMRES($k$) on devrait inclure un processus capable de régler automatiquement le paramètre $k$. Ce processus doit permettre de réduire $k$ *sans changer la relation d'orthogonalité* entre les vecteurs de la base de Krylov (voir prop. 3.1). Le réglage automatique de $k$ permettrait alors de commencer l'algorithme avec un $k$ assez grand, et, selon un certain critère, de le réduire progressivement (voir [S3] dans le cas d'une méthode d'orthonormalisation incomplète). ∎

Remarque 3.2. Il est clair que si $k$ est grand, l'une des parties la plus coûteuse de

l'algorithme est l'orthonormalisation de chaque vecteur $\mathbf{v}_{j+1}$ vis à vis des vecteurs $\mathbf{v}_i$ $(i \leq j)$ déjà calculés. Une alternative possible consiste à orthogonaliser $\mathbf{v}_{j+1}$ seulement vis à vis des $p$ derniers vecteurs calculés, $\mathbf{v}_{j-p+1}, \ldots, \mathbf{v}_j$ (cf. [S3]). ∎

Remarque 3.3. Dans les algorithmes 3.2 et 3.3 on n'utilise la matrice $A$ du système linéaire à résoudre que pour calculer des produits matrice–vecteur. Cette proprieté aura des conséquences très agréables dans l'extension de la méthode de GMRES pour la résolution de systèmes non linéaires (cf. §3.3). Elle permet aussi, dans beaucoup de cas pratiques, de programmer une méthode *sans stockage* de la matrice $A$. Bien sûr, dans ce cas le temps de calcul sera plus important, mais on pourra traiter des systèmes plus grands. ∎

Jusqu'ici nous avons supposé que les vecteurs $\mathbf{v}_1, \ldots, \mathbf{v}_{k+1}$ pouvaient être construits par le processus d'Arnoldi, ce qui signifie que les $h_{j+1,j}$ sont non nuls pour $j = 1, \ldots, k$. Ceci n'est pas restrictif. On peut en effet prouver que l'algorithme dégénère si et seulement si on possède la solution exacte. D'une façon plus complète, on a:

**Proposition 3.4.** *La solution $\mathbf{x}_k$ obtenue au $k$-ième pas de l'algorithme GMRES linéaire est exacte si et seulement si l'une des conditions équivalentes suivantes est vérifiée:*

*(1) L'algorithme s'arrête au $k$-ième pas,*

*(2)* $\hat{\mathbf{v}}_{k+1} = \mathbf{0}$,

*(3)* $h_{k+1,k} = 0$,

*(4) Le degré du polynôme minimal associé au résidu initial $\mathbf{r}_0$ est égal à $k$.* ∎

Une conséquence de la proposition 3.4 est que l'algorithme GMRES($k$) ne dégénère jamais. GMRES($k$) serait un algorithme très efficace s'il convergeait toujours. Malheureusement, ce n'est pas le cas. Mais on sait que sous certaines conditions, $\|\mathbf{r}_k\|_2$ converge vers zéro, pour $k$ assez grand, comme l'indique la proposition suivante:

**Proposition 3.5.** *Si $A$ est diagonalisable, alors, pour $k$ assez grand, GMRES($k$) converge pour tout vecteur initial $x_0$.* ∎

Il est simple pourtant de construire un exemple où la méthode ne converge pas, en choisissant la partie symétrique de $A$ non définie positive.

Intuitivement, pour $k$ assez grand, GMRES($k$) devrait converger. Le cas trivial où $k = N$ est un exemple où la méthode converge en un seul pas. Ce n'est pas intéressant dans la pratique. Cependant, il est intéressant de noter que si la matrice du système à résoudre est "presque" définie positive réelle (i.e. si elle n'a qu'un petit nombre de valeurs propres dans le demi-plan gauche), alors, on n'a pas besoin d'un $k$ très grand pour que l'algorithme converge.

### 3.3. La méthode de GMRES non linéaire

La méthode de GMRES a été introduite par Y. Saad et M. Shultz ([S1]) pour résoudre des systèmes linéaires non symétriques. Wigton, Yu et Young ([W1], [D1]) ont généralisé cette méthode pour résoudre des problèmes non linéaires. Leurs travaux portaient sur une gamme étendue de problèmes liés à l'aérodynamique et fournissaient un premier algorithme de résolution de problèmes non linéaires comportant un certain nombre de paramètres à régler.

L'apport principal de Y. Saad et P. Brown (cf. [B3]) a été de reformuler cet algorithme dans le cadre des méthodes de Newton et d'automatiser le choix des paramètres. Pour plus de détails sur les résultats exposés dans la suite se référer à [B3].

A la différence de la méthode de Newton, la version non linéaire de GMRES ne nécessite pas le calcul explicite du jacobien qui est difficile à évaluer dans la plupart des cas. Par rapport aux méthodes de moindres carrés non linéaires couplées à des algorithmes de gradient conjugué, elle n'exige pas la connaissance exacte du gradient de la fonction coût, ce qui, pour le type de problèmes nous intéressant, représente une opération complexe et coûteuse. Récemment ces méthodes ont été appliquées aux équations d'Euler compressible et aux équations de Navier-Stokes compressible et incompressible (cf. [B2], [M1]).

Soit $\mathbf{F}$ une fonction non linéaire de $\mathbb{R}^N$ dans $\mathbb{R}^N$. On veut résoudre le système d'équations non linéaires

$$\mathbf{F}(\mathbf{u}) = \mathbf{0}. \tag{3.11}$$

La méthode de Newton appliquée à (3.11) peut être décrite ainsi:

**Algorithme 3.4.** Méthode de Newton

1. Soit $\mathbf{u}_0$ une solution approchée de (3.11).
2. Pour $n = 0, 1, 2, \ldots$ jusqu'à la convergence, faire:

$$\text{Résoudre} \qquad J_n \delta_n = -\mathbf{F}_n \tag{3.12}$$

où $J_n = J(\mathbf{u}_n) = \mathbf{F}'(\mathbf{u}_n)$ est le jacobien de la fonction $\mathbf{F}$, et $\mathbf{F}_n = \mathbf{F}(\mathbf{u}_n)$.

Mettre à jour $\mathbf{u}_{n+1} = \mathbf{u}_n + \delta_n$. Si on satisfait le critère de convergence, s'arrêter. Sinon, faire $n = n + 1$ et continuer la boucle. ∎

En grande dimension, on utilise fréquemment des méthodes itératives pour résoudre le système (3.12). Dans la méthode de GMRES non linéaire on le résout de façon approchée par une méthode de Krylov (plus précisément par la méthode de GMRES linéaire, cf. §3.2).

A l'itération $n$, on choisit une approximation initiale $\delta_n^0$ de la solution de (3.12). En posant $\delta_n = \delta_n^0 + \mathbf{z}_n$, on obtient le système équivalent

$$J_n \mathbf{z}_n = \mathbf{r}_n^0, \tag{3.13}$$

avec le résidu initial $\mathbf{r}_n^0 = -\mathbf{F}_n - J_n \delta_n^0$ . Si $K_k$ est le sous-espace de Krylov de dimension $k$ associé à $\mathbf{r}_n^0$ et à $J_n$: $K_k = < \mathbf{r}_n^0, J_n \mathbf{r}_n^0, \ldots, J_n^{k-1} \mathbf{r}_n^0 >$, la méthode de GMRES cherche une solution approchée

$$\delta_n^k = \delta_n^0 + \mathbf{z}_n^k \quad \text{avec} \quad \mathbf{z}_n^k \in K_k \tag{3.14}$$

qui minimise la norme euclidienne du résidu:

$$\|r_n^0 - J_n \mathbf{z}_n^k\|_2 = \min_{\mathbf{z} \in K_k} \|\mathbf{r}_n^0 - J_n \mathbf{z}\|_2. \tag{3.15}$$

L'algorithme présenté ci-dessous est une version non-linéaire de l'algorithme de GMRES. A chaque itération il génère des vecteurs orthonormaux $\mathbf{v}_i$, pour $i = 1, \ldots, k+1$, et construit le vecteur $\delta_n^k$ selon (3.14) et (3.15). Les $\mathbf{v}_i$ sont calculés de sorte qu'ils constituent une base orthonormale du sous-espace de Krylov $K_k$, où $\mathbf{v}_1$ est obtenu en normalisant $\mathbf{r}_n^0$.

**Algorithme 3.5.** Algorithme de GMRES non linéaire

Etape 0: Initialisation de la boucle Newton:

Choisir $\mathbf{u}_0$ et un vecteur initial $\delta_0^0$. Poser $n = 0$.

Etape 1: Choisir une tolérance $\epsilon_n$.

Définir $J_n = J(\mathbf{u}_n) = \mathbf{F}'(\mathbf{u}_n)$ le jacobien de la fonction $\mathbf{F}$ et calculer $\mathbf{F}_n = \mathbf{F}(\mathbf{u}_n)$. Définir $\mathbf{r}_n^0 = -\mathbf{F}_n - J_n \delta_n^0$, $\beta_n = \|\mathbf{r}_n^0\|_2$ et $\mathbf{v}_1 = \mathbf{r}_n^0 / \beta_n$. Poser $j = 1$.

Etape 2: Définir:

$$h_{ij} = (J_n \mathbf{v}_j, \mathbf{v}_i) \qquad i = 1, \ldots, j$$

$$\hat{\mathbf{v}}_{j+1} = J_n \mathbf{v}_j - \textstyle\sum_{i=1}^{j} h_{ij} \mathbf{v}_i$$

$$h_{j+1,j} = \|\hat{\mathbf{v}}_{j+1}\|_2$$

$$\mathbf{v}_{j+1} = \hat{\mathbf{v}}_{j+1} / h_{j+1,j}.$$

Calculer la norme résiduelle $\rho_j = \|\mathbf{F}_n + J_n \delta_n^j\|_2$ de la solution $\delta_n^j$ qu'on obtiendrait si on s'arrêtait ici.

Si $\rho_j \leq \epsilon_n$, poser $k = j$ et aller à l'étape 3. Autrement, poser $j = j + 1$ et recommencer l'étape 2.

Etape 3: Calcul de la solution de (3.12):

Définir la matrice $\overline{H}_k$ de dimension $(k+1) \times k$, dont les coefficients non nuls sont les valeurs calculées à l'étape 1: $h_{ij}$ $(1 \leq j \leq k;\ 1 \leq i \leq j+1)$. Définir aussi la matrice $V_k = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k]$.

Trouver le vecteur $\mathbf{y}_k$ qui minimise $J(\mathbf{y}) = \|\beta_n \mathbf{e}_1 - \overline{H}_k \mathbf{y}\|_2$ sur $\mathbb{R}^k$ (avec $\mathbf{e}_1 = (1, 0, \ldots, 0)^T \in \mathbb{R}^{k+1}$).

Calculer $\delta_n^k = \delta_n^0 + V_k \mathbf{y}_k$ et $\mathbf{u}_{n+1} = \mathbf{u}_n + \delta_n^k$.

Etape 4: Test d'arrêt:

Si $\mathbf{u}_{n+1}$ est une solution satisfaisante de (3.11), finir. Autrement, faire $\mathbf{u}_n = \mathbf{u}_{n+1}$, $n = n+1$, choisir $\delta_n^0$ et aller à l'étape 1. ∎

Les étapes 2 et 3 de l'algorithme ci-dessus constituent précisément la méthode de GMRES linéaire pour résoudre le système linéaire (3.12). En pratique, on choisit $\delta_n^0 = 0$, pour tout $n$ positif.

Chaque pas de l'algorithme est divisé en deux parties. La première partie est la construction d'une base orthonormale $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ du sous-espace de Krylov $K_k$. La deuxième partie est le calcul de la solution approchée $\delta_n^k \in \delta_n^0 + K_k$ vérifiant la relation (3.15).

L'un des aspects les plus importants des méthodes de Krylov est que l'on n'a jamais besoin d'avoir explicitement la matrice jacobienne. La seule chose demandée par l'algorithme est le calcul des produits matrice-vecteur $J_n \mathbf{v}$, qui peuvent être approchés par

$$J(\mathbf{u})\mathbf{v} \approx \frac{\mathbf{F}(\mathbf{u} + \sigma \mathbf{v}) - \mathbf{F}(\mathbf{u})}{\sigma} \tag{3.16}$$

où $\sigma$ est un scalaire choisi en fonction de $\mathbf{F}(\mathbf{u})$. Brown (voir [B4]) a donné des conditions suffisantes sur $\sigma$ pour assurer une convergence locale de l'algorithme de Newton.

La convergence globale de l'algorithme 3.5, en utilisant l'approximation (3.16), a été étudiée par Brown et Saad (cf. [B3]).

Cependant, comme dans le cas linéaire, l'algorithme décrit ci-dessus présente un problème d'implémentation pratique: quand $k$ augmente, le nombre de vecteurs à stocker augmente avec la même vitesse (de l'ordre de $k$) et le nombre de produits (dans la résolution du système linéaire) comme $k^2N/2$. Pour remédier à ce problème on peut utiliser l'algorithme itérativement, i.e. recommencer l'algorithme chaque $k_{\max}$ pas, où $k_{\max}$ est un paramètre entier choisi selon des critères de stockage et de temps de calcul. Bien sûr, il est possible qu'on atteigne $k = k_{\max}$ dans l'étape 1 et que $\rho_k$ soit encore plus grand que $\epsilon_n$. Dans ce cas-là on peut poser $\delta_n^0 = \delta_n^k$ et recommencer le processus, i.e. la méthode de Krylov. Par analogie avec le cas linéaire, on appellera cette méthode GMRES($k_{\max}$). On peut encore

simplifier d'avantage, et se contenter de la solution obtenue en minimisant le problème (3.15) dans un espace de Krylov de dimension $k_{\max}$. La convergence d'une telle procédure n'est pas toujours garantie, mais cela marche bien dans la pratique.

Le calcul approché du jacobien moyennant la formule (3.16) indique que, en fait, GMRES ne résout pas le système (3.12) mais une approximation de celui-ci. Donc, la méthode de GMRES non linéaire appartienne aux *méthodes de Newton inexactes* (cf. [D4]).

Il est connu que la méthode de Newton peut ne pas converger si la solution initiale $\mathbf{u}_0$ est très éloignée de la solution exacte de (3.11). Pour obtenir une plus grande fiabilité de la méthode de GMRES, on utilise une procédure de "backtracking" qui nous assure que la nouvelle approximation $\mathbf{u}_{n+1}$ de la solution est effectivement "*acceptable*", i.e. qu'elle satisfait

$$\|\mathbf{F}(\mathbf{u}_{n+1})\|_2 \leq \|\mathbf{F}(\mathbf{u}_n)\|_2. \tag{3.17}$$

A priori, la résolution de (3.15) ne suffit pas pour garantir cette inégalité parce qu'on ne minimise que sur un sous-espace de $\mathbb{R}^N$.

**Définition 3.2.** *Soit $f: \mathbb{R}^N \to \mathbb{R}$ une fonction. On dira que $\mathbf{p} \in \mathbb{R}^N$ est une direction de descente locale de $f$ au point $\mathbf{u}$ s'il existe un $\lambda^*$ tel que*

$$f(\mathbf{u} + \lambda\mathbf{p}) < f(\mathbf{u}) \qquad \forall\ \lambda \in (0, \lambda^*].$$

On peut voir (cf. [D3]) que ceci équivaut à demander que

$$(\nabla f(\mathbf{u}), \mathbf{p}) < 0,$$

$(\cdot,\cdot)$ donné par (3.3).

Posons

$$f(\mathbf{u}) = \frac{1}{2}\mathbf{F}(\mathbf{u})^T\mathbf{F}(\mathbf{u});$$

Brown et Saad [B3] ont démontré que si $J_n$ est non singulière et si le vecteur $\delta_n^k$ donné par la méthode GMRES($k$) est non nul, alors $\delta_n^k$ est une direction de descente pour $f$ au point $\mathbf{u}_n$. Puisque $\delta_n^k$ est une direction de descente, il nous faut déterminer un pas $\lambda$ tel que $\mathbf{u}_{n+1} = \mathbf{u}_n + \lambda\delta_n^k$ soit acceptable. Bien sûr, en prenant $\lambda$ assez petit on vérifie l'inégalité (3.17); mais plus $\lambda$ est petit, plus lente est la convergence de l'algorithme. Pour éviter cela, on cherchera un $\lambda$ qui satisfasse les "*conditions $\alpha$ et $\beta$*", les dites conditions étant:

$$f(\mathbf{u}_n + \lambda\mathbf{p}) \leq f(\mathbf{u}_n) + \alpha\nabla f(\mathbf{u}_n)^T\mathbf{p} \tag{3.18}$$

$$f(\mathbf{u}_n + \lambda\mathbf{p}) \geq f(\mathbf{u}_n) + \beta\nabla f(\mathbf{u}_n)^T\mathbf{p} \tag{3.19}$$

respectivement, avec $0 < \alpha < \beta < 1$. Dans notre cas, on peut démontrer que, pour $0 < \alpha < \beta < 1$ donnés, il existe $\lambda_s > \lambda_i > 0$ tels que $\mathbf{u}_{n+1} = \mathbf{u}_n + \lambda_n \delta_n^k$ satisfasse les conditions $\alpha$ et $\beta$ pour tout $\lambda_n$ appartenant à $(\lambda_i, \lambda_s)$. Un tel $\mathbf{u}_{n+1}$ est acceptable selon la définition ci-dessus. Si on procède par dichotomie pour la recherche de $\lambda_n$ (comme dans notre cas de calcul), il faut tenir compte du fait que chaque changement de $\lambda$ entraîne une nouvelle évaluation de $f$. Alors, un algorithme efficace devrait limiter la recherche linéaire de $\lambda$ à un petit nombre d'essais.

Ainsi, l'algorithme GMRES($k_{\max}$) associé à une technique de "backtracking" est:

**Algorithme 3.6.** Algorithme de GMRES($k_{\max}$) non linéaire

Etape 0: Choisir $\mathbf{u}_0$ et un vecteur initial $\delta_n^0$. Choisir les paramètres $\alpha$ et $\beta$ necessaires pour la recherche linéaire, tels que $\alpha \in (0, 1/2)$ et $\beta \in (1/2, 1)$. Poser $n = 0$.

Etape 1: Choisir une tolérance $\epsilon_n$ et un paramètre $\sigma_n$ à utiliser dans la formule (3.16) au cours de l'algorithme.

Définir $J_n = J(\mathbf{u}_n) = \mathbf{F}'(\mathbf{u}_n)$ le jacobien de la fonction $\mathbf{F}$ et calculer $\mathbf{F}_n = \mathbf{F}(\mathbf{u}_n)$. Calculer $\mathbf{r}_n^0 = -\mathbf{F}_n - J_n \delta_n^0$, $\beta_n = \|\mathbf{r}_n^0\|_2$ et $\mathbf{v}_1 = \mathbf{r}_n^0/\beta_n$. Poser $j = 1$.

Etape 2: Définir:

$$h_{ij} = (J_n \mathbf{v}_j, \mathbf{v}_i) \qquad i = 1, \ldots, j$$

$$\hat{\mathbf{v}}_{j+1} = J_n \mathbf{v}_j - \textstyle\sum_{i=1}^{j} h_{ij} \mathbf{v}_i$$

$$h_{j+1,j} = \|\hat{\mathbf{v}}_{j+1}\|_2$$

$$\mathbf{v}_{j+1} = \hat{\mathbf{v}}_{j+1}/h_{j+1,j}.$$

Construire la $j$-ième colonne de la matrice $\overline{H}_j$ et mettre à jour sa factorisation $Q_j R_j$.

Calculer la norme résiduelle $\rho_j = \|\mathbf{F}_n + J_n \delta_n^j\|_2$ de la solution $\delta_n^j$ qu'on obtiendrait si on s'arrêtait ici.

Si $\rho_j \leq \epsilon_n$ ou si $j = k_{\max}$ poser $k = j$ et aller à l'étape 3. Sinon, poser $j = j + 1$ et recommencer l'étape 2.

Etape 3: Trouver le vecteur $\mathbf{y}_k$ qui minimise $J(\mathbf{y}) = \|\beta_n \mathbf{e}_1 - \overline{H}_k \mathbf{y}\|_2$ sur $\mathbb{R}^k$ (avec $\mathbf{e}_1 = (1, 0, \ldots, 0)^T \in \mathbb{R}^{k+1}$).

Calculer $\delta_n^k = \delta_n^0 + V_k \mathbf{y}_k$.

Choisir $\lambda_n$ par dichotomie en fonction de $\mathbf{u}^n$ et de $\delta_n^k$ de façon à satisfaire les inégalités (3.18) et (3.19) (on limite le nombre d'essais à trois).

Mettre à jour $\mathbf{u}_{n+1} = \mathbf{u}_n + \lambda_n \delta_n^k$.

Etape 4: Si $\mathbf{u}_{n+1}$ est une solution satisfaisante de (3.11), finir; autrement, poser $\mathbf{u}_n = \mathbf{u}_{n+1}$, $n = n + 1$, définir $\delta_n^0$ et aller à l'étape 1. ∎

Remarque 3.4. Dans l'étape 4 on pourrait utiliser plusieurs critères d'arrêt différents; par exemple:

(i) $\|\mathbf{F}(\mathbf{u}_{n+1})\|_2 < \epsilon_n$

(ii) $\frac{\|\mathbf{F}(\mathbf{u}_{n+1})\|_2}{\|\mathbf{F}(\mathbf{u}_0)\|_2} < \epsilon_n$

Dans notre cas particulier on a utilisé le critère (ii). ∎

Remarque 3.5. Comme dans le cas linéaire, on peut montrer que les vecteurs obtenus au cours de l'algorithme sont orthonormaux, i.e.

$$(\mathbf{v}_i, \mathbf{v}_j) = \delta_{ij} \qquad 1 \leq i, j \leq k+1.$$

On peut aussi diminuer le coût du processus d'orthogonalisation en orthogonalisant $\mathbf{v}_{j+1}$ seulement par rapport aux derniers $p$ vecteurs calculés, $\mathbf{v}_{j-p+1}, \ldots, \mathbf{v}_j$. Cet algorithme est appelé IGMRES (Incomplete GMRES) (cf. [B3]). ∎

Remarque 3.6. Pour calculer $J(\mathbf{u})\mathbf{v}$ on peut utiliser, à la place de (3.16), l'approximation (de deuxième ordre) de $\mathbf{F}'(\mathbf{u})\mathbf{v}$ donnée par

$$J(\mathbf{u})\mathbf{v} \approx \frac{\mathbf{F}(\mathbf{u} + \sigma\mathbf{v}) - \mathbf{F}(\mathbf{u} - \sigma\mathbf{v})}{2\sigma} \qquad (3.16)'$$

Le coût d'une évaluation de plus de la fonction pour chaque produit $J(\mathbf{u})\mathbf{v}$ est équilibré par le gain d'un ordre de précision par l'utilisation de la formule (3.16)'.

## 4. Différents préconditionneurs utilisés

Les méthodes de Krylov, et en particulier la méthode de GMRES, font intervenir un processus d'orthogonalisation générateur d'erreurs d'arrondi qui peuvent déstabiliser l'algorithme si la matrice jacobienne est mal conditionnée.

D'autre part, bien que l'algorithme de GMRES soit théoriquement une méthode directe, il nécessite en pratique des réinitialisations et, de ce fait, est une méthode itérative.

Pour ces différentes raisons, la recherche de préconditionneurs est importante à la fois pour la stabilité de la méthode (aux grandes valeurs de $k$, $k$ étant la dimension de l'espace de Krylov utilisé) et pour sa vitesse de convergence (aux faibles valeurs de $k$).

Nous voulons résoudre le problème:

$$\mathbf{F}(\mathbf{u}) = \mathbf{0} \tag{4.1}$$

où $\mathbf{F}$ est une fonction éventuellement non linéaire, de $\mathbb{R}^N$ dans $\mathbb{R}^N$, par la méthode de GMRES préconditionnée.

Dans ce travail nous utilisons un préconditionnement linéaire à gauche. Il s'agit d'une mise à l'échelle des lignes. On se donne une matrice par blocs régulière $S$ d'ordre $N$, et on modiffie le problème (4.1) de la façon suivante:

$$S^{-1}\mathbf{F}(\mathbf{u}) = \mathbf{0}. \tag{4.2}$$

Bien sûr, $\mathbf{u}$ est une solution de (4.2) si et seulement si $\mathbf{u}$ est une solution de (4.1).

Pour appliquer l'algorithme de GMRES à (4.2), il suffit de définir $\mathbf{G} = S^{-1}\mathbf{F}$ et utiliser l'algorithme 3.3 avec la fonction $\mathbf{G}$.

Maintenant il faut choisir l'opérateur $S$, de façon que le système (4.2) soit mieux conditionné que celui de départ (i.e. plus simple à résoudre numériquement) et que le coût de la mise en œuvre du préconditionneur soit raisonnable par rapport au coût de la résolution de (4.1) sans préconditionner.

Dans la description de l'algorithme de GMRES on a vu qu'il résout (d'une façon approchée) la méthode de Newton par une méthode de Krylov. Alors, pour qu'il soit efficace, il faut être capable de résoudre, à un coût raisonnable, le système

$$\partial\mathbf{G}(\mathbf{u}_n)\delta = -\mathbf{G}(\mathbf{u}_n). \tag{4.3}$$

Comme $S$ est linéaire, $\partial\mathbf{G}(\mathbf{u}_n) = S^{-1}\partial\mathbf{F}(\mathbf{u}_n)$. Alors, il est clair que le choix optimal pour $S$ (si on ne regarde que le conditionnement du système) est $\partial\mathbf{F}(\mathbf{u}_n)$ (cf. annexe 1 pour la définition de $\partial\mathbf{F}$). L'inconvenient est qu'il ne faut pas seulement calculer le jacobien du système à résoudre mais il faut aussi pouvoir calculer son inverse!

Comme à chaque pas de temps il faut résoudre plusieurs fois le système linéaire associé à la matrice $S$ (si la dimension de l'espace de Krylov utilisé est $k$ et on fait $m$ pas de l'algorithme de GMRES avant d'avoir une solution satisfaisante, le nombre de systèmes linéaires à résoudre associés à $S$ est proche de $m(k+1)+1$ *), nous avons décidé de faire des décompositions de Gauss plus ou moins complètes de la matrice jacobienne. La description précise des algorithmes définissant ces matrices est donnée dans §5.

Nous avons analysé plusieurs choix pour la matrice $S$:

$$\begin{aligned} S_1 &= I \\ S_2 &= (LDU)_I \\ S_3(\epsilon) &= (LDU)_\epsilon \\ S_4 &= LDU. \end{aligned}$$

$S_1$ est le choix le plus simple: on ne préconditionne pas l'algorithme de GMRES. De l'autre côté, $S_4$ est, par rapport à la convergence, le plus efficace: elle est la factorisation $LDU$ complète de la matrice jacobienne associée á notre système non linéaire. Pour calculer $S_2$ on fait la factorisation $LDU$ incomplète usuelle de la matrice jacobienne associée à la fonction $\mathbf{F}$ (voir §5.2.2), tandis que $S_3$ est une factorisation "dynamique" incomplète de cette matrice (voir §5.2.3, remarque 5.11).

Bien entendu, $S_2$ et $S_3$ ne coïncident pas avec la matrice jacobienne du système non linéaire à résoudre, mais les systèmes linéaires associées à ces matrices sont simples à inverser, et cette façon de décomposer $\partial\mathbf{F}$ en un produit d'une matrice diagonale et de deux matrices triangulaires par blocs est considérablement moins chère que dans le cas de la factorisation complète.

* Il peut diminuer si la convergence à la solution est très bonne ou augmenter dans le cas contraire.

## 5. La méthode d'élimination de Gauss pour une matrice par blocs

Dans cette partie nous nous proposons d'etendre la méthode de décomposition (ou de factorisation) de Gauss à des matrices par blocs. Nous proposons des algorithmes qui donnent la dite décomposition et une autre décomposition équivalente. Nous considérons aussi le cas des factorisations incomplètes par blocs de $A$. Avec cette section nous finisons les définitions faites dans §4 à propos de nos opérateurs de préconditionnement. Pour plus de détails se référer à [D2].

Par la suite $A$ désignera une matrice carrée *par blocs*, avec $N$ blocs par ligne et $N$ blocs par colonne. Nous parlerons de *lignes* et de *colonnes* de la matrice $A$ pour désigner des *lignes* et des *colonnes par blocs*. Par contre pour un bloc donné, les termes lignes et colonnes possèdent le sens usuel. On notera par $A_{ij}$ le $(i,j)$-ième bloc de la matrice. Les blocs diagonaux de $A$ sont carrés et on désignera par $M_i$ l'ordre du bloc $A_{ii}$ $(1 \leq i \leq N)$. On permet $M_i \neq M_j$ pour $i \neq j$. Cela entraîne la possibilité de blocs (extra-diagonaux) *non carrés*, la dimension du bloc $A_{ij}$ étant $M_i \times M_j$. On dira aussi que $N$ est l'"*ordre*" de la matrice "*$A$ par blocs*" (en fait, l'ordre usuel de la matrice $A$ est $\sum_{i=1}^{N} M_i$).

On fera la factorisation de $A$ *sans utiliser de stratégie de pivot*. Il est connu que, bien qu'une matrice soit non singulière, au cours d'une factorisation de Gauss elle peut être "transformée" en une matrice singulière à cause des erreurs d'arrondi qu'occasionne le calcul en arithmétique finie (cf. par exemple [L1], tome 1). Ce risque est sensiblement diminué par l'utilisation des pivots, mais cela ne l'élimine pas nécessairement. Considérant la nécessité de réduire autant que possible le temps CPU, on a préféré ne pas utiliser de stratégie de pivot, sachant bien sûr que ce choix empêche d'appliquer l'algorithme aux matrices mal conditionnées.

Nous supposons aussi que les algorithmes ci-dessous peuvent toujours être appliqués, i.e. que les matrices à considérer sont assez régulières (cf. théorème 5.1) pour permettre l'utilisation de ces algorithmes.

### 5.1. Décomposition de Gauss complète

#### 5.1.1. La décomposition $LU$

On cherche une matrice $L$ triangulaire inférieure ayant pour blocs diagonaux des matrices identité, et une matrice $U$ triangulaire supérieure par blocs, telles que

$$A = LU. \tag{5.1}$$

On initialise les matrices $L$ et $U$ par l'identité et la matrice nulle respectivement, i.e., $L = I$, $U = 0$. Au cours de l'algorithme on écrit les calculs intermédiaires *dans la place*

*mémoire* occupé par la matrice $A$. Donc, si on a besoin de $A$ pour des calculs postérieurs, il faut la sauvegarder *ailleurs*. Comme il est d'usage, le bloc $A_{ij}$ désignera la valeur courante du dit bloc, qui ne coïncide pas nécessairement avec la valeur de départ du $(i,j)$-ième bloc.

La généralisation de la factorisation de Gauss complète à des matrices par blocs que nous proposons est la suivante:

**Algorithme 5.1.** Décomposition $LU$ d'une matrice par blocs

Pour $k = 1, \ldots, N$ faire :
  $U_{kk} = A_{kk}$
  Pour $i = k+1, \ldots, N$ faire :
    $L_{ik} = A_{ik} U_{kk}^{-1}$
    Pour $j = k+1, \ldots, N$ faire :
      $A_{ij} = A_{ij} - L_{ik} A_{kj}$
    Fin boucle $j$.
  Fin boucle $i$.
  Pour $j = k+1, \ldots, N$ faire :
    $U_{kj} = A_{kj}$
  Fin boucle $j$.
Fin boucle $k$. ∎

On stocke les blocs nécessaires pour définir les matrices $L$ et $U$ dans la place mémoire prévue pour la matrice $A$. Plus précisement,

$$\text{le bloc } A_{ij} \text{ est remplacé par } \begin{cases} L_{ij} & \text{si} \quad 1 \le j < i \le N \\ U_{ij} & \text{si} \quad 1 \le i \le j \le N. \end{cases}$$

Les matrices $L$ et $U$ sont facilement reconstruites à partir du stockage effectué, car on connaît la valeur des blocs non stockés, à savoir:

$$\begin{aligned} L_{ij} = U_{ji} = 0 \qquad & \text{si} \qquad 1 \le i < j \le N \\ L_{ii} = I \qquad & \text{si} \qquad 1 \le i \le N. \end{aligned}$$

**Définition 5.1.** *Soit $A$ une matrice par blocs d'ordre $N$: $A = \left(A_{ij}\right)_{1 \le i,j \le N}$. On appelle sous-matrice principale par blocs d'ordre $k$ de $A$, la matrice $A_k$ formée des blocs appartenant à la fois aux $k$ premières colonnes et aux $k$ premières lignes de $A$, c'est-à-dire:*

$$A_k = \begin{pmatrix} A_{11} & \cdots & A_{1k} \\ \vdots & \ddots & \vdots \\ A_{k1} & \cdots & A_{kk} \end{pmatrix}. \quad ∎$$

On peut prouver les résultats suivants:

**Théorème 5.1.** *On peut appliquer l'algorithme de factorisation à une matrice carrée $A$ par blocs d'ordre $N$ si et seulement si toutes les sous-matrices principales par blocs $A_k$ de $A$ (avec $1 \leq k \leq N$) sont régulières.* ∎

**Théorème 5.2.** *Soit $A$ une matrice par blocs, carrée, d'ordre $N$, à laquelle on peut appliquer l'algorithme de factorisation. Alors, il existe deux matrices $L$ et $U$, $L$ étant triangulaire inférieure avec des matrices identité pour blocs diagonaux, et $U$ étant triangulaire supérieure par blocs, telles que:*

$$A = LU.$$

*De plus, les matrices $L$ et $U$ sont uniques.* ∎

Remarque 5.1. On remarque à nouveau que tous les résultats précédents sont valables même si les blocs diagonaux n'ont pas la même dimension.

Du point de vue informatique la factorisation de Gauss dans le cas général d'une matrice qui a des blocs non carrés est beaucoup plus difficile que dans le cas particulier d'une matrice ayant tous les blocs de la même dimension. En effet, dans le premier cas il est convenable de stocker l'adresse du premier élément de chaque bloc, tandis que dans le deuxième on peut la calculer directement à partir de la position du dit bloc (cf. [D2]). Dans le cas général on doit calculer aussi le nombre de lignes et le nombre de colonnes de chaque bloc à utiliser. Cette opération ne demande pas plus de stockage mais il faut connaître la dimension des blocs diagonaux appartenant à la ligne et à la colonne en question, ce qui entraîne une augmentation du temps CPU. ∎

### 5.1.2. Une décomposition *LDU*

Maintenant on veut construire trois matrices par blocs d'ordre $N$, disons $L$, $D$ et $U$, avec $L$ et $U$ des matrices triangulaires inférieure et supérieure par blocs respectivement avec des blocs identité sur la diagonale, et avec $D$ une matrice diagonale par blocs telles que

$$A = LDU. \tag{5.2}$$

Pour factoriser $A$ sous la forme $LDU$ au lieu de la forme $LU$, il suffit de modifier légèrement la dernière boucle de l'algorithme 5.1.

On conserve les notations de §5.1.1, et on initialise $L$, $D$ et $U$ par $L = I$, $U = I$ et $D = 0$. L'algorithme 5.2 peut alors s'écrire:

**Algorithme 5.2.** Décomposition $LDU$ d'une matrice par blocs

Pour $k = 1, \ldots, N$ faire :
  $D_{kk} = A_{kk}$
  Pour $i = k+1, \ldots, N$ faire :
    $L_{ik} = A_{ik} D_{kk}^{-1}$
    Pour $j = k+1, \ldots, N$ faire :
      $A_{ij} = A_{ij} - L_{ik} A_{kj}$
    Fin boucle $j$.
  Fin boucle $i$.
  Pour $j = k+1, \ldots, N$ faire :
    $U_{kj} = D_{kk}^{-1} A_{kj}$
  Fin boucle $j$.
Fin boucle $k$. ∎

On vérifie facilement que $DU = \tilde{U}$ avec $\tilde{U}$ la matrice obtenue par l'algorithme 5.1 $\left(\tilde{U}_{kj} = \sum_i D_{ki} U_{ij} = D_{kk} U_{kj} = D_{kk}(D_{kk}^{-1} \tilde{U}_{kj})\right)$.

Remarque 5.2. Dans les algorithmes précédents on n'a besoin que des inverses des blocs diagonaux, mais non des blocs diagonaux eux-mêmes. Alors, pour chaque $k$ ($1 \leq k \leq N$), le calcul des blocs $L_{ik}$ et $U_{ki}$ (avec $k+1 \leq i \leq N$) est réduit à un simple produit des matrices si on stocke le bloc $D_{kk}^{-1}$ (resp. le bloc $U_{kk}^{-1}$ si on fait la décomposition $LU$) à la place du bloc $D_{kk}$ (resp. du bloc $U_{kk}$).

Remarque 5.3. A propos du stockage des matrices issues de la factorisation, on fait des considérations analogues à celles de §5.1.1. ∎

## 5.2. Matrices creuses: différentes sortes de factorisations incomplètes

### 5.2.1. Définition d'une factorisation incomplète

Dans les applications industrielles l'ordre de la matrice $A$ peut être très grand (de l'ordre de dix mille en 2-D), rendant le stockage de $A$, $L$ et $U$ dans la mémoire centrale de l'ordinateur important voire même impossible. De plus, bien que dans la pratique les éléments non nuls de $A$ soient peu nombreux ($A$, en général, est une matrice *creuse*), les matrices $L$ et $U$, elles, sont malheureusement toujours *pleines*.

En conséquence des unités de mémoire auxiliaire (disques ou bandes) doivent être

utilisées, nécessitant des transferts de données coûteux voire même prohibitifs dans un contexte industriel (problèmes d'entrée-sortie, temps CPU, etc.).

Dans la suite on va considérer seulement le cas de matrices par blocs *creuses*. Donc, le problème posé par le stockage de $A$ peut être résolu en utilisant des techniques de stockage morse (voir [D2], [P1]), mais on ne possède toujours pas de solution pratique au stockage de $L$ et de $U$.

Si on ne peut pas stocker les matrices $L$ et $U$ au complet, on peut penser à stocker un nombre "*raisonnable*" de blocs contenant l'information "*représentative*" de ces matrices. On parle alors d'une *factorisation incomplète* de la matrice $A$. On va appliquer ce principe à la factorisation $LDU$. Le cas de la factorisation $LU$ s'en déduit facilement.

On cherche une matrice triangulaire inférieure $L$ et une matrice triangulaire supérieure $U$, aussi creuses que possible, avec des blocs identité sur la diagonale, et une matrice diagonale $D$ telles que $LDU$ soit "*proche*" de $A$, dans un sens à définir. Par exemple, en posant $E = A - LDU$, on peut demander que $\frac{\|E\|}{\|A\|}$ soit petit pour une norme matricielle $\|\cdot\|$ donnée.

Souvent on impose a priori la structure nulle de $L$, de $D$ et de $U$, c'est-à-dire, on se donne un ensemble d'indices $K \subset \{(i,j) \in \mathbf{N}^2 \mid 1 \leq i,j \leq N\}$ et

$$\text{pour chaque } (i,j) \in K, \text{on demande } \begin{cases} U_{ij} = 0 & \text{si } i < j \\ D_{ij} = 0 & \text{si } i = j \\ L_{ij} = 0 & \text{si } i > j. \end{cases}$$

Bien entendu, par définition des matrices $L$, $D$ et $U$ on a aussi:

$$\begin{aligned} L_{ij} = U_{ji} = 0 \quad & \text{si} \quad 1 \leq i < j \leq N \\ D_{ij} = 0 \quad & \text{si} \quad 1 \leq i \neq j \leq N \\ L_{ii} = U_{ii} = I \quad & \text{si} \quad 1 \leq i \leq N. \end{aligned}$$

Il s'agit évidemment de trouver un compromis. Quand $K$ est petit, on s'attend à avoir $\|E\|$ petit, mais le calcul de $L$, de $D$ et de $U$ devient coûteux. Lorsque $K = \emptyset$, on obtient la décomposition complète de $A$ et on a $E = 0$.

Remarque 5.4. En fait, dans la pratique, on exige que le bloc diagonal $D_{ii}$ soit non singulier pour tout $i$, et alors aucun des indices $(i,i)$ n'appartient à $K$. ∎

### 5.2.2. Définition de la factorisation incomplète usuelle

En particulier, on peut imposer à $L$ et à $U$ la même structure nulle que celle de la matrice $A$, ce qui permet le stockage de $L$, de $D$ et de $U$ dans la place de mémoire réservée

pour $A$, c.à.d.:

$$K = \{(i,j) \in \mathbf{N}^2 \mid 1 \leq i \neq j \leq N \quad \text{et} \quad A_{ij} = 0\}. \tag{5.3}$$

Remarque 5.5. Pour ne pas nuire à la généralité, on n'a pas besoin que les blocs diagonaux de notre matrice de départ soient non nuls (il suffit qu'au moment d'appliquer la $k$-ième étape de notre algorithme on puisse choisir le $k$-ième bloc diagonal comme pivot). Pourtant il faut réserver de la place mémoire pour stocker la matrice diagonale par blocs $D$. Si on envisage stocker les matrices issues de l'algorithme de factorisation incomplète dans la place mémoire réservée pour $A$, il faut donc stocker les blocs diagonaux de $A$, indépendamment de leurs valeurs. ∎

Il reste à expliquer comment trouver les valeurs numériques de $L$, de $D$ et de $U$ qui minimisent $\|A - LDU\|$ pour ce $K$.

Très souvent on détermine les coefficients de $L$, de $D$ et de $U$ en imposant $E_{ij} = 0$ pour $(i,j) \notin K$. Dans ce cas-ci, le calcul des coefficients de $L$, de $D$ et de $U$ se fait, comme dans le cas de la décomposition complète, de proche en proche (par exemple, ligne par ligne ou colonne par colonne), en utilisant le fait que, dans les divers pas de l'algorithme 5.2, les *seuls indices à intervenir* sont ceux qui *n'appartiennent pas à* $K$, c'est-à-dire en pratique très peu.

En ce qui concerne les coefficients de $D$, cette façon de les calculer n'est pas nécessairement la meilleure façon de réduire $\|E\|$. De plus, il faut que tous les blocs $D_{kk}$ soient non singuliers (pour pouvoir continuer le processus), ce qui peut être incompatible avec $E_{kk} = 0$. Dans ce cas, on privilégie la première exigence (non singularité de $D_{kk}$) par rapport à la deuxième ($E_{kk} = 0$), en posant $D_{kk} = I$ si le bloc issu du calcul est singulier.

Dans la suite on appellera simplement *factorisation incomplète* la factorisation qu'on vient de décrire. La définition de la matrice $S_2$ donnée dans §4 est faite selon cette méthode. Comme dans l'algorithme 5.2 on initialise les matrices $L$, $D$ et $U$ par: $L = I$, $U = I$ et $D = 0$. L'ensemble $K$ est défini par (5.3). On fait les mêmes considérations effectuées dans §5.1.1 à propos du stockage des matrices issues de la factorisation, mutatis mutandi (cf. remarques 5.2 et 5.3.). L'algorithme est le suivant:

**Algorithme 5.3.** Factorisation $LDU$ incomplète usuelle

Pour $k = 1, \ldots, N$ faire :

$D_{kk} = A_{kk}$

Si $D_{kk}$ est singulier $\rightarrow$ Poser $D_{kk} = I$.

Pour $i = k+1, \ldots, N$ faire :

Si $(i,k) \in K$, aller fin boucle $i$.

$L_{ik} = A_{ik} D_{kk}^{-1}$

Pour $j = k+1, \ldots, N$ faire :

Si $(i,j) \in K$ ou $(k,j) \in K$, aller fin boucle $j$.

$A_{ij} = A_{ij} - L_{ik} A_{kj}$

Fin boucle $j$.

Fin boucle $i$.

Pour $j = k+1, \ldots, N$ faire :

Si $(k,j) \in K$, aller fin boucle $j$.

$U_{kj} = D_{kk}^{-1} A_{kj}$

Fin boucle $j$.

Fin boucle $k$. ∎

Remarque 5.6. La factorisation $LU$ incomplète usuelle d'une matrice par blocs est analogue. ∎

Remarque 5.7. Après la $k$-ième étape de l'algorithme 5.3, les blocs appartenant à la $k$-ième ligne ne seront plus ni requis ni modifiés par l'algorithme. On constate la même chose pour les blocs de la $k$-ième colonne. En plus, on peut écrire les blocs $L_{ik}$, $D_{kk}$ et $U_{ki}$ dans la place mémoire utilisée respectivement par $A_{ik}$, $A_{kk}$ et $A_{ki}$ (avec $k < i \leq N$). ∎

### 5.2.3. Définition d'une factorisation "dynamique" incomplète

Une sérieuse difficulté à surmonter dans la décomposition $LDU$ (ou $LU$) basée sur la structure nulle de la matrice $A$ est que, à un pas quelconque, on peut trouver non seulement un pivot égal à zéro (ou singulier) mais une ligne complètement nulle. Cette situation se produit dans l'exemple suivant:

$$\begin{pmatrix} x & x & x & 0 \\ x & 0 & 0 & 0 \\ 0 & 0 & x & 0 \\ x & 0 & 0 & x \end{pmatrix}$$

où $x$ désigne un élément (resp. un bloc) non nul. Comme on le vérifie aisément, cette matrice est non singulière si aucun $x$ n'est nul (resp. non singulier). Pourtant, en appliquant l'algorithme 5.3, on obtient après le premier pas une deuxième ligne identiquement nulle. Même une technique de pivot partiel n'empêche pas l'arrêt de l'algorithme, parce qu'au second pas la ligne et la colonne sont nulles.

Une solution à ce problème consiste à éliminer les blocs de la matrice selon leur grandeur plutôt que selon leur position. On doit définir alors des paramètres de tolérance $T_i^{(k)}$ ("drop-

tolerance") qui détermineront quand un bloc est *"grand"* (alors on le conserve) ou *"petit"* (alors on peut le remplacer par zéro). En général $T_i^{(k)}$ va dépendre de la ligne $i$ et de l'indice $k$ de l'étape d'élimination. Malheureusement, même avec cette technique, la possibilité de trouver des lignes nulles n'est pas exclue. Pourtant si on permet un remplissage abondant (i.e. $T_i^{(k)}$ est *"petit"*), les éventuelles lignes nulles apparaîtront à la fin du processus, après que la plus grande partie de la matrice ait été factorisée. Alors, on termine l'algorithme, et on complète les matrices $L$, $D$ et $U$ par des matrices identité.

Le schéma de l'algorithme est le suivant: à la $k$-ième étape on fait l'élimination $LDU$ standard (comme dans l'algorithme 5.2). Ensuite, on choisit les blocs à conserver à l'aide du paramètre de tolérance $T_i^{(k)}$ (voir remarque 5.10) et d'une mesure des blocs notée par $\|\cdot\|$ (voir remarque 5.9), en remplaçant par des blocs nuls les blocs qui ne satisfont pas le critère (cf. [A1]).

Pour diminuer le coût relatif au remplissage de la matrice et au nombre d'opérations à effectuer, on a introduit certaines simplifications à cette procédure. On se place dans la $k$-ième étape de factorisation. Comme on l'a déjà signalé dans la remarque 5.7, au cours de la $k$-ième étape on va définir les blocs $D_{kk}$, $L_{ik}$ et $U_{ki}$ (avec $k < i \leq N$) d'une façon définitive (c.à.d., ils ne seront plus modifiés dans l'execution de l'algorithme). En particulier, le bloc $U_{kj}$ (avec $k < j \leq N$) sera défini comme $D_{kk}^{-1}A_{kj}$ *à la fin* de l'étape et il sera stocké à la place du bloc $A_{kj}$. Par définition du processus de factorisation, le bloc $A_{kj}$ intervient dans la *modification de toutes les lignes* $i$ postérieures à la $k$-ième ayant un multiplicateur $L_{ik}$ non nul. Il est donc important de savoir quelle est l'information "negligeable" existant dans la $k$-ième ligne. Nous avons decidé de placer *au début* de la $k$-ième étape un test (un peu plus faible que d'habitude) sur la grandeur des blocs $U_{kj}$ ($j > k$). Comme à ce moment on n'a pas encore interêt à calculer leurs valeurs, on utilise la proprieté sous-multiplicative des certaines normes matricielles:

$$\|AB\| \leq \|A\|\|B\| \tag{5.4}$$

pour remplacer par zéro les blocs $A_{kj}$ qui satisfont $\|D_{kk}^{-1}\|\|A_{kj}\| < T_k^{(k)}$. Il est clair que pour pouvoir utiliser l'algorithme tel quel, il faut choisir une norme matricielle satisfaisant (5.4) (ce qui n'est pas très restrictif dans la pratique).

On ajoute aussi dans la $k$-ième étape d'autres tests étudiant le remplissage de chaque ligne $i$ postérieure à la $k$-ième, concernant les multiplicateurs $L_{ik}$ et les blocs $A_{ij}$ qu'on vient de (ré-)définir, $j > k$.

L'algorithme est le suivant:

**Algorithme 5.4.** Décomposition $LDU$ dynamique incomplète

Pour $k = 1, \ldots, N$ faire:

$D_{kk} = A_{kk}$

Si $D_{kk}$ est singulier $\rightarrow$ Poser $D_{kk} = I$

Pour $j = k+1, \ldots, N$ faire:

Si $\|D_{kk}^{-1}\| \|A_{kj}\| < T_k^{(k)} \rightarrow$ Poser $A_{kj} = 0$

Fin boucle $j$.

Pour $i = k+1, \ldots, N$ faire:

$L_{ik} = A_{ik} D_{kk}^{-1}$

Si $\|L_{ik}\| < T_i^{(k)} \rightarrow$ Poser $L_{ik} = 0$ et aller à la fin de la boucle $i$.

Pour $j = k+1, \ldots, N$ faire:

$A_{ij} = A_{ij} - L_{ik} A_{kj}$.

Si $\|A_{ij}\| < T_i^{(k)} \rightarrow$ Poser $A_{ij} = 0$

Fin boucle $j$.

Fin boucle $i$.

Pour $j = k+1, \ldots, N$ faire:

$U_{kj} = D_{kk}^{-1} A_{kj}$

Fin boucle $j$.

Fin boucle $k$. ∎

Dans tout le paragraphe §5.2 nous ne considérons que des matrices $A$ par blocs *creuses*, comme nous l'avions signalé dans §5.2.1. Il est clair que pour profiter de cet avantage, il faut stocker la matrice $A$ selon sa structure creuse. L'algorithme 5.4 présente une différence fondamentale par rapport aux autres algorithmes de factorisation présentés au préalable: le stockage des matrices $L$, $D$ et $U$. Ayant toujours présente l'idée de stocker cettes matrices dans la place de mémoire prevue pour la matrice $A$, on voit qu'au cours de l'algorithme 5.4 il est possible définir des blocs appartenant à la structure nulle de $A$, et viceversa, de transformer en blocs nuls des blocs qui étaient non nuls au départ. Donc, sauf le cas trivial du stockage de $A$ comme une matrice *pleine*, il faut absolument avoir une *gestion dynamique de la mémoire* pour suprimer ou insérer des blocs à partir de la structure de départ de la matrice $A$.

Remarque 5.8. En travaillant d'une façon analogue à l'algorithme 5.4, obtient un algorithme de factorisation "dynamique" incomplète $LU$ à partir de l'algorithme 5.1. ∎

Remarque 5.9. Pour mesurer la grandeur des blocs on peut utiliser différentes normes matricielles. Si $A = (a_{ij})$ est une matrice de dimension $M \times N$, on a:

(i) $\|A\|_S = \left(\sum_{i=1}^{M} \sum_{j=1}^{N} |a_{ij}|^2\right)^{1/2}$ Norme de Schur

(ii) $\|A\|_1 = \max_{1 \le j \le N} \left(\sum_{i=1}^{M} |a_{ij}|\right)$ Norme 1

(iii) $\|A\|_\infty = \max_{1 \le i \le M} \left(\sum_{j=1}^{N} |a_{ij}|\right)$ Norme $\infty$

On peut démontrer que les normes 1 et $\infty$ sont des normes matricielles induites, tandis que ce n'est pas le cas pour la norme de Schur (cf. [L1]). La norme de Schur est aussi appelée *norme de Frobenius*; elle est associée au produit scalaire matriciel: $A \cdot B = \mathrm{tr}(A^T B)$.

Dans les trois cas, ces normes satisfont (5.4).

De toute façon l'algorithme ne devrait pas être très dépendant de ce choix puisque toutes les normes matricielles sont équivalentes en dimension finie. ∎

Remarque 5.10. L'élimination des blocs *"petits"* peut se faire selon des critères absolus ou relatifs. On note $A_{ij}^{(k)}$ la valeur du bloc $A_{ij}$ à la $k$-ième étape. On se donne un paramètre réel $T$ et on élimine les blocs $A_{ij}$ tels que:

(i) $\|A_{ij}\| < T\|A_{ii}\|$ $(T_i^{(k)} = T\|A_{ii}^{(k)}\|)$ i.e. $A_{ij}$ est *"petit"* par rapport au bloc diagonal de la même ligne (l'élimination est faite par lignes et non par colonnes).

(ii) $\|A_{ij}\| < Ta_i$ avec $a_i = \max_{1 \le l \le N} \|A_{il}\|$ $(T_i^{(k)} = Ta_i^{(k)})$, i.e. $A_{ij}$ est *"petit"* par rapport au plus grand bloc de la ligne.

(iii) $\|A_{ij}\| < T$ $(T_i^{(k)} = T)$.

(iv) On fixe a priori le nombre $q$ de blocs à conserver dans chaque ligne, et on sauve à chaque étape les $q$ blocs de plus grande norme.
(Pour les critères (ii) et (iii) voir [O1] et [Z1], et pour (iv) voir [S2]).

L'avantage du critère (iv) est qu'il fournit une borne supérieure pour la place de mémoire à utiliser, tandis qu'avec les autres critères on ne peut rien dire. Par contre, pour le critère (iv), on doit choisir un paramètre de plus, à savoir $q$. Dans certains cas, cela peut se faire en fonction des paramètres du problème à résoudre. Par exemple, pour résoudre un problème par la méthode d'éléments finis, on pourrait choisir $q$ proche du nombre maximal des connectivités entre les noeuds appartenant à la discrétisation. ∎

Remarque 5.11. Le préconditionneur $S_3(\epsilon) = (LDU)_\epsilon$ donné dans §4 est obtenu en applicant l'algorithme 5.4 à $\partial \mathbf{F}$ selon deux critères concernant le remplissage de la matrice: un critère absolu (cf. remarque 5.10 (iii)) et un critère rélatif (cf. remarque 5.10 (i)). $\epsilon$ est un paramètre réel positif. ∎

## 6. Résultats numériques

Les résultats numériques visent deux objectifs. D'abord, étudier la performance de nos opérateurs de préconditionnement à un pas de temps donné et, dans une deuxième étape, étudier l'influence de ceux-ci sur la convergence globale vers la solution stationnaire.

Dans la première étape, nous n'avons pas pris en compte *le coût du calcul* de nos préconditionneurs, c'est-à-dire, nous avons regardé la convergence de l'algorithme de GMRES une fois que l'operateur $S_i$ était déjà calculé. Cette étape est particulierement importante, non seulement pour la solution des problèmes stationnaires, mais surtout pour des problèmes instationnaires. En effet, dans ce cas-ci, il faut résoudre le problème non linéaire posé à chaque pas de temps plus précisement que dans le cas stationnaire.

Dans la deuxième étape nous avons pris en compte tous les calculs intermédiaires, notamment le calcul et la factorisation de nos préconditionneurs.

Le critère d'arrêt utilisé dans l'algorithme de GMRES est explicité dans la remarque 3.4. Le temps CPU sur l'axe des abscisses est indiqué en millisecondes.

Tous les exemples ont été testés sur un ordinateur IBM 4381.

Nous avons travaillé sur deux problèmes différents:

### Problème 1:

On considère un écoulement stationnaire autour d'un corps elliptique. La triangulation $\mathcal{T}_h$ comporte 840 sommets et 1600 éléments (figures 6.1 et 6.2), tandis que la triangulation $\mathcal{T}_{h/2}$ a 3280 nœuds et 6400 triangles (figures 6.3 et 6.4).

### Problème 2:

Le cas test considéré est un écoulement stationnaire supersonique autour d'un profil d'aile NACA0012. La triangulation $\mathcal{T}_h$ comporte 2354 nœuds et 4568 triangles (figures 6.5 et 6.6). La triangulation $\mathcal{T}_{h/2}$ a 9276 nœuds et 18272 éléments (figures 6.7 et 6.8).

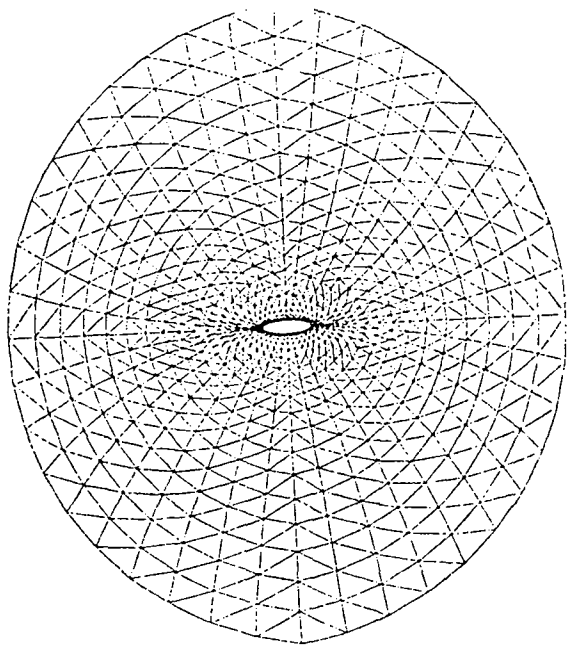
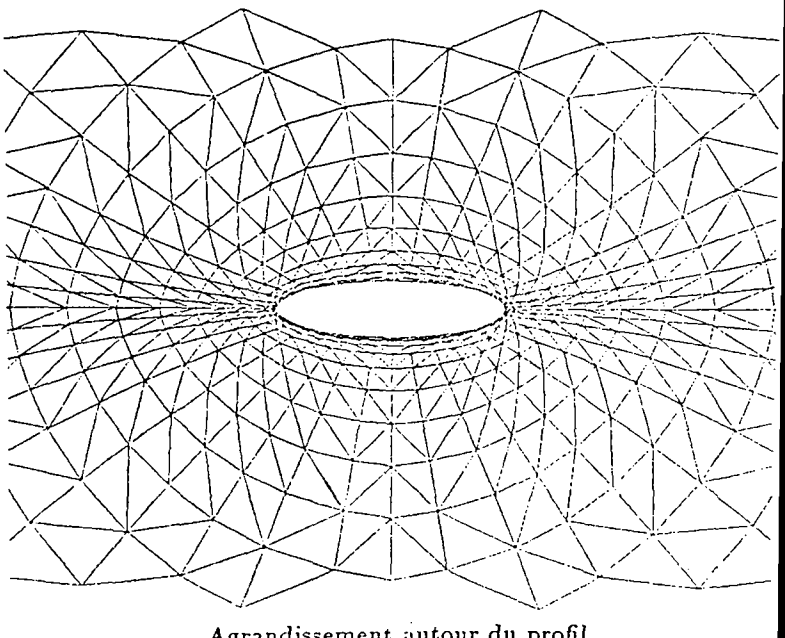


Figure 6.1

Triangulation $\mathcal{T}_h$: 840 nœuds; 1600 triangles.



Agrandissement autour du profil.

Figure 6.2

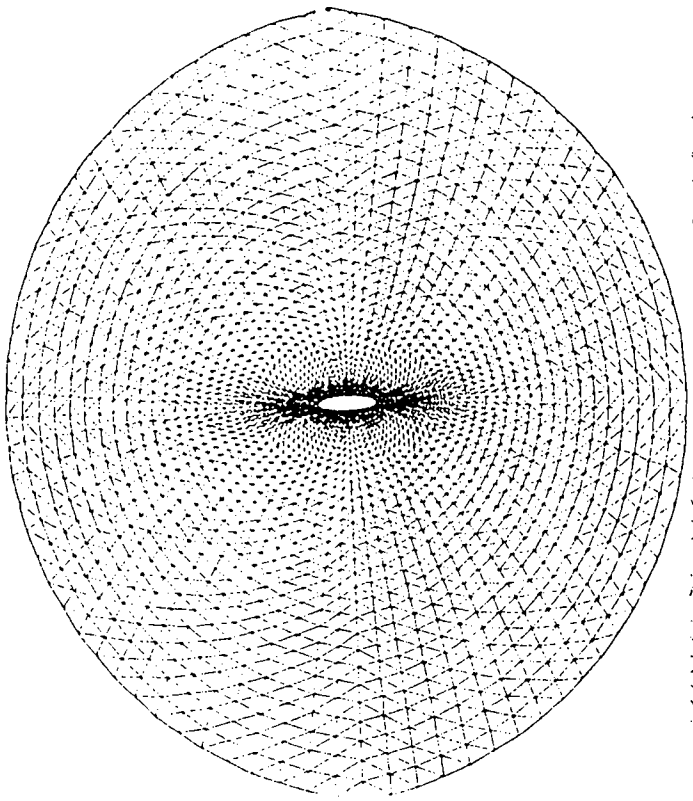
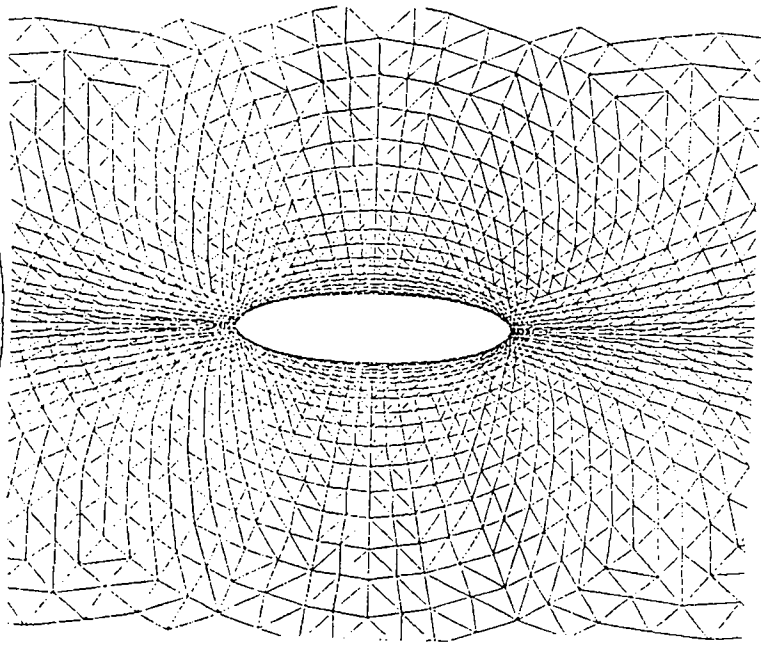


Figure 6.3



Agrandissement autour du profil.

Figure 6.4

Triangulation $\mathcal{T}_{h/2}$: 3280 nœuds; 6400 triangles.

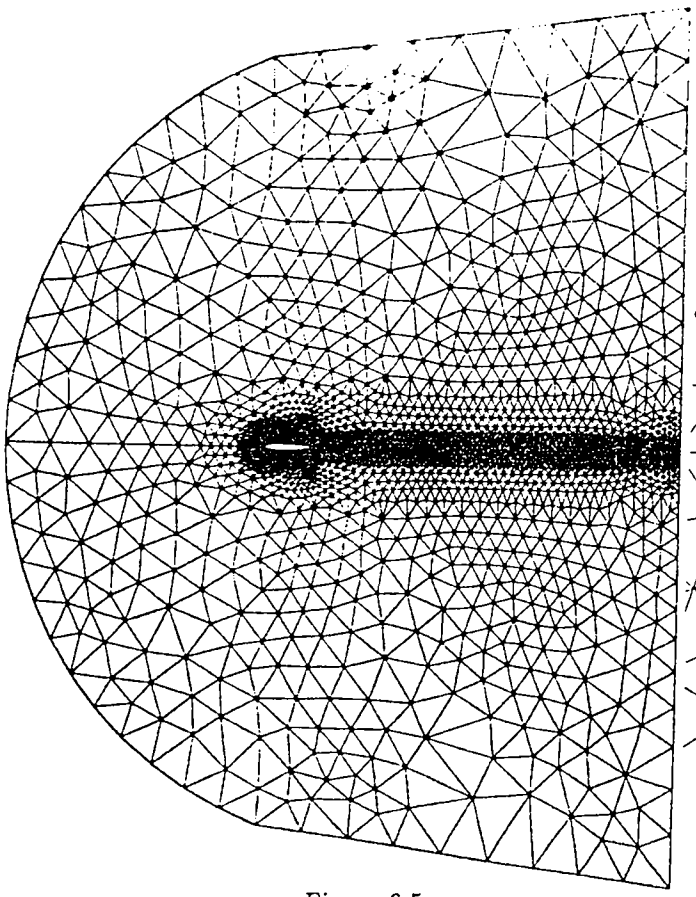
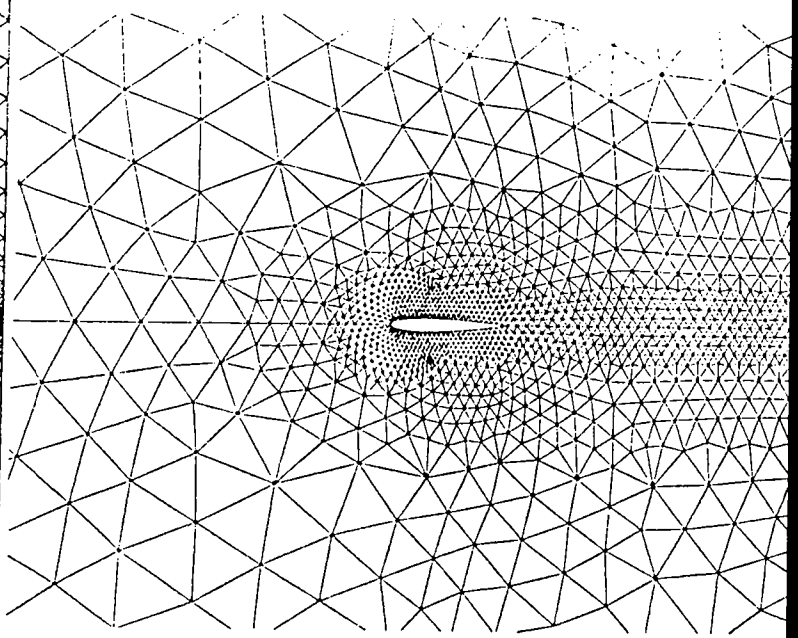


Figure 6.5



Agrandissement autour du profil.

Figure 6.6

Triangulation $\mathcal{T}_h$: 2354 nœuds; 4568 triangles.

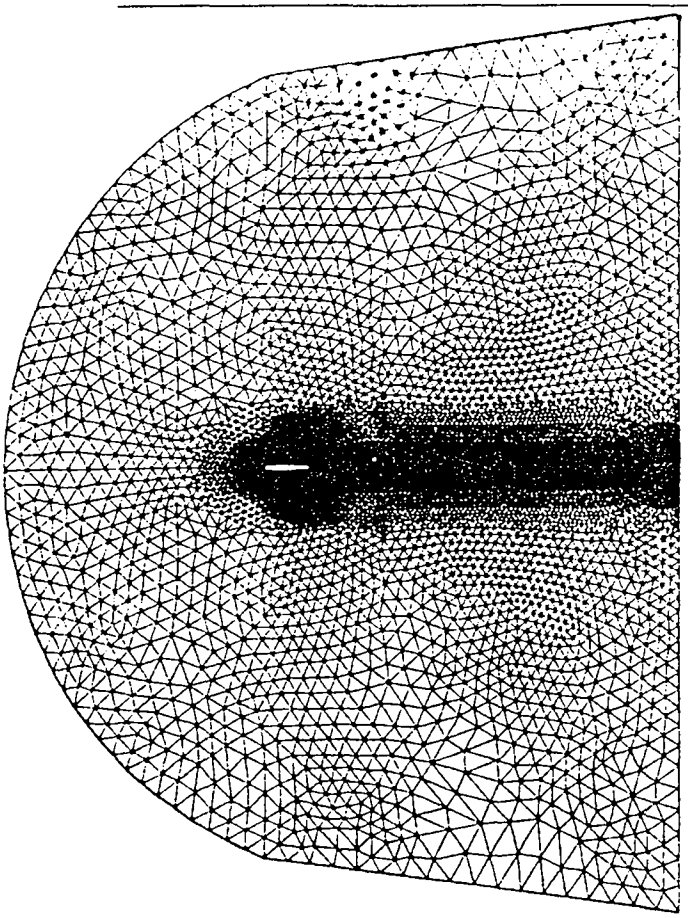
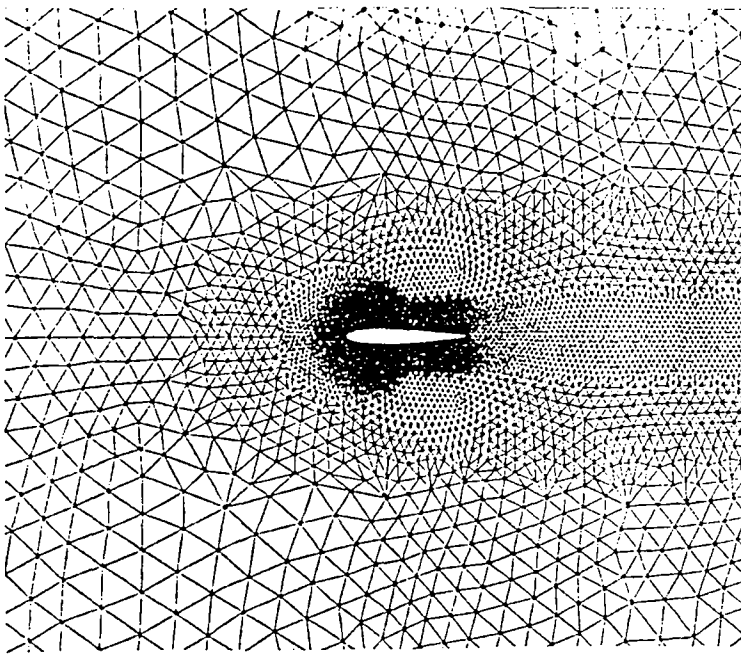


Figure 6.7
Triangulation $\mathcal{T}_{h/2}$: 9276 nœuds; 18272 triangles.



Agrandissement autour du profil.
Figure 6.8

### 6.1. Calculs autour d'une ellipse

On a utilisé un pas de temps constant $\Delta t = .05$. Les données du problème considéré sont $M_\infty = .8$, $\alpha = 0$ et $\mathrm{Re} = 1000$. On montre les résultats obtenus au onzième pas de temps (en prenant la même solution initial pour tous les cas de calcul). On utilise des espaces de Krylov de dimension 4 et on atteint une précision de $10^{-7}$.

Dans les figures 6.9 et 6.10 on compare les résultats obtenus avec certains des préconditionneurs presentés. On peut constater que, en préconditionnant, on divise par deux ou trois le nombre d'itérations nécessaires pour l'algorithme standard. Bien entendu, la relation s'affaiblit un peu quand on considère le temps CPU utilisé. Cependant, le gain reste non negligeable.

Maintenant nous allons étudier un peu plus en détail le préconditionneur $S_3$, qui est le seul à avoir des paramètres à contrôler.

Nous définissons:

$$\lambda(\epsilon) = \frac{\text{nombre total d'éléments de } S_3(\epsilon) \times 100}{\text{nombre total d'éléments de } S_4},$$

qui indique le "pourcentage" associé au remplissage de la matrice $S_3(\epsilon)$ par rapport à la factorisation $LDU$ complète (bien entendu, on considère $S_4$ stockée comme matrice *pleine*). Dans la figure 6.11 nous avons une courbe classique: si $\epsilon$ augmente, le stockage diminue, et viceversa. Le cas limite ($\epsilon = 0$) nous donne la factorisation complète ($S_4$) et alors $\lambda(0) = 100$.

La figure 6.12 nous montre aussi un cas typique, cette fois-ci il s'agit de l'influence de $\epsilon$ par rapport au nombre d'itérations nécessaires pour la résolution du système non linéaire. Quand on augmente la valeur de $\epsilon$ (par exemple $\epsilon = 1$), le stockage diminue, ainsi que l'éfficacité du préconditionneur en question. Inversement, quand $\epsilon$ est plus petit, le stockage augmente et le nombre d'itérations ITER($\epsilon$) diminue. Dans tous les cas de calcul, on a constaté l'existence d'un seuil ITER($\epsilon_0$), limite du nombre d'itérations quand $\epsilon$ tend vers zéro.

Les figures 6.13 à 6.18 comparent le temps CPU utilisé dans la résolution du système non linéaire selon la valeur du paramètre $\epsilon$. Nous indiquons avec une droite parallèle à l'axe des abscisses le niveau où est situé le temps de résolution employé par la matrice $S_2$.

Dans les figures 6.13 à 6.15 on a utilisé un critère absolu pour contrôler le remplissage de la matrice de préconditionnement (cf. remarque 5.10 (iii)). On constate que $S_3$ est avantageux par rapport à $S_2$, au sens que l'on peut obtenir le même temps de calcul en employant beaucoup moins de place mémoire (par exemple, en utilisant la norme de Schur, $S_3(.9)$ emploie un quart du stockage nécessaire pour $S_2$). Si on regarde les préconditionneurs qui utilisent la même place mémoire, on constate que $S_3(.3)$ (avec la norme de Schur) utilise à peu près la moitié des itérations nécessaires pour résoudre le problème à l'aide de $S_2$. Le temps de calcul est, lui aussi, divisé par deux.

Bien qu'en regardant les figures 6.13–6.15 on puisse être optimiste sur le "futur" des préconditionneurs de type $S_3$, l'experience nous montre que, pour les problèmes nous intéressant, *il n'est pas utilisable*, par le moment, *d'une façon pratique.* Le temps de factorisation de la matrice demeure, malheureusement, prohibitif. A titre d'exemple, il suffit de dire que pour ce cas-ci le temps de calcul de la factorisation $LDU$ incomplète de la matrice jacobienne est de 12", mais la factorisation $S_3(.9)$ prends 23' et celle de $S_3(.3)$, 33'!

Dans les figures 6.16 à 6.18 on a utilisé un critère relatif pour contrôler le remplissage de la matrice (cf. remarque 5.10 (i)). Avec les trois normes matricielles utilisées, on constate que l'intervalle "compétitif" pour choisir $\epsilon$ (par rapport à la matrice $S_2$) est très petit, donc, pas intéressant.

On n'a pas utilisé un préconditionneur de type $S_4$ parce que le coût en place mémoire

et en temps CPU nécessaire pour la factorisation est très élévé, étant nul le gain possible en temps CPU pour la résolution du problème (on a déjà remarqué que $S_3(0) = S_4$).

Dans la figure 6.19 on montre, à titre d'exemple, la structure de la matrice jacobienne et aussi de la matrice de préconditionnement $S_2$. Elle comporte 27520 blocs non nuls faisant un total de 217760 coefficients stockés.

o Sans préconditionnement ($S_1$).

$\triangle$ Préconditionnement $LDU$ incomplète ($S_2$)

$+$ Préconditionnement $S_3$ avec $\epsilon = .3$, $\|\cdot\|_\infty$ et critère absolu.

$\times$ Préconditionnement $S_3$ avec $\epsilon = .55$, $\|\cdot\|_\infty$ et critère absolu.

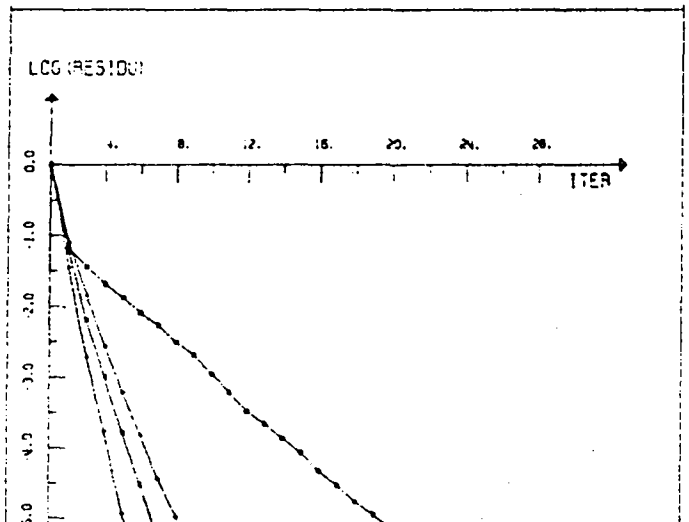



Figure 6.9

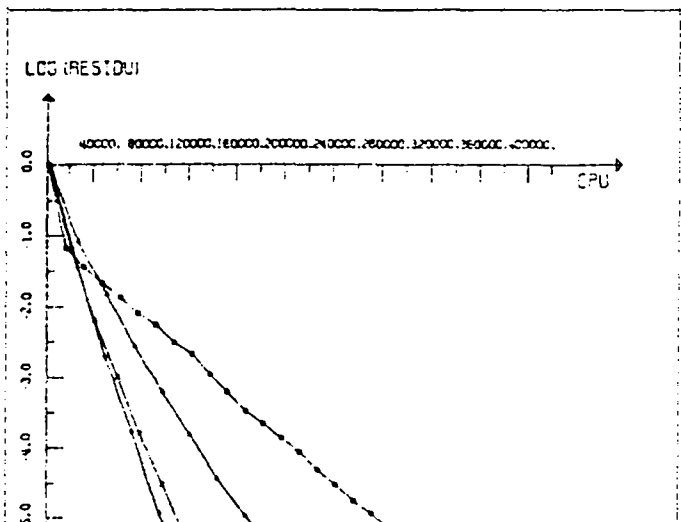



Figure 6.10

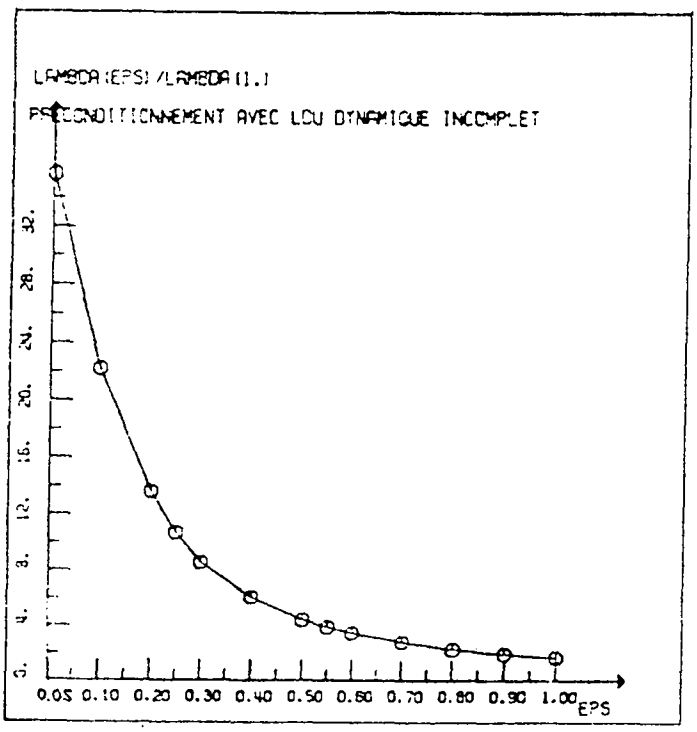



Figure 6.11

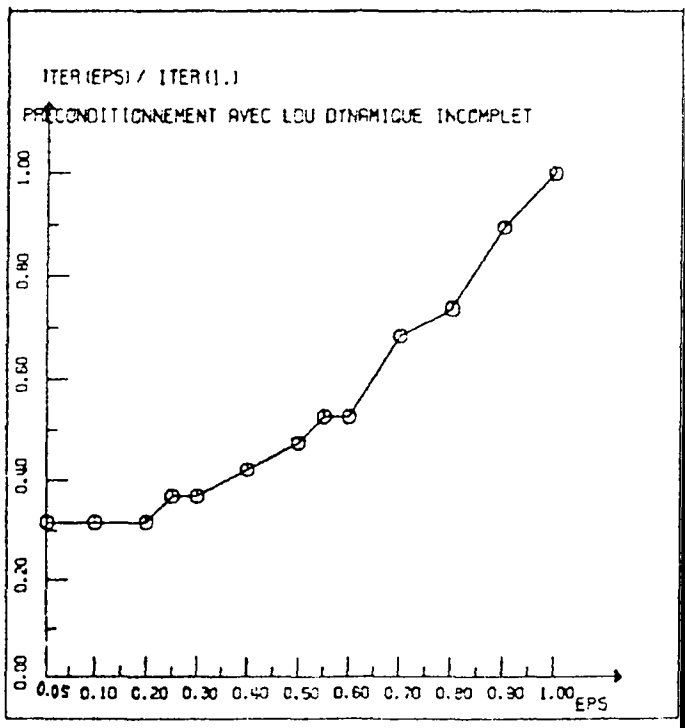
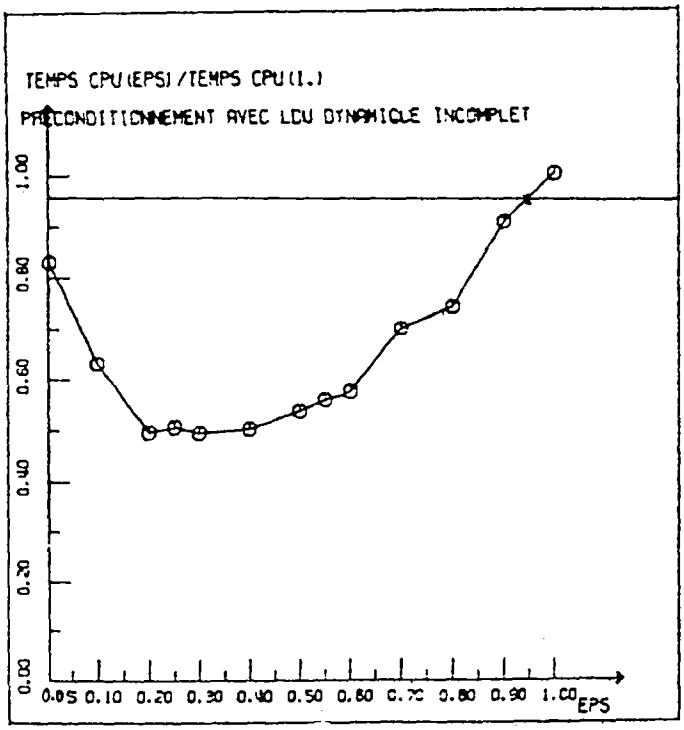


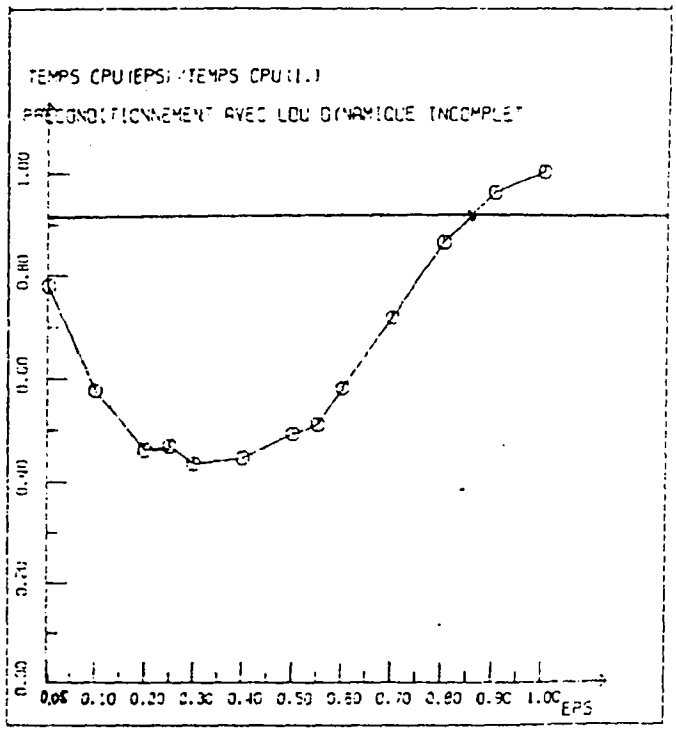

Figure 6.12

$$M_\infty = .8, \mathrm{Re} = 1000., \alpha = 0$$



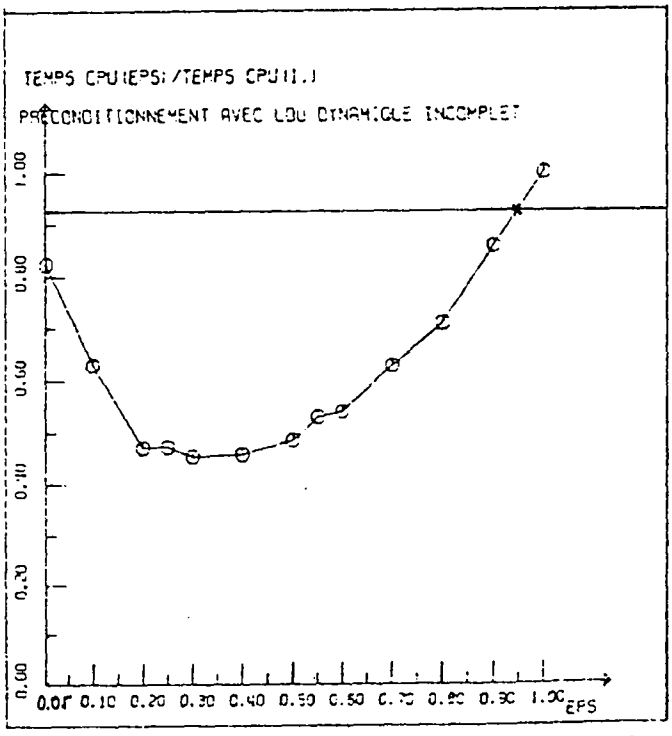

Préconditionnement $S_3$ avec $\|\cdot\|_S$ et critère absolu.

Figure 6.13



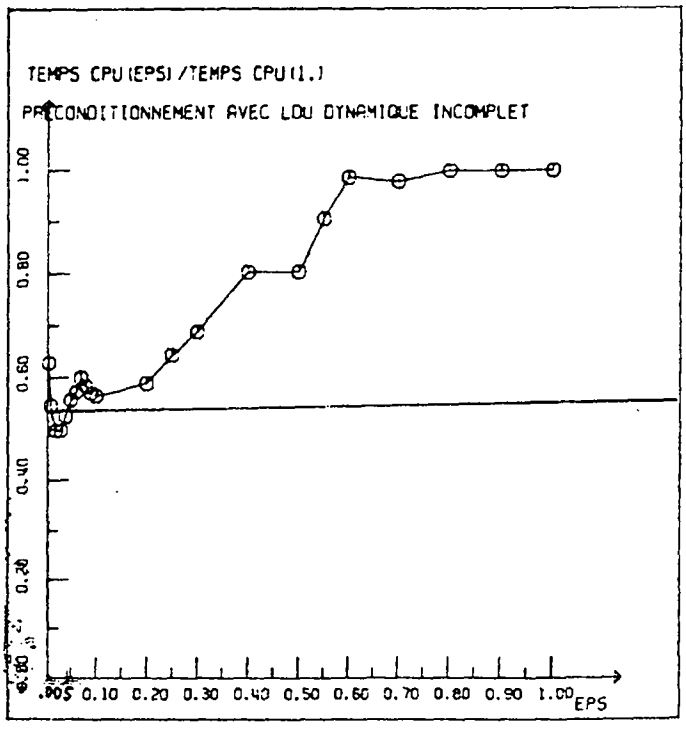

Préconditionnement $S_3$ avec $\|\cdot\|_1$ et critère absolu.

Figure 6.14




Préconditionnement $S_3$ avec $\|\cdot\|_\infty$ et critère absolu.

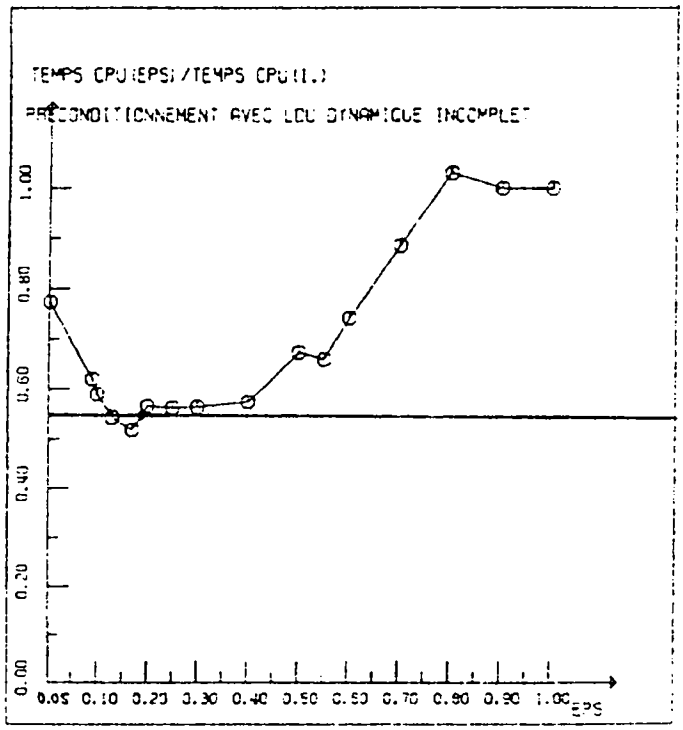
Figure 6.15




Préconditionnement $S_3$ avec $\|\cdot\|_S$ et critère relatif.

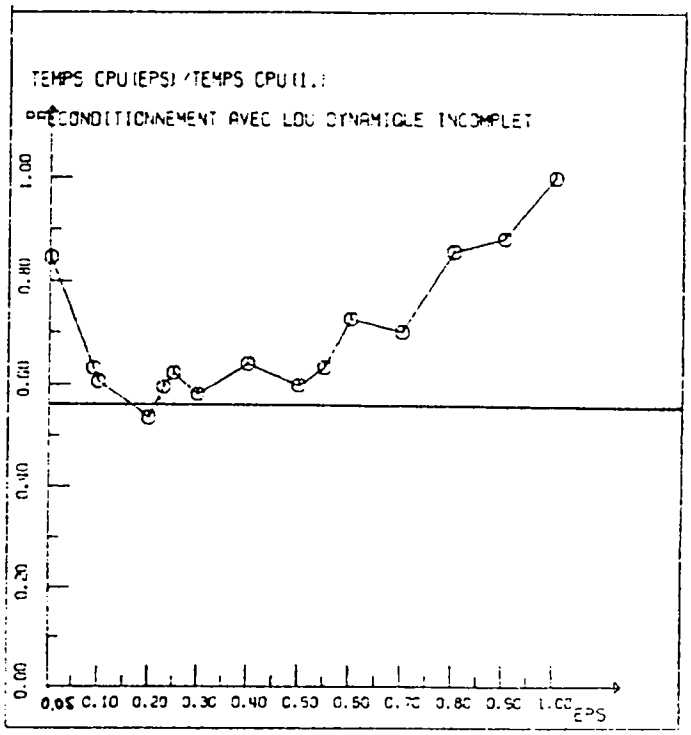
Figure 6.16

$$M_\infty = .8, \mathrm{Re} = 1000., \alpha = 0$$




Préconditionnement $S_3$ avec $\|\cdot\|_1$ et critère relatif.

Figure 6.17




Préconditionnement $S_3$ avec $\|\cdot\|_\infty$ et critère relatif.

Figure 6.18

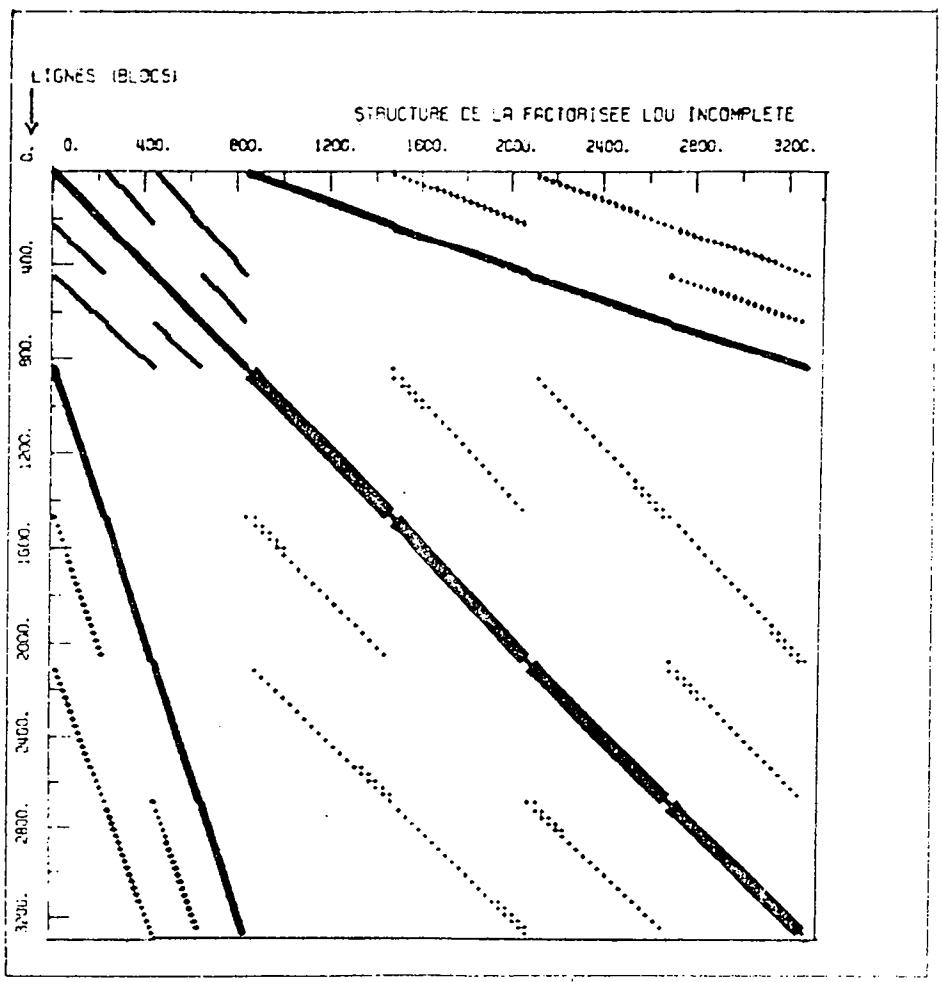



Figure 6.19

$$M_\infty = .8, \mathrm{Re} = 1000., \alpha = 0$$

## 6.2. Calculs autour d'un profil d'aile NACA0012

On considère un écoulement stationnaire supersonique autour d'un profil d'aile NACA0012, avec $M_\infty = 2$, Re = 106 et $\alpha = 10$. Pour résoudre le système non linéaire posé à chaque pas de temps, on utilise des espaces de Krylov de dimension 6 et on demande une précision de $10^{-3}$.

La difficulté du cas de calcul consideré permet que le coût associé à la construction de l'opérateur de préconditionnement soit négligeable face au coût du système non linéaire à résoudre à chaque pas de temps. Les temps de construction de la matrice et de sa factorisation incomplète sont à peu près de 35" chacun.

On étudie le problème selon trois choix différents du pas de temps, à savoir:

(i) $\Delta t = .25$ (constant);

(ii) $\Delta t = .50$ (constant);

(iii) pas de temps local (pour accélerer la méthode on multiplie la valeur obtenue sur chaque nœud par 10. On note: CFL=10).

Dans les tableaux 6.1–6.3 on donne une idée de la difficulté rencontrée au cours de la résolution du problème stationnaire. On considère le problème sans préconditionner ("préconditionneur" $S_1$) et on le compare avec le système préconditionné par la factorisation $LDU$ incomplète de la matrice jacobienne (préconditionneur $S_2$). On donne le nombre d'itérations demandées pour résoudre le système non linéaire par GMRES, *à un pas de temps donné*. On donne aussi le temps CPU que cela demande. On n'est pas interessé ici à *une* itération de GMRES en particulier, mais *au temps total* de résolution du système non linéaire.

| | 10ième. pas | 20ième. pas | 30ième. pas | ··· | 60ième. pas |
|---|---|---|---|---|---|
| **Préc.** $S_1$ | 17 iter - 16'30" | 14 iter - 15' | 13 iter - 13'30" | ··· | 10 iter - 10'40" |
| **Préc.** $S_2$ | 5 iter - 6'40" | 5 iter - 6'40" | 5 iter - 6'40" | ··· | 4 iter - 5'20" |

Tableau 6.1. Cas (i) $\Delta t = .25$

| | 10ième. pas | 20ième. pas |
|---|---|---|
| **Préc.** $S_1$ | 33 iter - 34'30" | 28 iter - 29'30" |
| **Préc.** $S_2$ | 14 iter - 21'30" | 12 iter - 20' |

Tableau 6.2. Cas (ii) $\Delta t = .50$

| | 10ième. pas | 20ième. pas | 30ième. pas | ··· | 80ième. pas |
|---|---|---|---|---|---|
| **Préc.** $S_1$ | 9 iter - 9'30" | 8 iter - 8'30" | 7 iter - 7'30" | ··· | 5 iter - 5'25" |
| **Préc.** $S_2$ | 3 iter - 5' | 3 iter - 5' | 3 iter - 5' | ··· | 3 iter - 5' |

Tableau 6.3. Cas (iii) CFL= 10

Les cas (i) et (iii) ont été initialisés par la résolution d'un problème de Poisson, tandis que dans le cas (ii) on est parti d'une solution initiale moins arbitraire (on a utilisé la solution du premier pas de temps de l'algorithme sans préconditionner). Cela a été nécessaire à cause des instabilités présentés dans le démarrage du problème préconditionné.

Les cinq premiers pas de temps ont eu un traitement spécial, à savoir, la mise à jour de la matrice de préconditionnement à chacun de ces pas de temps. Après le cinquième pas de temps, dans les cas (i) et (ii) on a mis à jour la matrice de préconditionnement chaque cinq pas de temps, tandis que dans le cas (iii) on ne l'a fait que chaque dix pas de temps.

Dans le tableau 6.3 on constate que, sauf dans les cas particulierement durs, il est convenable de diminuer la fréquence de mises à jour de la matrice de préconditionnement à mesure que la méthode instationnaire converge vers la solution stationnaire, voire même ne plus préconditionner du tout quand le système non linéaire à résoudre est suffissament simple. Cela s'explique facilement. Quand on est assez proche de la solution stationnaire, l'algorithme de GMRES converge très rapidement même si on ne le préconditionne pas. Alors, le coût des produits matrice–vecteur dans le cas préconditionné devient non négligeable par rapport à son efficacité et il a un effet négatif sur l'amélioration du coût total de la résolution du système non linéaire.

Dans la figure 6.20 on affiche la convergence de la solution obtenue au cours du temps vers la solution stationnaire (cas (i)). Sur l'axe des ordonnés on a $\log(\frac{\partial \sigma}{\partial t})$ où $\frac{\partial \sigma}{\partial t}$ est approché par $\frac{\sigma^{n+1}-\sigma^n}{\Delta t}$, avec $\sigma$ le logarithme de la densité. Sur l'axe des abscisses on a le pas de temps correspondant. On constate que les deux courbes (avec et sans préconditionnement) coïncident. Cela veut dire que, à chaque pas de temps, on a obtenu la convergence de l'algorithme de GMRES. Le même fait est observé dans les deux autres cas (on omet les courbes associées).

Les figures 6.21–6.23 affichent les courbes de convergence globale par rapport au temps CPU, dans les cas (i)–(iii) respectivement. Ces courbes font bien voir le gain en temps CPU qu'on obtient en employant la méthode préconditionnée.

Sans préconditionnement $(S_1)$.
Préconditionnement $LDU$ incomplète $(S_2)$

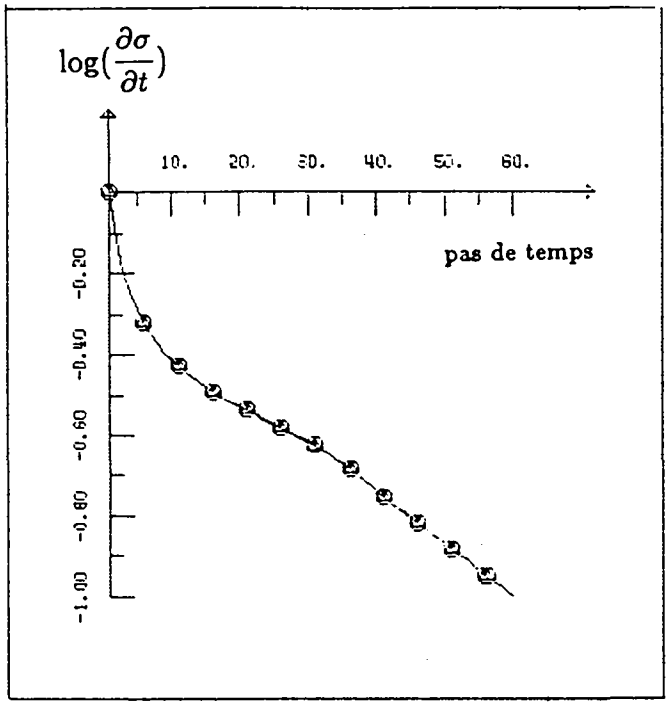



Figure 6.20

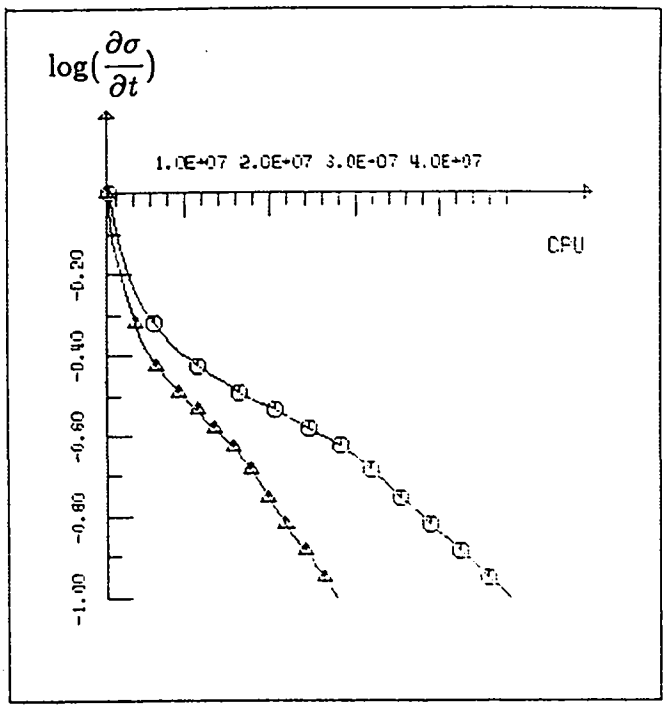



Figure 6.21

60 pas de temps
Cas (i)

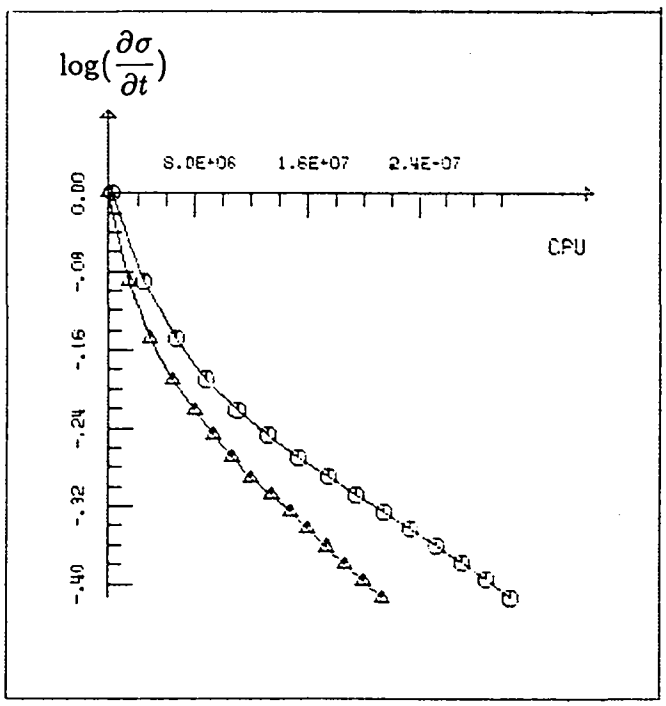



Figure 6.22
Cas (ii)
15 pas de temps.

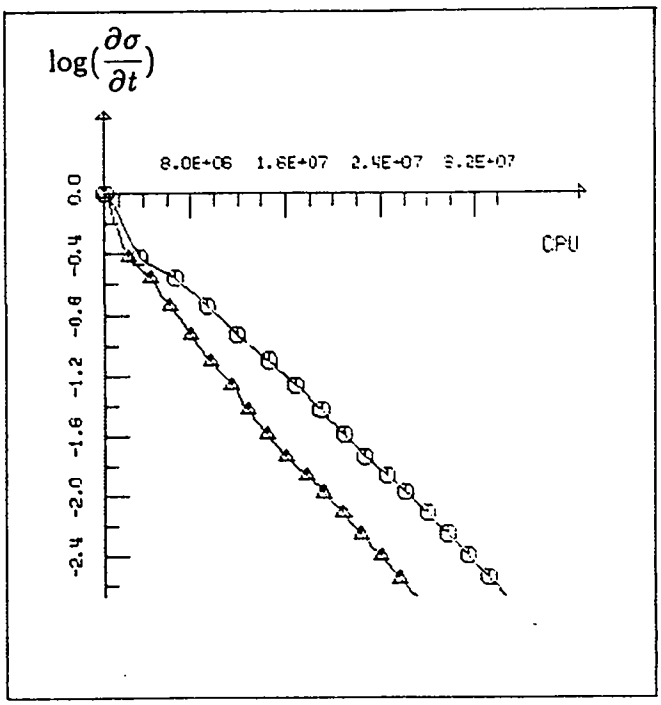
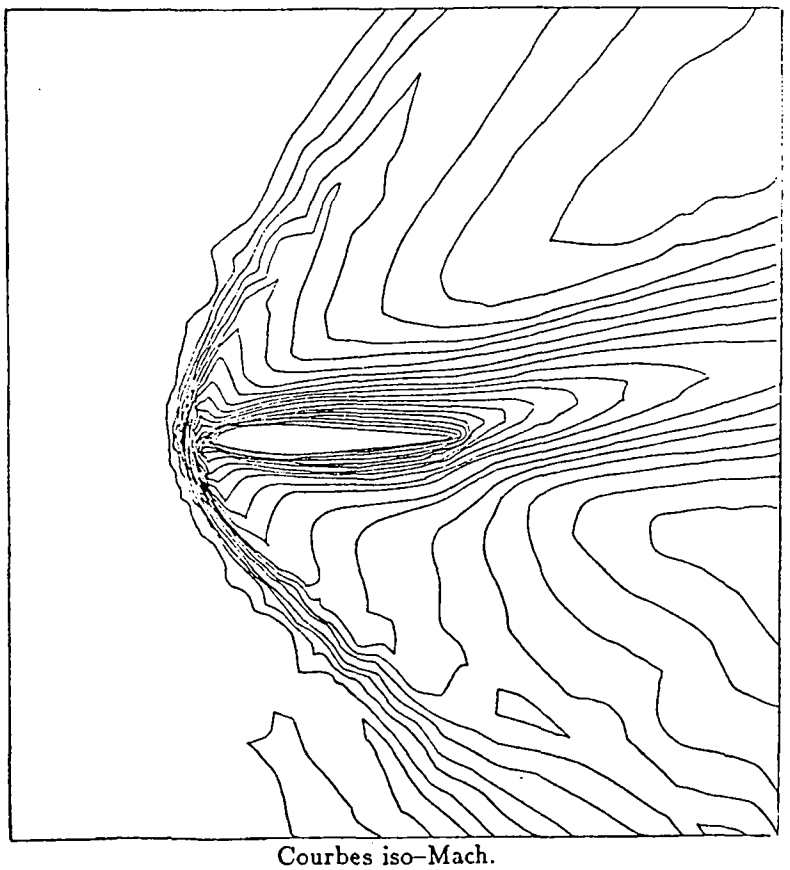


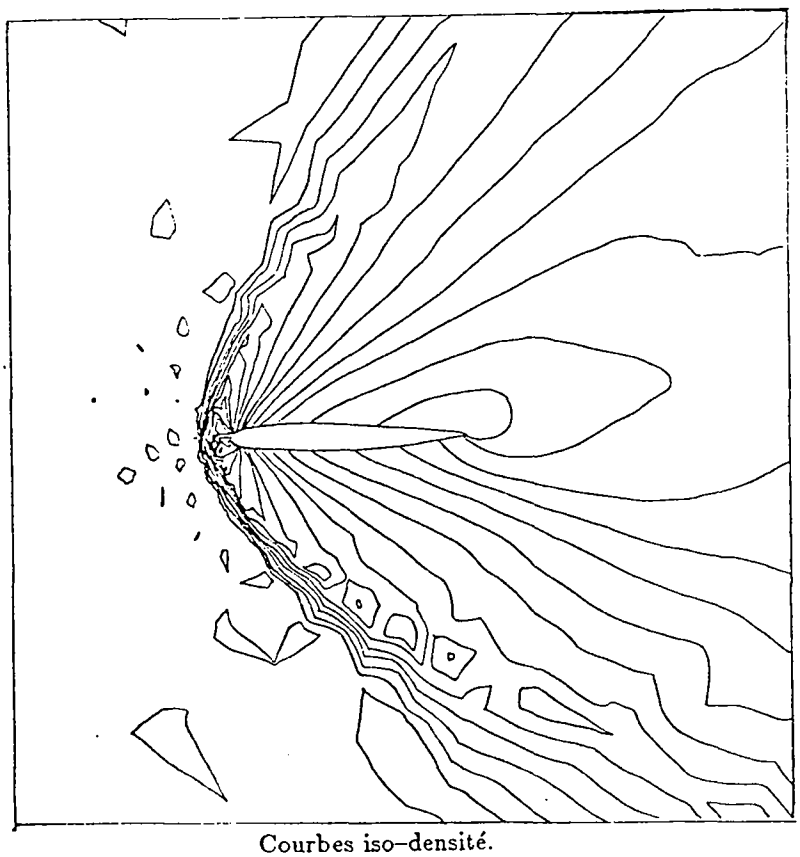

Figure 6.23
Cas (iii).
80 pas de temps.

$M_\infty = 2., \mathrm{Re} = 106., \alpha = 10$
Convergence vers la solution stationnaire.

Finalement les figures 6.27 et 6.28 montrent les contours de la densité et du nombre de Mach pour le cas de calcul (i), à un temps $t = 15$. L'erreur en $\sigma$ dans la solution obtenue est de .65E-2. Malheureusement le maillage n'est pas adapté pour capturer le choc, et cela explique la lenteur de la convergence vers la solution stationnaire.




Courbes iso–Mach.
Figure 6.27




Courbes iso–densité.
Figure 6.28
$M_\infty = 2., \mathrm{Re} = 106., \alpha = 10$

## 7. Conclusion

Nous avons présenté une façon de discrétiser les équations de Navier–Stokes pour un fluide compressible, écrites sous une forme non conservative.

Nous avons détaillé aussi la méthode de résolution utilisée, à savoir, la méthode de GMRES. Celle-ci est une méthode générale, très simple à mettre en œuvre, qui peut être utilisée dans le cas linéaire (en particulier, pour les systèmes non symétriques) et, moyennant quelques modifications, dans le cas non linéaire.

Nous avons présenté plusieurs façons de préconditionner cette méthode appliquée aux équations de Navier–Stokes. Néanmoins, les diverses méthodes proposées pour obtenir des factorisations incomplètes de la matrice jacobienne sont générales et peuvent être utilisées pour la résolution numérique d'autres systèmes d'équations différentielles.

Les préconditionneurs $S_3(\epsilon)$ sont prometteurs vis-à-vis de leur efficacité, mais pour l'instant ils ne sont pas utilisables dans la pratique à cause du coût exagéré de la factorisation qu'ils demandent.

Le préconditionneur $S_2$ est, sans doute, le plus intéressant. On constate que, à un pas de temps donné, il accélère notablement la résolution du problème non linéaire.

Quand on est intéressé par une solution stationnaire des équations de Navier–Stokes, il n'est pas nécessaire d'actualiser la matrice jacobienne (et sa factorisation $LDU$ associée) à chaque pas de temps, surtout si on a déjà effectué quelques itérations de la méthode pseudo–instationnaire. Pour utiliser nos préconditionneurs d'une façon optimale il faut régler la fréquence de modification de la matrice au cours du temps.

Des préconditionneurs autres que ceux qui ont été présentés ici ont été testés, mais les résultats ne sont pas particulièrement intéressants. Pour plus de détails, voir [D2].

Des études envisageant la résolution des équations de Navier–Stokes par des méthodes de type Quasi–Newton sont en cours.

**Remerciements:**

Je voudrais remercier très chaleureusement Mme. Marie–Odile Bristeau et M. Gilbert Rogé pour l'aide amicale qu'ils m'ont prodiguée continuellement au cours de mes travaux.

Je tiens aussi à remercier M. Jacques Périaux et M. Edmundo Rofman pour m'avoir conduit, par leur soutien et leurs conseils, à mener à bon terme le présent travail.

## ANNEXE 1. Calcul du jacobien associé aux équations de Navier–Stokes

**Définition A1.1.** *Une transformation* $\mathbf{F} : D \subset \mathbb{R}^n \to \mathbb{R}^m$ *est dérivable au sens de Gateaux dans un point* $\mathbf{x} \in D^o$ *s'il existe un transformation linéaire* $A \in L(\mathbb{R}^n, \mathbb{R}^m)$ *telle que pour tout* $\mathbf{h}$ *dans* $\mathbb{R}^n$

$$\lim_{t\to 0} \frac{1}{t}\|\mathbf{F}(\mathbf{x}+t\mathbf{h}) - \mathbf{F}(\mathbf{x}) - tA\mathbf{h}\| = 0. \qquad \blacksquare \qquad (A1.1)$$

Si la dérivée de $\mathbf{F}$ existe, alors elle est unique et on écrit $A = \mathbf{F}'(\mathbf{x}) = (a_{ij})$. En plus, si on pose $F^T = (f_1, \ldots, f_m)$, et on choisit les vecteurs de la base canonique dans (A1.1), on peut voir qu'il existe les dérivées partielles de chaque $f_i$ et que les composantes de la transformation linéaire $A$ coïncident avec celles-ci, i.e.

$$a_{ij} = \partial_j f_i.$$

La transformation linéaire $A$ est appelée le *jacobien* de la transformation $\mathbf{F}$.

On peut voir que l'existence de la dérivée de Gateaux de $\mathbf{F}$ entraîne l'existence du jacobien mais l'affirmation réciproque n'est pas toujours vraie (cf. [O2]).

A partir de notre classement des inconnues par blocs, la matrice jacobienne aura des blocs rectangulaires, selon que le nœud appartient à la triangulation $\mathcal{T}_h$ ou $\mathcal{T}_{h/2}$.

Schématiquement, si $A_{ij}$ est le $(i,j)$-ième bloc du jacobien associé à la fonction $\mathbf{F}$(définie dans la section §2 par (2.25) et (2.26)), $A_{ij} = \partial_j \mathbf{F}_i$ est composé par les dérivées suivantes:

$$\text{si} \quad \begin{cases} i \in \mathcal{T}_h, \\ j \in \mathcal{T}_h \end{cases} \quad \Rightarrow \quad A_{ij} = \begin{pmatrix} \partial_{\sigma_j} f_1^{(i)} & \partial_{u_j} f_1^{(i)} & \partial_{v_j} f_1^{(i)} & \partial_{T_j} f_1^{(i)} \\ \partial_{\sigma_j} f_2^{(i)} & \partial_{u_j} f_2^{(i)} & \partial_{v_j} f_2^{(i)} & \partial_{T_j} f_2^{(i)} \\ \partial_{\sigma_j} f_3^{(i)} & \partial_{u_j} f_3^{(i)} & \partial_{v_j} f_3^{(i)} & \partial_{T_j} f_3^{(i)} \\ \partial_{\sigma_j} f_4^{(i)} & \partial_{u_j} f_4^{(i)} & \partial_{v_j} f_4^{(i)} & \partial_{T_j} f_4^{(i)} \end{pmatrix};$$

$$\text{si} \quad \begin{cases} i \in \mathcal{T}_h, \\ j \in \mathcal{T}_{h/2}, \\ j \notin \mathcal{T}_h \end{cases} \quad \Rightarrow \quad A_{ij} = \begin{pmatrix} \partial_{u_j} f_1^{(i)} & \partial_{v_j} f_1^{(i)} \\ \partial_{u_j} f_2^{(i)} & \partial_{v_j} f_2^{(i)} \\ \partial_{u_j} f_3^{(i)} & \partial_{v_j} f_3^{(i)} \\ \partial_{u_j} f_4^{(i)} & \partial_{v_j} f_4^{(i)} \end{pmatrix};$$

$$\text{si} \quad \begin{cases} i \in \mathcal{T}_{h/2}, \\ i \notin \mathcal{T}_h, \\ j \in \mathcal{T}_h \end{cases} \quad \Rightarrow \quad A_{ij} = \begin{pmatrix} \partial_{\sigma_j} f_1^{(i)} & \partial_{u_j} f_1^{(i)} & \partial_{v_j} f_1^{(i)} & \partial_{T_j} f_1^{(i)} \\ \partial_{\sigma_j} f_2^{(i)} & \partial_{u_j} f_2^{(i)} & \partial_{v_j} f_2^{(i)} & \partial_{T_j} f_2^{(i)} \end{pmatrix};$$

$$\text{si} \quad \begin{cases} i \in \mathcal{T}_{h/2}, \\ i \notin \mathcal{T}_h, \\ j \in \mathcal{T}_{h/2} \\ j \notin \mathcal{T}_h \end{cases} \quad \Rightarrow \quad A_{ij} = \begin{pmatrix} \partial_{u_j} f_1^{(i)} & \partial_{v_j} f_1^{(i)} \\ \partial_{u_j} f_2^{(i)} & \partial_{v_j} f_2^{(i)} \end{pmatrix}.$$

Pour ne pas alourdir le notation, on calculera les dérivées *par rapport au problème continu*, mais dans la pratique, il faut le faire *par rapport au problème discret*, i.e. à partir de la définition de $\mathbf{F}$ donnée dans la section §2 par (2.25) et (2.26).

$$
\begin{aligned}
\frac{\partial H_1}{\partial \sigma}(\mathbf{U},\delta\sigma) =& \alpha\delta\sigma + u\frac{\partial\delta\sigma}{\partial x} + v\frac{\partial\delta\sigma}{\partial y}\\
\frac{\partial H_2}{\partial \sigma}(\mathbf{U},\delta\sigma) =& (\gamma-1)T\frac{\partial\delta\sigma}{\partial x} - \frac{e^{-\sigma}}{\mathrm{Re}}\delta\sigma\Big(\Delta u + \frac{1}{3}\frac{\partial}{\partial x}\Big(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\Big)\Big)\\
\frac{\partial H_3}{\partial \sigma}(\mathbf{U},\delta\sigma) =& (\gamma-1)T\frac{\partial\delta\sigma}{\partial y} - \frac{e^{-\sigma}}{\mathrm{Re}}\delta\sigma\Big(\Delta v + \frac{1}{3}\frac{\partial}{\partial y}\Big(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\Big)\Big)\\
\frac{\partial H_4}{\partial \sigma}(\mathbf{U},\delta\sigma) =& \frac{e^{-\sigma}}{\mathrm{Re}}\delta\sigma\Big(\frac{\gamma}{\mathrm{Pr}}\Delta T + F(\nabla\mathbf{u})\Big)
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial H_1}{\partial u}(\mathbf{U},\delta u) =& \delta u\frac{\partial\sigma}{\partial x} + \frac{\partial\delta u}{\partial x}\\
\frac{\partial H_2}{\partial u}(\mathbf{U},\delta u) =& \alpha\delta u + \Big(u\frac{\partial\delta u}{\partial x} + v\frac{\partial\delta u}{\partial y}\Big) + \delta u\frac{\partial u}{\partial x} - \frac{e^{-\sigma}}{\mathrm{Re}}\Big(\Delta\delta u + \frac{1}{3}\frac{\partial^2\delta u}{\partial x^2}\Big)\\
\frac{\partial H_3}{\partial u}(\mathbf{U},\delta u) =& \delta u\frac{\partial v}{\partial x} - \frac{1}{3}\frac{e^{-\sigma}}{\mathrm{Re}}\frac{\partial^2\delta u}{\partial y\partial x}\\
\frac{\partial H_4}{\partial u}(\mathbf{U},\delta u) =& \delta u\frac{\partial T}{\partial x} + (\gamma-1)T\frac{\partial\delta u}{\partial x}\\
&- \frac{e^{-\sigma}}{\mathrm{Re}}\Big(\frac{4}{3}\Big(2\frac{\partial u}{\partial x} - \frac{\partial v}{\partial y}\Big)\frac{\partial\delta u}{\partial x} + 2\Big(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\Big)\frac{\partial\delta u}{\partial y}\Big)
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial H_1}{\partial v}(\mathbf{U},\delta v) =& \delta v\frac{\partial\sigma}{\partial y} + \frac{\partial\delta v}{\partial y}\\
\frac{\partial H_2}{\partial v}(\mathbf{U},\delta v) =& \delta v\frac{\partial u}{\partial y} - \frac{1}{3}\frac{e^{-\sigma}}{\mathrm{Re}}\frac{\partial^2\delta v}{\partial x\partial y}\\
\frac{\partial H_3}{\partial v}(\mathbf{U},\delta v) =& \alpha\delta v + \Big(u\frac{\partial\delta v}{\partial x} + v\frac{\partial\delta v}{\partial y}\Big) + \delta v\frac{\partial v}{\partial y} - \frac{e^{-\sigma}}{\mathrm{Re}}\Big(\Delta\delta v + \frac{1}{3}\frac{\partial^2\delta v}{\partial y^2}\Big)\\
\frac{\partial H_4}{\partial v}(\mathbf{U},\delta v) =& \delta v\frac{\partial T}{\partial y} + (\gamma-1)T\frac{\partial\delta v}{\partial y}\\
&- \frac{e^{-\sigma}}{\mathrm{Re}}\Big(\frac{4}{3}\Big(2\frac{\partial v}{\partial y} - \frac{\partial u}{\partial x}\Big)\frac{\partial\delta v}{\partial y} + 2\Big(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\Big)\frac{\partial\delta v}{\partial x}\Big)
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial H_1}{\partial T}(\mathbf{U},\delta T) =& 0\\
\frac{\partial H_2}{\partial T}(\mathbf{U},\delta T) =& (\gamma-1)\delta T\frac{\partial\sigma}{\partial x} + \frac{\partial\delta T}{\partial x}\\
\frac{\partial H_3}{\partial T}(\mathbf{U},\delta T) =& (\gamma-1)\delta T\frac{\partial\sigma}{\partial y} + \frac{\partial\delta T}{\partial y}
\end{aligned}
$$

$$\frac{\partial H_4}{\partial T}(\mathbf{U}, \delta T) = \alpha \delta T + \left(u\frac{\partial \delta T}{\partial x} + v\frac{\partial \delta T}{\partial y}\right) + (\gamma - 1)\delta T\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right) - \frac{\gamma}{\mathrm{Re\ Pr}} e^{-\sigma} \Delta \delta T.$$

## Bibliographie

[A1] C. Atamian: Rapport de DEA de l'Université P. et M. Curie (Paris VI), juin 1988.

[B1] M. O. Bristeau, R. Glowinski, J. Périaux: *Numerical methods for the Navier-Stokes equations. Applications in the simulation of compressible and incompressible viscous flows*; Finite Element Methods in Physics, R. Gruber ed., Computer Physics Report **6** (1987), North Holland, Amsterdam.

[B2] C. Bègue, M. O. Bristeau, R. Glowinski, B. Mantel, J. Périaux: *Acceleration of the convergence for viscous flow calculations*; Numeta 87, Vol. **2** (1987), C. N. Paude, J. Middleton eds., Martinus Pughoff Publishers, Dordrecht, pp. T4/1–T4/20.

[B3] P. Brown, Y. Saad: *Hybrid Krylov methods for nonlinear systems of equations*; Lawrence Livermore National Laboratory Research Report UCRL–97645, nov. 1987.

[B4] P. N. Brown: *A local convergence theory for combined inexact Newton/finite difference projection methods*; SIAM J. of Numerical Anal. **24** (1987), pp. 407–434.

[B5] M. O. Bristeau, R. Glowinski, L. Dutto, J. Périaux, G. Rogé: *Compressible viscous flow calculations using compatible finite element approximations*; International Journal for Numerical Methods in Fluids (FEMIF.7 Proceedings) (à paraître).

[B6] M. Bercovier, O. Pironneau: *Error estimates for finite element solution of the Stokes problem in the primitive variables*; Numer. Math. **33** (1979), pp. 211–224.

[B7] I. Babuška: *The finite element method with Lagrangian multipliers*; Numer. Math. **20** (1973), pp. 179–192.

[B8] F. Brezzi: *On the existence, uniqueness and approximation of saddle-point problems arising from Lagrangian multiplier*; R.A.I.R.O., Série Anal. Numer., R2 (1974), pp. 129–151.

[C1] T. Chan, K. Jackson: *Nonlinearly Preconditioned Krylov Subspaces Methods for Discrete Newton Algorithms*; SIAM J. Sci. Stat. Comput., Vol. **5**, N° 3 (1984), pp. 533–542.

[D1] J. E. Dennis, R. B. Schnabel: *Numerical Methods for Unconstrained Optimisation and Nonlinear Equations*; Prentice Hall, Englewood Cliffs, NJ, 1983.

[D2] L. Dutto: Thèse de l'Université Pierre et Marie Curie (Paris VI), 1990 (à paraître).

[D3] J. Dennis, J. Moré: *Quasi-Newton methods, motivation and theory*; SIAM Review, Vol. **19**, N° 1 (1977), pp. 46–89.

[D4] R. S. Dembo, S. C. Eisenstat, T. Steihang: *Inexact Newton methods*; SIAM J. of Numer. Anal. **19** (1982), pp. 400–408.

[F1] M. Fortin, A. Soulaimani: *Finite element approximation of compressible viscous flow*;

Proc. Computational Methods in Flow Analysis, Vol. **2** (1988), H. Niki and M. Kawahara eds., Okayama University of Sciences Press, pp. 951–956.

[G1] V. Girault, P. A. Raviart: *Finite Element Methods for Navier-Stokes Equations*; Springer–Verlag, Berlin, 1986.

[H1] L. Halpern: *Artificial Boundary Conditions for Incompletely Parabolic Perturbations of Hyperbolic Systems*; UCLA, Comp. and Applied Mathematics, CAM Report 88–20.

[L1] P. Lascaux, R. Théodor: *Analyse numérique matricielle appliquée à l'art de l'ingénieur*; Masson, Paris, Tome 1 (1986), Tome 2 (1987).

[L2] L. Landau, E. Lifchitz: *Mécanique des fluides*; MIR Moscou, 1953.

[M1] M. Mallet, J. Périaux, B. Stoufflet: *On fast Euler and Navier-Stokes solvers*; Proceedings of 7th GAMM Conf. on Numerical Methods in Fluid Mechanics, Louvain, 1987.

[O1] O. Osterby, Z. Zlater: *Direct Methods for Sparce Matrices*; Lecture Notes in Computer Science, N° **157**, Springer–Verlag, Berlin, 1983.

[O2] J. M. Ortega, W. C. Rheinboldt: *Iterative Solution on Nonlinear Equations in Several Variables*; Academic Press, New York, 1970.

[P1] A. Perronnet: *Quelques modules de traitement de la structure de donnée MUA: matrice profil. Quelques résolutions directes des systèmes linéaires de matrice profil*; MODULEF **21** (1985), INRIA (Rocquencourt), France.

[P2] O. Pironneau, J. Rappaz: *Numerical analysis for compressible viscous adiabatic stationary flows*; IMPACT of computing in Science and Engineering, Academic Press, Boston **1** (1989).

[P3] R. Peyret, T. D. Taylor: *Computational Methods for Fluid Flow*; Springer–Verlag, New York, 1982.

[R1] G. Rogé: *Sur l'approximation et l'accéleration de la convergence lors de la simulation numérique en éléments finis d'écoulements de fluides visqueux compressibles*; Thèse de l'Université Pierre et Marie Curie (Paris VI), 1990 (à paraître).

[S1] Y. Saad, M. H. Schultz: *GMRES: a Generalized Minimal Residual algorithm for solving nonsymmetric linear systems*; Research Report YALEU/DCS/RR–254 August 24, 1983; et SIAM J. Sci. Stat. Comp., Vol. **7**, N° 3 (1986), pp. 856–869.

[S2] Y. Saad: *Preconditionning techniques for nonsymmetric and indefinite linear systems*, Journal of Comp. and Applied Mathematics **24** (1988), pp. 89–105.

[S3] Y. Saad: *Practical use of some Krylov Subspace methods for solving indefinite and*

*nonsymmetric linear systems*; SIAM J. Sci. Stat. Comput., Vol. 5, N° 1 (1984), pp. 203–228.

[S4] Y. Saad: *On the Lanczos Method for Solving Symmetric Linear Systems with Several Right-Hand Sides*; Math. of Computation, Vol. **48**, N° 178 (1987), pp. 651–662.

[W1] L. B. Wigton, N. J. Yu, D. P. Young: *GMRES Acceleration of Computational fluid dynamics codes*; AIAA 7th. Computational fluid dynamics Conference, Cincinnati, Ohio, july 1985, paper 85–1494, pp. 67–74.

[Z1] Z. Zlatev: *Use of iterative refinement in the solution of sparse linear systems*; SIAM J. Numer. Anal., Vol. **19**, N° 2 (1982), pp. 381–399.

Imprimé en France
par
. l'Institut National de Recherche en Informatique et en Automatique

ISSN 0249-6399