CRS4

Center for Advanced Studies,

Research and Development in Sardinia

Uta - (CA)

Development, implementation and numerical tests of implicit methods

G. Delussu and M. Mulas

Computational Fluid Dynamics Area, CRS4, Italy

Date: November 30^{th} 2000

Contents

1 Introduction

The goal of this work was to implement an implicit version of Karalis, a Navier Stokes equations solver.

Besides the Spalart & Allmaras equation is formulated in an implicit way.

In this paper the first part is devoted to the implementation details of the implicit Navier-Stokes equations while the latter to the Spalart & Allmaras equation. The routines are then briefly described.

2 Implicit Navier-Stokes

2.1 Governing Equations

The system of Navier-Stokes equations in three dimensions in conservative form can be written

$$\begin{aligned}
\tfrac{\partial\rho}{\partial t} \quad & + \frac{\partial \rho u}{\partial x} + \frac{\partial \rho v}{\partial y} + \frac{\partial \rho w}{\partial z} = 0 \\
\tfrac{\partial\rho u}{\partial t} \quad & + \frac{\partial}{\partial x}\left(\rho u u + p - \tau_{xx}\right) + \frac{\partial}{\partial y}\left(\rho u v - \tau_{xy}\right) + \frac{\partial}{\partial z}\left(\rho u w - \tau_{xz}\right) = 0 \\
\tfrac{\partial\rho v}{\partial t} \quad & + \frac{\partial}{\partial x}\left(\rho v u - \tau_{yx}\right) + \frac{\partial}{\partial y}\left(\rho v v + p - \tau_{yy}\right) + \frac{\partial}{\partial z}\left(\rho v w - \tau_{yz}\right) = 0 \\
\tfrac{\partial\rho w}{\partial t} \quad & + \frac{\partial}{\partial x}\left(\rho w u - \tau_{zx}\right) + \frac{\partial}{\partial y}\left(\rho w v - \tau_{zy}\right) + \frac{\partial}{\partial z}\left(\rho w w + p - \tau_{xz}\right) = 0 \\
\tfrac{\partial\rho E}{\partial t} \quad & + \frac{\partial}{\partial x}\left(\rho H u - u\tau_{xx} - v\tau_{yx} - w\tau_{zx} + q_x\right) + \\
\tfrac{\partial}{\partial y} \quad & \left(\rho H v - u\tau_{xy} - v\tau_{yy} - w\tau_{zy} + q_y\right) + \frac{\partial}{\partial z}\left(\rho H w - u\tau_{xz} - v\tau_{yz} - w\tau_{zz} + q_z\right) = 0
\end{aligned} \tag{1}$$

or in a more compact way

$$\frac{\partial Q}{\partial t} + \frac{\partial \tilde{A}}{\partial x} + \frac{\partial \tilde{B}}{\partial y} + \frac{\partial \tilde{C}}{\partial z} = 0 \tag{2}$$

where Q is the array of conservative variables:

$$Q = [\rho, \rho u, \rho v, \rho w, \rho e]^T$$

and $\tilde{A}$,$\tilde{B}$,$\tilde{C}$ are the components along the three axis of the flux vector T

$$\overrightarrow{T} = \tilde{A}\overrightarrow{i} + \tilde{B}\overrightarrow{j} + \tilde{C}\overrightarrow{k}$$

In integral form the system 2, for a control volume ΔVol and with a preconditioning matrix P^{-1}, becomes

$$P^{-1}\frac{\partial}{\partial t}\int_{\Delta Vol} Q dVol + \int_{\Delta Vol}\left(\frac{\partial \tilde{A}}{\partial x} + \frac{\partial \tilde{B}}{\partial y} + \frac{\partial \tilde{C}}{\partial z}\right) dVol = 0$$

and using the Stokes theorem it changes into

$$P^{-1}\frac{\partial}{\partial t}\int_{\Delta Vol} Q dVol + \int_{\Delta S}\left(\overrightarrow{T}\cdot\overrightarrow{n}\right) dS = 0 \tag{3}$$

The flux vector $\overrightarrow{T}$ can be split in an inviscid component and in a viscid one:

$$T = F - G$$

with

$$\begin{aligned} F &= \left[\rho V, \rho u V + p\overrightarrow{i}, \rho v V + p\overrightarrow{j}, \rho w V + p\overrightarrow{k}, \rho H V\right]^T \\ G &= \left[0, \tau_{xi}, \tau_{yi}, \tau_{zi}, \tau_{ij} V_j + q\right]^T \end{aligned}$$

and substituting in 3

$$P^{-1}\frac{\partial}{\partial t}\int_{\Delta Vol} Q dVol + \int_{\Delta S}\left[\left(\overrightarrow{F} - \overrightarrow{G}\right)\cdot\overrightarrow{n}\right] dS = 0 \tag{4}$$

Finally the convective part can be written in terms of a matrix D_n multiplied by the conservative variables:

$$\overrightarrow{F}\cdot\overrightarrow{n} = F_n = D_n Q \tag{5}$$

It will be useful, for the Roe's approximate Riemann problem solver, the diagonalization of $\mid D_n \mid$ obtained by mean of the two eigenvector matrices L and R:

$$\mid D_n \mid = R \mid \Lambda \mid L \tag{6}$$

where $R = L^{-1}$ and Λ is the diagonal matrix containing the eigenvalues of D_n, that is characteristic information speed and direction.

2.2 Preconditioning

It can be introduced M as the matrix which satisfies

$$\Delta Q = M \Delta Q_v \tag{7}$$

being Q the conservatives variables and Q_v the viscous ones:

$$Q_v = [p, u, v, w, T]^T$$

Then calling M_m the Merkle modified M, it is possible to define P as follows:

$$P = M{M_m}^{-1}$$

or

$$P^{-1}M = M_m \tag{8}$$

2.3 Discretization

The governing set of equations 3 is then spatially discretized adopting a finite volume scheme, wherein the physical domain is divided into cells and the integral system is applied to each cell. Assuming an average value of Q over any cell exists, and calling it $Q_{i,j,k}$, the system 3 can be written, for any cell defined by the three indices (i, j, k), as follows:

$$P^{-1}\Delta Vol_{i,j,k}\frac{\partial Q_{i,j,k}}{\partial t} + \sum_{nfaces}^{jj=1} T_{n_{jj}}\Delta S_{jj} = 0$$

or emphasizing the different fluxes:

$$P^{-1}\Delta Vol_{i,j,k}\frac{\partial Q_{i,j,k}}{\partial t}+\sum_{nfaces}^{jj=1}\left(F_{n_{jj}}-G_{n_{jj}}\right)\Delta S_{jj}=0 \tag{9}$$

In a compact way introducing the residual $R_{i,j,k}(Q)$:

$$P^{-1}\Delta Vol_{i,j,k}\frac{\partial Q_{i,j,k}}{\partial t}=-R_{i,j,k}(Q) \tag{10}$$

Depending on whether the residual $R_{i,j,k}(Q)$ is evaluated at the previous temporal step or at the current one, an explicit or implicit formulation respectively is created.

2.4 Implicit vs Explicit formulation

2.4.1 Explicit

The system of equations 10 with an explicit Euler time-marching method can be written , dropping the subscripts (i,j,k) for convenience, as follows:

$$P^{-1}\Delta Vol\frac{Q^{n+1}-Q^n}{\Delta t}=-R(Q^n)$$

or

$$\left(P^{-1}\frac{\Delta Vol}{\Delta t}I\right)\Delta Q=-R(Q^n) \tag{11}$$

2.4.2 Implicit

The system of equations 10 with an implicit Euler time-marching method can be written, dropping the subscripts (i,j,k) for convenience, as follows:

$$P^{-1}\Delta Vol\frac{Q^{n+1}-Q^n}{\Delta t}=-R(Q^{n+1})$$

or

$$\left(P^{-1}\frac{\Delta Vol}{\Delta t}I\right)\Delta Q=-R(Q^{n+1}) \tag{12}$$

Expanding the righthanside in Taylor series about time level n:

$$R(Q^{n+1})=R(Q^n)+\left(\frac{\partial R}{\partial Q}\right)^n\Delta Q+O(\Delta Q^2) \tag{13}$$

and substituting equation 13 in system 12 , dropping the superscript n and the second order terms:

$$\left(P^{-1}\frac{\Delta Vol}{\Delta t}I\right)\Delta Q=-R(Q)-\left(\frac{\partial R}{\partial Q}\right)\Delta Q$$

or

$$\left[P^{-1}\frac{\Delta Vol}{\Delta t}I+\left(\frac{\partial R}{\partial Q}\right)\right]\Delta Q=-R(Q) \tag{14}$$

It should be underlined here that it is possible to evaluate the residual jacobian $\left(\frac{\partial R}{\partial Q}\right)$ and the residual $R(Q)$ with different approximations.

Recalling the definition of M (equation 7) the system 14 becomes

$$\left[P^{-1}\frac{\Delta Vol}{\Delta t}I + \left(\frac{\partial R}{\partial Q}\right)\right] M\Delta Q_v = -R(Q)$$

and carrying M inside the brackets:

$$\left[P^{-1}\frac{\Delta Vol}{\Delta t}IM + \left(\frac{\partial R}{\partial Q}\right)M\right] \Delta Q_v = -R(Q)$$

with some matrix algebra here shown:

$$\begin{aligned} P^{-1}\frac{\Delta Vol}{\Delta t}IM &= P^{-1}M\frac{\Delta Vol}{\Delta t}I \\ \left(\frac{\partial R}{\partial Q}\right)M &= \left(\frac{\partial R}{\partial Q_v}\right)\left(\frac{\partial Q_v}{\partial Q}\right)M = \left(\frac{\partial R}{\partial Q_v}\right)M^{-1}M = \left(\frac{\partial R}{\partial Q_v}\right) \end{aligned}$$

substituting the previous system of equations becomes

$$\left[M_m\frac{\Delta Vol}{\Delta t}I + \left(\frac{\partial R}{\partial Q_v}\right)\right] \Delta Q_v = -R(Q) \tag{15}$$

If the preconditioner is not used M_m will trasform into M, now being P the identity matrix.

2.5 Implicit Solution Schemes

Once called :

$$K = \left[M_m\frac{\Delta Vol}{\Delta t}I + \left(\frac{\partial R}{\partial Q_v}\right)\right]$$

the system 15 can be written as:

$$K\Delta Q_v = -R$$

which represents a linear system and can be seen as an iterative method, like the Newton-Raphson method, applied to the system of equation 11. At each step the linear system is solved (relaxation step) and then the solution is updated (Newton-Raphson step). ΔQ_v is a vector of dimension $5 \times ncells$ and contains all the contributions from the neigbouring cells.

2.5.1 Red/Black Point Zebra Scheme

Separating the contributions from neighbours and highlighting the indeces, the system 15 becomes:

$$\begin{aligned} A_{i,j,k}\Delta Q_{v_{i,j,k}} = -R_{i,j,k} - B_{I_{i,j,k}}\Delta Q_{v_{i-1,j,k}} - B_{J_{i,j,k}}\Delta Q_{v_{i,j-1,k}} - B_{K_{i,j,k}}\Delta Q_{v_{i,j,k-1}} + \\ -C_{I_{i,j,k}}\Delta Q_{v_{i+1,j,k}} - C_{J_{i,j,k}}\Delta Q_{v_{i,j+1,k}} - C_{K_{i,j,k}}\Delta Q_{v_{i,j,k+1}} \end{aligned} \tag{16}$$

where A,B_I,B_J,B_K,C_I,C_J,C_K are 5×5 matrices.

In other words A is the sum of ${M_m}^{-1}\frac{\Delta Vol}{\Delta t}I$ and the residuals jacobian which acts on the cell (i, j, k).

For any cell the previous system can be written as:

$$\Delta Q_{v_{i,j,k}} = A^{-1}_{i,j,k} RHS(\Delta Q_v) \tag{17}$$

A given number of relaxation steps are performed and the resulting ΔQ_v is used to update the viscous variables:

$$Q_{v_{new}} = Q_{vold} + 0.15 \Delta Q_v \tag{18}$$

where the factor 0.15 stands for stability purposes.

The relaxation steps are done according to a red/black point zebra scheme. Two main sweeps are performed at each relaxation step. The first one solves system 17 every two cells in each index direction (figure 1). In 2D can be visualized as the white squares in a check board. The latter solves system 17 on any other cell. Keeping on the previous analogy that means on the black squares. At the first iteration of relaxation all the ΔQ_v are zero and then are updated at every sweep.

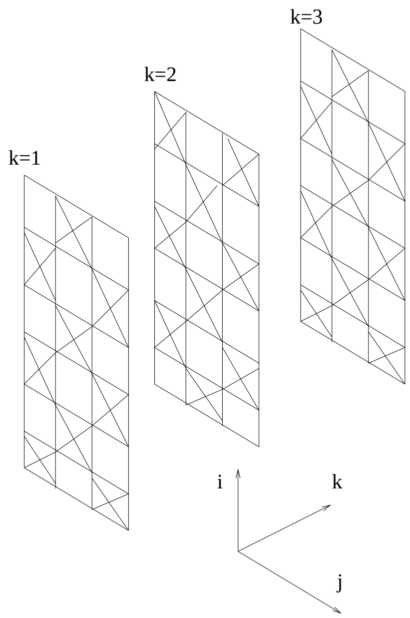

Figure 1: Point Zebra: Solved Cells by First Sweep

Table 1 resumes the steps of the solution algorithm.

1	start with initial solution of Q_v
2	calculate $A_{i,j,k}$,$B_{I_{i,j,k}}$,$B_{J_{i,j,k}}$,$B_{K_{i,j,k}}$,$C_{I_{i,j,k}}$,$C_{J_{i,j,k}}$,$C_{K_{i,j,k}}$
3	calculate $R_{i,j,k}$
4	invert $A_{i,j,k}$
5	perform relaxation steps
	a) first sweep on half of the cells (eq 17)
	b) second sweep on the remaining half (eq 17)
6	update Q_v (eq 18)

Table 1: Steps for solution on any internal cell

2.5.2 Line Zebra Scheme

In the line zebra scheme are separated all the contributions from neighbouring cells but those of the ones standing on the chosen index line. So, if i has been chosen for instance, the system 15 becomes :

$$ABC_{j,k}\Delta Q_{v_{j,k}} = -R_{j,k} - B_{J_{j,k}}\Delta Q_{v_{j-1,k}} - B_{K_{j,k}}\Delta Q_{v_{j,k-1}} - C_{J_{j,k}}\Delta Q_{v_{j+1,k}} - C_{K_{j,k}}\Delta Q_{v_{j,k+1}} \tag{19}$$

where ABC is a block tridiagonal where every block is a 5×5, therefore ABC is a $[5\times ni, 5\times ni]$, and encloses all the $A_{i,j,k}$, $B_{I_{i,j,k}}$, $C_{I_{i,j,k}}$ with i that goes from 1 to $ni-1$ for any given j, k. Figure 2 shows how ABC is assembled.

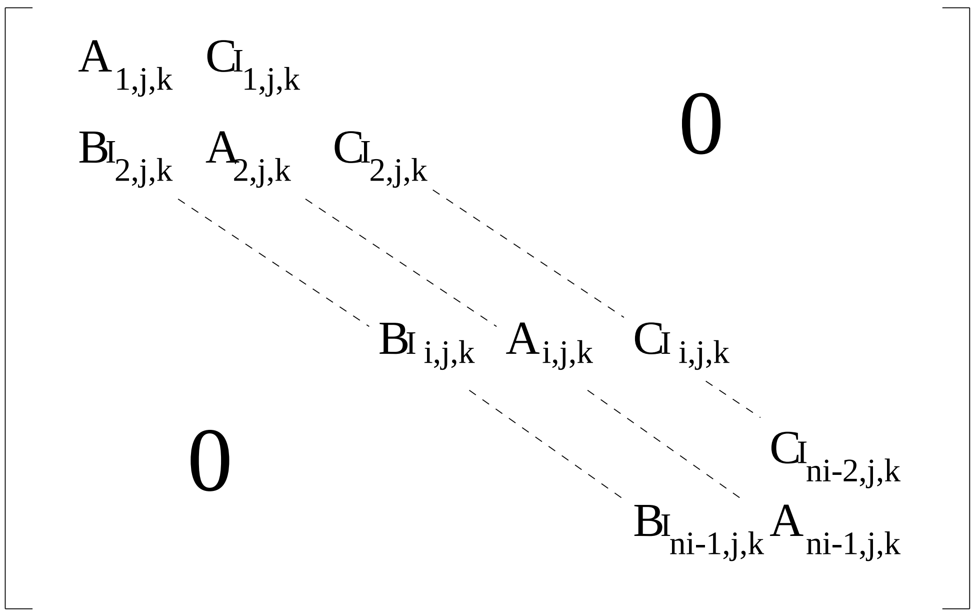

Figure 2: Matrix ABC Structure

$R_{j,k}$, $B_{J_{j,k}}$, $B_{K_{j,k}}$, $C_{J_{j,k}}$, $C_{K_{j,k}}$ are matrices $5\times ni$. They are created, for any given j,k, respectively by all the $R_{i,j,k}$, $B_{J_{i,j,k}}$,$B_{K_{i,j,k}}$,$C_{J_{i,j,k}}$,$C_{K_{i,j,k}}$ with i that goes from 1 to $ni-1$.

In order to solve the system of equations 19 firstly the LU decomposition of ABC is calculated.

A certain number of relaxation steps, set by the user, is performed and the resulting ΔQ_v is used to update the viscous variables following the equation 18.

The relaxation steps follow a line zebra scheme. Two main sweeps are performed at each relaxation step. The first one solves system 19, with ABC factorized, every two lines $i = constant$ in both direction j and k (figure 3). The latter solves system 19, with ABC factorized, on the remaining lines $i = constant$.

Table 2 resumes the steps of the solution algorithm.

2.6 Code Speedup

Two ways have been implemented in order to improve the implicit code convergence: SLOR and updating optimization. The first one aims to give a higher rate of convergence in the relaxation phase. The second one has the purpose of improving the updating of the variables, once obtained a ΔQ_v through the relaxation steps. Both have been implemented only on the line zebra scheme.

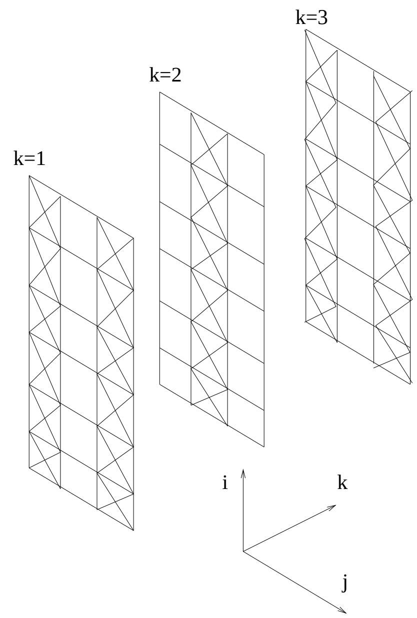

Figure 3: Line Zebra: Solved cells by first sweep (with sweep along i)

1	start with initial solution of Q_v
2	calculate $A_{i,j,k}$,$B_{I_{i,j,k}}$,$B_{J_{i,j,k}}$,$B_{K_{i,j,k}}$,$C_{I_{i,j,k}}$,$C_{J_{i,j,k}}$,$C_{K_{i,j,k}}$
3	calculate $R_{i,j,k}$
4	calculate $ABC_{i,j,k}$,$R_{j,k}$,$B_{J_{j,k}}$,$B_{K_{j,k}}$,$C_{J_{j,k}}$,$C_{K_{j,k}}$
5	compute the LU factorization of $ABC_{i,j,k}$
6	perform relaxation steps
	a) first sweep on half of the cells (eq 19 with ABC factorized)
	b) second sweep on the remaining half (eq 19 with ABC factorized)
7	update Q_v (eq 18)

Table 2: Steps for solution on any line $i = constant$ of internal cells

2.6.1 Successive Line OverRelaxation

Let's examine the table 2. At step 6 the two sweeps are done with equation 19, written here for ease of reading:

$$ABC_{j,k}\Delta Q_{v_{j,k}} = -R_{j,k} - B_{J_{j,k}}\Delta Q_{v_{j-1,k}} - B_{K_{j,k}}\Delta Q_{v_{j,k-1}} - C_{J_{j,k}}\Delta Q_{v_{j+1,k}} - C_{K_{j,k}}\Delta Q_{v_{j,k+1}}$$

Instead of solving the previous equation in SLOR method is solved the following

$$ABC_{j,k}\Delta\tilde{Q}_{v_{j,k}} = -R_{j,k} - B_{J_{j,k}}\Delta Q_{v_{j-1,k}} - B_{K_{j,k}}\Delta Q_{v_{j,k-1}} - C_{J_{j,k}}\Delta Q_{v_{j+1,k}} - C_{K_{j,k}}\Delta Q_{v_{j,k+1}} \quad (20)$$

where :

$$\Delta Q_{v_{j,k}} = \omega_{SLOR}\Delta\tilde{Q}_{v_{j,k}} + (1-\omega_{SLOR})\left(\Delta Q_{v_{j,k}}\right)_{OLD} \quad (21)$$

and ω_{SLOR} is the overrelaxation factor, which is bounded by $1 < \omega_{SLOR} < 2$. For ω_{SLOR} equal to one the scheme is not overrelaxated at all.

2.6.2 Optimization of updating

The updating of the viscous variables is performed through equation 18. The factor 0.15 has been found in literature to give a good stability to the scheme but actually slows down the convergence. Generalizing equation 18 by mean of a factor ω:

$$Q_{v_{new}} = Q_{vold} + \omega \Delta Q_v \tag{22}$$

Then it's hypotized that the root mean square of the residual chosen to be monitored by the user, is a quadratic function of ω , that is:

$$RMS_{RES} = a\omega^2 + b\omega + c$$

The minimum of this function is obtained deriving it:

$$\frac{\partial RMS_{RES}}{\partial \omega} = 2\omega a + b$$

and then setting it to zero:

$$\omega_{opt} = -\frac{b}{2a} \tag{23}$$

In order to calculate the coefficients a, b, c three evaluation of RMS, and therefore of residuals, are necessary. The user choose three ω's in input and by mean of equation 22 consequently three new set of $Q_{v_{new}}$ are obtained. Then are calculated the residuals and the RMS of the chosen residual. When an evaluation of residuals fails, i.e. gives $\pm NaN$ or $\pm\infty$, ω is cut into half and a new evaluation is tried. Calling ω_1,ω_2 and ω_3 the three ω's and RMS_1,RMS_2 and RMS_3 the corresponding RMS, the system to solve is the following:

$$\begin{bmatrix} {\omega_1}^2 & \omega_1 & 1 \\ {\omega_2}^2 & \omega_2 & 1 \\ {\omega_3}^2 & \omega_3 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} RMS_1 \\ RMS_2 \\ RMS_3 \end{bmatrix} \tag{24}$$

a, b are then substituted in equation 23 and a ω_{opt} found. A check is done on ω_{opt} if it lies in the interval $[0.15, \omega_3]$ and in case it's outside is brought back to the nearer value in the interval , i.e. 0.15 or ω_3.

2.7 Residuals Jacobian Building

In the Navier-Stokes equations the residuals jacobian receives contribution from both the inviscid convective and the viscous diffusive part of residuals. The residuals jacobian is calculated, according to the system of equations 15, with respect to the viscous variables, that is $\left(\frac{\partial R}{\partial Q_v}\right)$. Moreover it is useful to consider the part of jacobian belonging to the cell on which calculation is performed, and the part that goes to the neighbours (neighbouring cells). In order to separate those contributions are introduced the matrices A,B_I,C_I,B_J,C_J,B_K,C_K.

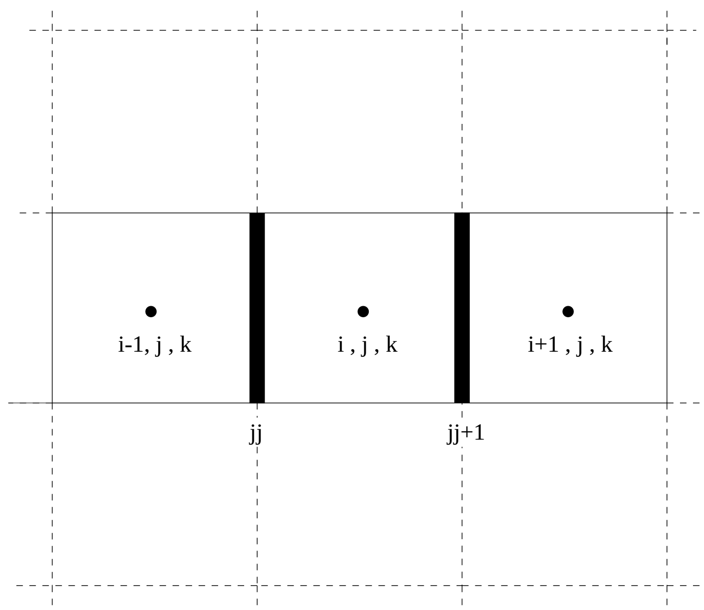

Figure 4: Cell Fluxes Scheme

The following sections will adopt the figure 4 as a reference.

2.7.1 Inviscid Part of Residuals Jacobian

The inviscid fluxes appearing in system 9 are evaluated by an upwind TVD flux difference splitting or a symmetric TVD scheme. Both approaches take advantage of the fact that F contains informations propagating through the domain with speed and direction according to the eigenvalues of the Navier-Stokes equations, i.e. along the characteristics. Upwind TVD flux difference splitting splits F into parts, each one containing the information traveling in a particular direction, and the split fluxes are upwind differenced according to their corrispondent eigenvalues. All this leads to the following formula for the convective flux at boundary face jj:

$$F_{jj} = \frac{1}{2}\left(F_{i,j,k} + F_{i-1,j,k}\right) - \frac{1}{2}P^{-1}MR_{P_v} \mid \Lambda \mid L_{P_v}\left(Q_{v_{i,j,k}} - Q_{v_{i-1,j,k}}\right) \tag{25}$$

or

$$F_{jj} = \frac{1}{2}\left(F_{i,j,k} + F_{i-1,j,k}\right) - \frac{1}{2}M_m R_{P_v} \mid \Lambda \mid L_{P_v}\left(Q_{v_{i,j,k}} - Q_{v_{i-1,j,k}}\right)$$

formula that without preconditioning becomes:

$$F_{jj} = \frac{1}{2}\left(F_{i,j,k} + F_{i-1,j,k}\right) - \frac{1}{2}MR_v \mid \Lambda \mid L_v\left(Q_{v_{i,j,k}} - Q_{v_{i-1,j,k}}\right) \tag{26}$$

Let's analyze the two equations just written, remembering expression 6. $F_{i,j,k}$ and $F_{i-1,j,k}$ are the fluxes of the cells sharing boundary face jj. R_v is the equivalent for viscous variables of R ($MR_v = R$). Again $L_v \Delta Q_v = LM^{-1}M\Delta Q = L\Delta Q$ so what's on the second term of the lefthandside is the diagonalization of D_n multiplied by $\frac{1}{2}$ and ΔQ. The preconditioned equation has the same meaning. $M_m R_{P_v}$ and $L_{P_v} M^{-1}$ diagonalize the matrix PD_n.

In order to obtain the convective residuals jacobian equation 26 is derived. Taking the approximation that tha quantity $MR_v \mid \Lambda \mid L_v$, in the followings called $\mid X_{jj} \mid$, is constant with Q_v the jacobian at boundary face jj is

$$\frac{\partial F_{jj}}{\partial Q_v} = \frac{1}{2}\left(\frac{\partial F_{i,j,k}}{\partial Q_v} + \frac{\partial F_{i-1,j,k}}{\partial Q_v}\right) - \frac{1}{2} MR_v \mid \Lambda \mid L_v \tag{27}$$

The matrix $\frac{\partial F_{i,j,k}}{\partial Q_v}$ is calculated through the formula

$$\frac{\partial F_{i,j,k}}{\partial Q_v} = \frac{\partial F_{i,j,k}}{\partial Q}\frac{\partial Q}{\partial Q_v} = D_n M$$

an in the followings will be called $J_{F_{i,j,k}}$.

Separating the contribution to the right and to the left cell , the first one is

$$\frac{\partial F_{jj}}{\partial Q_{i_v}} = \frac{1}{2} J_{F_{i,j,k}} - \frac{1}{2} \mid X_{jj} \mid \tag{28}$$

and the second one is

$$\frac{\partial F_{jj}}{\partial Q_{i-1_v}} = \frac{1}{2} J_{F_{i-1,j,k}} + \frac{1}{2} \mid X_{jj} \mid \tag{29}$$

Writing the two contribution at the boundary face $JJ+1$:

$$\frac{\partial F_{jj+1}}{\partial Q_{i_v}} = \frac{1}{2} J_{F_{i,j,k}} + \frac{1}{2} \mid X_{jj+1} \mid \tag{30}$$

$$\frac{\partial F_{jj+1}}{\partial Q_{i+1_v}} = \frac{1}{2} J_{F_{i+1,j,k}} - \frac{1}{2} \mid X_{jj+1} \mid \tag{31}$$

Assuming that a flux is positive when is directed outward, with respect to the chosen cell, the flux F_{jj+1} is positive and F_{jj} is negative.

Now recalling that:

$$\begin{aligned} A_{i,j,k} &= \sum_{boundaries} \frac{\partial F_{boundary}}{\partial Q_{i_v}} \Delta S_{boundary} \\ B_{I_{i,j,k}} &= \sum_{boundaries} \frac{\partial F_{boundary}}{\partial Q_{i-1_v}} \Delta S_{boundary} \\ C_{I_{i,j,k}} &= \sum_{boundaries} \frac{\partial F_{boundary}}{\partial Q_{i+1_v}} \Delta S_{boundary} \end{aligned}$$

and adding the contributions from the boundary faces jj and $jj+1$, the matrices $A_{i,j,k}$,$B_{I_{i,j,k}}$,$C_{I_{i,j,k}}$ can be filled:

$$A_{i,j,k} = A_{i,j,k} - \left(J_{F_{i,j,k}} - \mid X_{jj} \mid\right) \frac{\Delta S_{jj}}{2} + \left(J_{F_{i,j,k}} + \mid X_{jj+1} \mid\right) \frac{\Delta S_{jj+1}}{2}$$

$$
\begin{aligned}
B_{I_{i,j,k}} &= B_{I_{i,j,k}} - \left(J_{F_{i-1,j,k}} + \mid X_{jj} \mid\right) \frac{\Delta S_{jj}}{2} \\
C_{I_{i,j,k}} &= C_{I_{i,j,k}} + \left(J_{F_{i+1,j,k}} - \mid X_{jj+1} \mid\right) \frac{\Delta S_{jj+1}}{2}
\end{aligned}
$$

Examining the expressions just written , recursive contributions can be deducted. At each boundary face jj three contribution are then added:

$$
\begin{array}{lllllll}
J_{F_{i,j,k}} \dfrac{\Delta S_{jj}}{2} & is & given & to & A_{i,j,k} & with\ \ sign & minus \\
 & & and & to & C_{I_{i-1,j,k}} & with\ \ sign & plus \\
 & & & & & & \\
J_{F_{i-1,j,k}} \dfrac{\Delta S_{jj}}{2} & is & given & to & B_{I_{i,j,k}} & with\ \ sign & minus \\
 & & and & to & A_{i-1,j,k} & with\ \ sign & plus \\
 & & & & & & \\
\mid X_{jj} \mid \dfrac{\Delta S_{jj}}{2} & is & given & to & A_{i,j,k} & with\ \ sign & plus \\
 & & and & to & B_{I_{i,j,k}} & with\ \ sign & minus \\
 & & and & to & A_{i-1,j,k} & with\ \ sign & plus \\
 & & and & to & C_{I_{i-1,j,k}} & with\ \ sign & minus
\end{array}
\tag{32}
$$

2.7.2 Viscous Part of Residuals Jacobian

The viscous fluxes for the Navier-Stokes equations (see system 1) at boundary face jj are given by:

$$
\begin{aligned}
G_{jj_\rho} &= 0 \\
G_{jj_{\rho u}} &= \tau_{xx}n_x + \tau_{xy}n_y + \tau_{xz}n_z \\
G_{jj_{\rho v}} &= \tau_{yx}n_x + \tau_{yy}n_y + \tau_{yz}n_z \\
G_{jj_{\rho w}} &= \tau_{zx}n_x + \tau_{zy}n_y + \tau_{zz}n_z \\
G_{jj_{\rho e}} &= \left(u\tau_{xx} + v\tau_{yx} + w\tau_{zx} + q_x\right)n_x + \\
&\quad \left(u\tau_{xy} + v\tau_{yy} + w\tau_{zy} + q_y\right)n_y + \left(u\tau_{xz} + v\tau_{yz} + w\tau_{zz} + q_z\right)n_z
\end{aligned}
\tag{33}
$$

where:

$$
\begin{aligned}
\tau_{xx} &= 2\mu\frac{\partial u}{\partial x} - \frac{2}{3}\mu\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right) \\
\tau_{xy} &= \tau_{yx} = \mu\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) \\
\tau_{xz} &= \tau_{zx} = \mu\left(\frac{\partial u}{\partial z} + \frac{\partial w}{\partial x}\right) \\
\tau_{yy} &= 2\mu\frac{\partial v}{\partial y} - \frac{2}{3}\mu\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right) \\
\tau_{yz} &= \tau_{zy} = \mu\left(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}\right) \\
\tau_{zz} &= 2\mu\frac{\partial w}{\partial z} - \frac{2}{3}\mu\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right) \\
q_x &= -\kappa\frac{\partial T}{\partial x} \\
q_y &= -\kappa\frac{\partial T}{\partial y} \\
q_z &= -\kappa\frac{\partial T}{\partial z}
\end{aligned}
$$

The viscous fluxes can be written in terms of the spatial gradients of the viscous variables as:

$$
\begin{aligned}
G_{jj} &= \left(R_{xx}\frac{\partial u}{\partial x} + R_{xy}\frac{\partial u}{\partial y} + R_{xz}\frac{\partial u}{\partial z}\right) + \\
&\quad + \left(R_{yx}\frac{\partial v}{\partial x} + R_{yy}\frac{\partial v}{\partial y} + R_{yz}\frac{\partial v}{\partial z}\right) + \\
&\quad + \left(R_{zx}\frac{\partial w}{\partial x} + R_{zy}\frac{\partial w}{\partial y} + R_{zz}\frac{\partial w}{\partial z}\right) + R_{xt}\frac{\partial T}{\partial x} + R_{yt}\frac{\partial T}{\partial y} + R_{zt}\frac{\partial T}{\partial z}
\end{aligned}
$$

In section 5.2 all the $R_{\alpha\beta}$ are presented. Deriving the previous expression with respect to the viscous variables, assuming all the $R_{\alpha\beta}$ constants, the viscous flux jacobian at boundary face jj assigned to the right cell is

$$\begin{aligned}\frac{\partial G_{jj}}{\partial Q_{i_v}} &= -\left(R_{xx}+R_{xy}+R_{xz}\right)\frac{ds_x}{ds^2}+ \\ &-\left(R_{yx}+R_{yy}+R_{yz}\right)\frac{ds_y}{ds^2}-\left(R_{zx}+R_{zy}+R_{zz}\right)\frac{ds_z}{ds^2}-R_{xt}\frac{ds_x}{ds^2}-R_{yt}\frac{ds_y}{ds^2}-R_{zt}\frac{ds_z}{ds^2}=-Z_{jj}\end{aligned} \tag{34}$$

where Z is introduced for sake of future readability and $\overrightarrow{ds}$ is the vector connecting cell centers across face jj. The three terms $\frac{ds_x}{ds^2},\frac{ds_y}{ds^2},\frac{ds_z}{ds^2}$ can be approximed by:

$$\begin{aligned}\frac{ds_x}{ds^2} &= \frac{\frac{Vol_m}{S_{jj}{}^2}S_{jj_x}}{\frac{Vol_m{}^2}{S_{jj}{}^2}}=\frac{S_{jj_x}}{Vol_m} \\ \frac{ds_y}{ds^2} &= \frac{\frac{Vol_m}{S_{jj}{}^2}S_{jj_y}}{\frac{Vol_m{}^2}{S_{jj}{}^2}}=\frac{S_{jj_y}}{Vol_m} \\ \frac{ds_z}{ds^2} &= \frac{\frac{Vol_m}{S_{jj}{}^2}S_{jj_z}}{\frac{Vol_m{}^2}{S_{jj}{}^2}}=\frac{S_{jj_z}}{Vol_m}\end{aligned}$$

where $Vol_m=\frac{Vol_{i,j,k}+Vol_{i-1,j,k}}{2}$ is the average volume of the two cells.

Making more clear the different contribution for any viscous variable:

$$\begin{aligned}\frac{\partial G_{jj}}{\partial p_i} &= 0 \\ \frac{\partial G_{jj}}{\partial u_i} &= -\left(R_{xx}+R_{xy}+R_{xz}\right) \\ \frac{\partial G_{jj}}{\partial v_i} &= -\left(R_{yx}+R_{yy}+R_{yz}\right) \\ \frac{\partial G_{jj}}{\partial w_i} &= -\left(R_{zx}+R_{zy}+R_{zz}\right) \\ \frac{\partial G_{jj}}{\partial T_i} &= -R_{xt}-R_{yt}-R_{zt}\end{aligned}$$

Returning to equation 34 and applying it to the left cell at boundary face jj:

$$\frac{\partial G_{jj}}{\partial Q_{i-1_v}}=Z_{jj} \tag{35}$$

Likewise writing the equations 34 and 35 at boundary face $jj+1$:

$$\frac{\partial G_{jj+1}}{\partial Q_{i+1_v}} = -Z_{jj+1} \tag{36}$$

$$\frac{\partial G_{jj+1}}{\partial Q_{i_v}} = Z_{jj+1} \tag{37}$$

Now adding the contributions from the boundary faces jj and $jj+1$, the matrices $A_{i,j,k}$,$B_{I_{i,j,k}}$,$C_{I_{i,j,k}}$ can be filled:

$$\begin{aligned} A_{i,j,k} &= A_{i,j,k} + Z_{jj}\Delta S_{jj} + Z_{jj+1}\Delta S_{jj+1} \\ B_{I_{i,j,k}} &= B_{I_{i,j,k}} - Z_{jj}\Delta S_{jj} \\ C_{I_{i,j,k}} &= C_{I_{i,j,k}} - Z_{jj+1}\Delta S_{jj+1} \end{aligned}$$

Examining the expressions just written , recursive contributions can be deducted. At each boundary face jj:

$$\begin{array}{cccccccc} Z_{jj} & is & given & to & A_{i,j,k} & with & sign & plus \\ & & and & to & C_{I_{i-1,j,k}} & with & sign & minus \\ & & and & to & B_{I_{i,j,k}} & with & sign & minus \\ & & and & to & A_{i-1,j,k} & with & sign & plus \end{array} \tag{38}$$

2.8 Results

In the following results sections the contribution to the jacobian from the viscous fluxes has'nt been considered. In other words the jacobian is built only from the inviscid fluxes. Furthermore, unless specified, all the calculations have been conducted with the symmetric TVD scheme.

2.8.1 Cylinder

As inviscid benchmark has been chosen a cylinder, represented in fig 5, washed by a potential flow. The boundary conditions applied are:

$$\begin{aligned}
(P)_\infty &= 101325.Pa \\
(T)_\infty &= 293.K \\
(\rho)_\infty &= 1.2049\frac{Kg}{m^3} \\
(u)_\infty &= 30.\frac{m}{s} \\
(v)_\infty &= 0.\frac{m}{s} \\
(w)_\infty &= 0.\frac{m}{s}
\end{aligned}$$

In order to minimize the perturbations on the flow around the cylinder, the boundary $BC236$ has been preferred to the boundary $BC221$. The cylinder mesh has the following number of points: 97 along i , 33 along j and 2 along k. j is the radial index while i is the circumferential one.

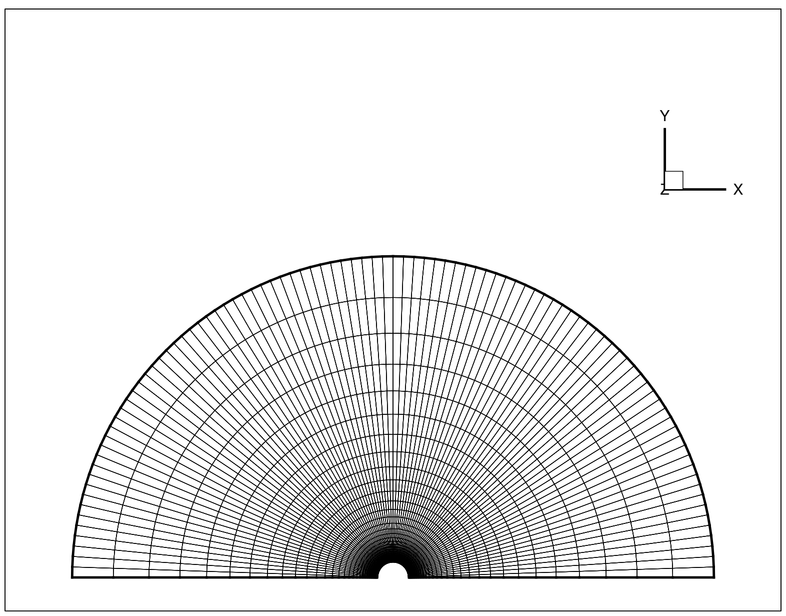

Figure 5: Cylinder mesh

In figures 6 is shown the x-momentum convergence of the implicit and explicit schemes. It is evident the better behaviour of the implicit schemes. The time consumption is reported in table 3, where $Nitrel$ is the number of iterations of relaxation.

Scheme	Nitrel	Time (s)
Explicit	-	502
Implicit red/black	50	2641
Implicit line zebra along I	50	4160

Table 3: Cylinder: Different schemes time consumption for 1000 iterations

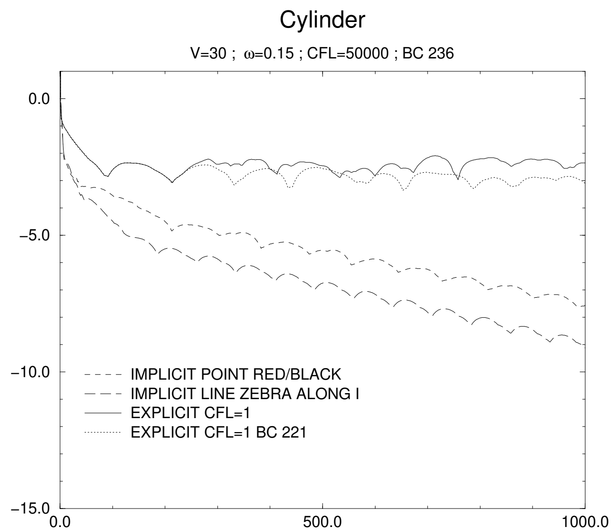

Figure 6: Cylinder: x-momentum convergence

The pressure coefficient comparison between the explicit and line zebra implicit is reported in figure 7. C_P is evalued on the boundary cells of the cylinder. The angle *alpha* is assumed starting from the leading edge and therefore takes the value 180 at the trailing edge.

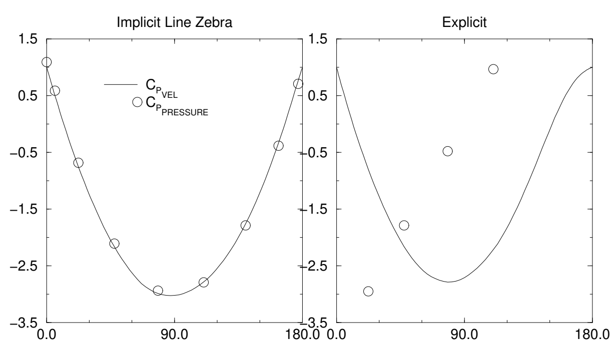

Figure 7: Cylinder: Pressure Coefficient Comparison

The C_P field has been calculated in two ways, so two C_P's have been introducted: the $C_{P_{PRESSURE}}$ and the $C_{P_{VEL}}$. The first one comes from the definition of C_P:

$$C_{P_{PRESSURE}} = \frac{(p - p_\infty)}{\frac{1}{2}\rho {V_\infty}^2} \qquad (39)$$

and remembering that, under the assumption of steady, inviscid and adiabatic flow, the Bernoulli equation is valid:

$$p_\infty + \frac{1}{2}\rho_\infty {V_\infty}^2 = p + \frac{1}{2}\rho V^2$$

the equation 39 becomes :

$$C_{P_{VEL}} = 1 - \left(\frac{V}{V_\infty}\right)^2 \tag{40}$$

The exact, from theory, full potential solution is:

$$C_P = 1 - 4\sin^2\alpha$$

While the line zebra implicit obtains a good agreement in both the C_P's the explicit has a very bad $C_{P_{PRESSURE}}$ field. The reason must lie on the poor convergence of explicit that causes more damages on the more sensitive field that is the pressure one, because of the smaller variations.

In fact the difference between the results for a convergence of $1.E-5$ of initial residual and $1.E-9$ are not much different. Figure 8 shows on the C_P's field at 100 and 1000 iterations for the line zebra implicit.

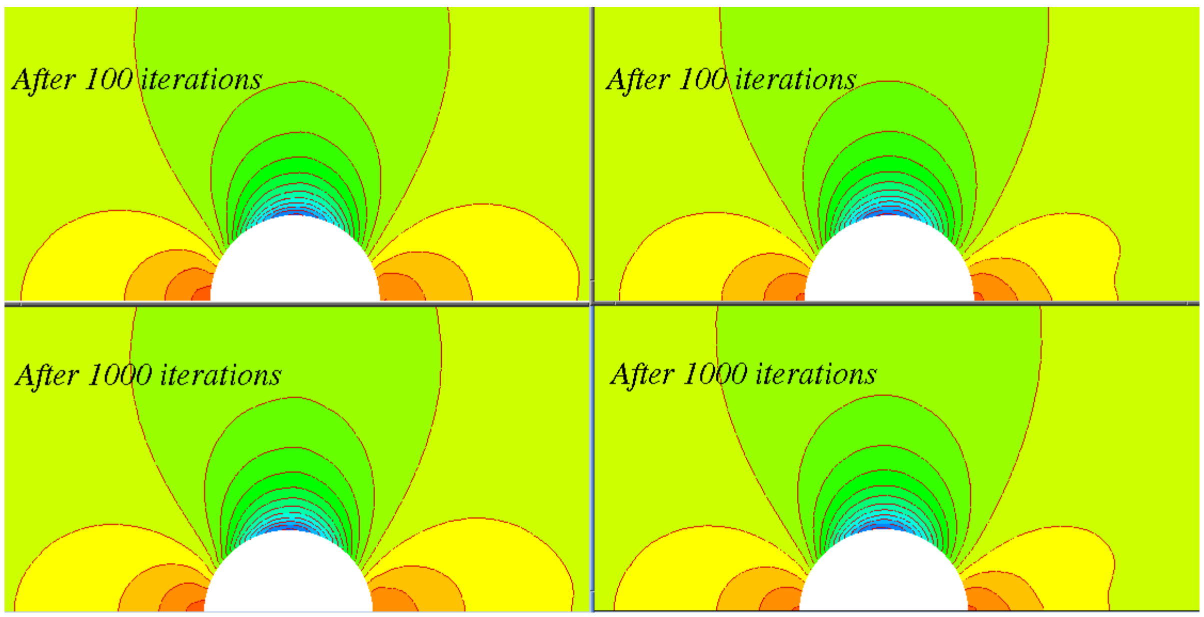

Figure 8: Cylinder: On the left column $C_{P_{PRESSURE}}$ field, on the right $C_{P_{VEL}}$ field after 100 and 1000 iterations

Figure 9 shows the effect of parameter ω on the convergence. Starting from scratch the bigger ω that can be used is 0.33. Once stabilized, after one hundred iterations, the ω can be raised to 0.5. The improvement in convergence is manifest.

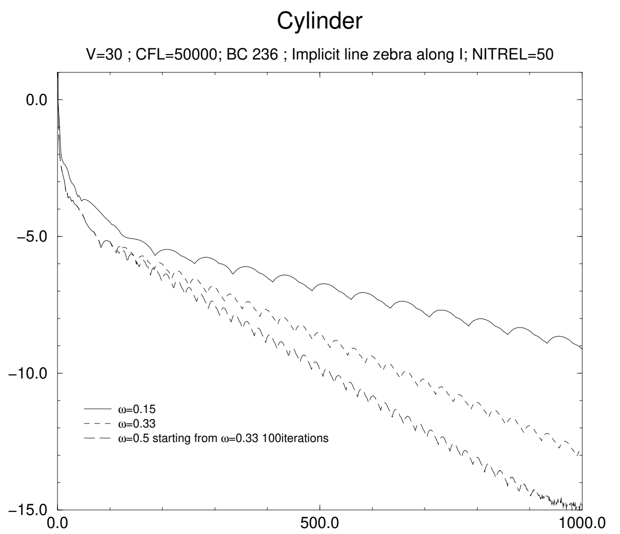

Figure 9: Cylinder: Effect of ω on convergence

Figures 10 and 11 display the behaviour of the x-momentum convergence with respect to the variation of $Nitrel$, starting respectevely from scratch and from a solution stabilized after one hundred iterations.

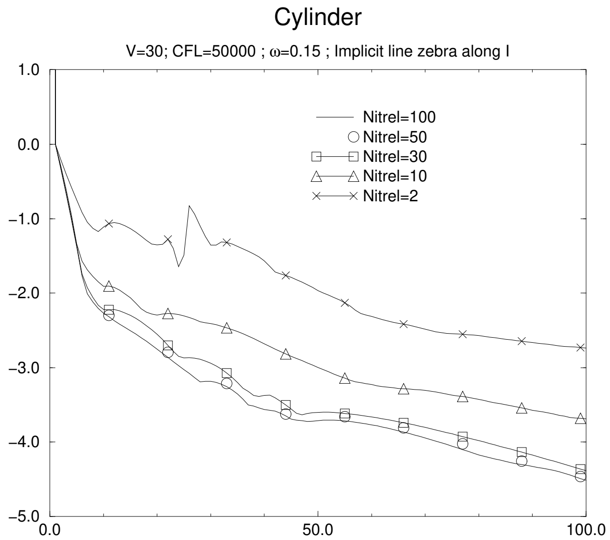

Figure 10: Cylinder: Effect of $Nitrel$ on convergence starting from scratch

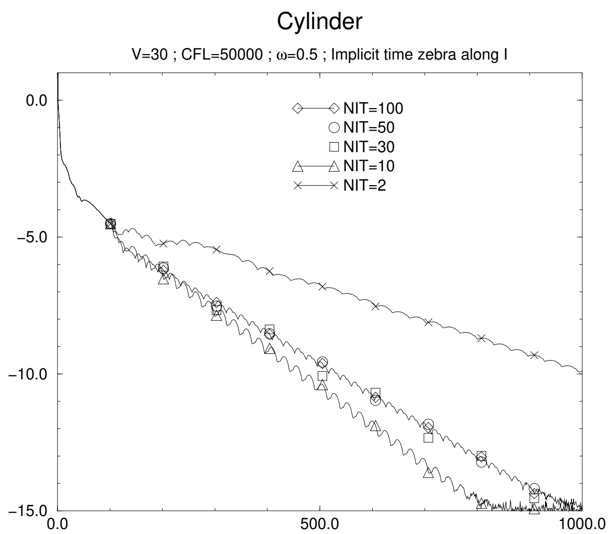

Figure 11: Cylinder: Effect of $Nitrel$ on convergence starting from 100 iterations

Tables 4 and 5 resume the time consumption for these cases.

Nitrel	Time (s)
2	86
10	141
30	283
50	430
100	772

Table 4: Cylinder: Time consumption for different $Nitrel$ starting from scratch

Nitrel	Time (s)
2	791
10	1262
30	2570
50	3861
100	7003

Table 5: Cylinder: Time consumption for different $Nitrel$ starting from 100 iterations

The impact of sweeping on different indeces is shown in figure 12. Being the mesh bidimensional, sweeping along k is equivalent to a point red/black scheme. The better results are obtained sweeping along i. The reason could be that the mesh is stretched along j so the variation between a cell and its neighbour along that direction is relatively small, while the variation along i instead is more strong and a line zebra better capture it.

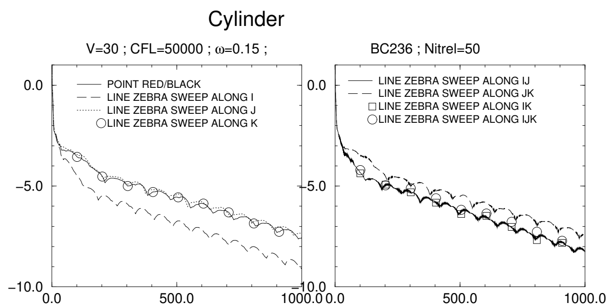

Figure 12: Cylinder: Effect of sweeping along different indeces on convergence

Table 6 shows the different time consumption to get to 1000 iterations for the different sweeps.

Sweep along	Time (s)
I	4160
J	4686
K	3717
IJ	4037
JK	3777
IK	3951
IJK	4165

Table 6: Cylinder: Time consumption for different sweep directions with Nitrel=50

Figure 13 is about the influence of SLOR factor, ω_{SLOR}, on convergence. There is no convergence for values of ω_{SLOR} bigger than 1.12. It can be seen that, at least in this case, this influence is slightly negative.

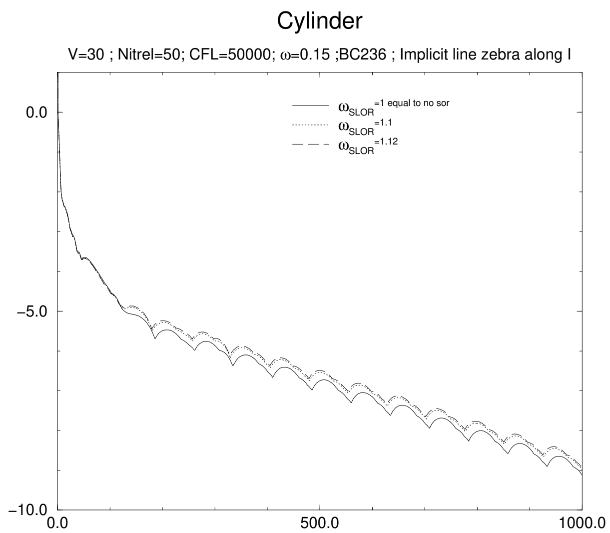

Figure 13: Cylinder: Effect of SLOR on convergence

The effect of ω has been seen previously. A good solver should choose the optimum ω at every step. In order to do so, as said in section 2.6, an ω variable has been implemented. Figure 14 shows the improvement in x-momentum convergence given by the ω variable. Three ω's must be chosen and two different sets, between all the possible choices, are shown. In fact there's not much difference between them. Figure 15 shows the ω_{opt} computed and bounded. It should be reminded here that if ω_{opt} exceeds the extremes ,the minimum 0.15 and the maximum ω_3, takes the value of the nearer extreme; in that sense it is called bounded.

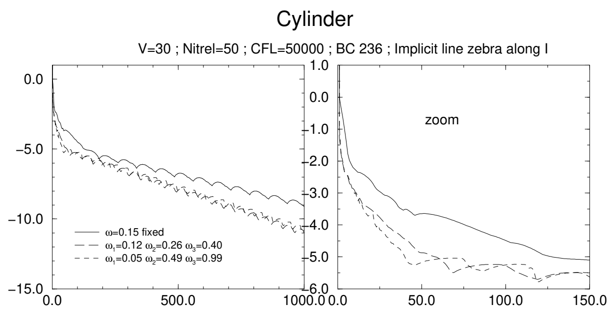

Figure 14: Cylinder: Effect of ω variable on convergence

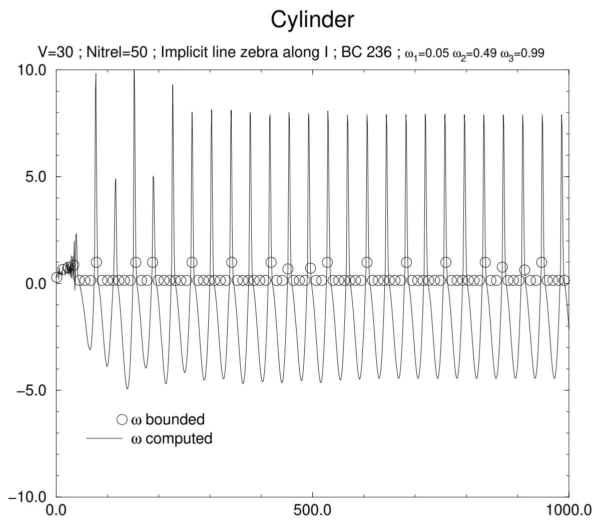

Figure 15: Cylinder: ω_{opt} computed and bounded

Table 7 resumes the time consumption for the test of figure 14.

ω along	Iterations	Time (s)
whole iterations		
ω=0.15 fixed	1000	4160
ω_1=0.12,ω_2=0.26,ω_3=0.40	1000	4301
ω_1=0.05,ω_2=0.49,ω_3=0.99	1000	4301
zoomed part		
ω=0.15 fixed	150	384
ω_1=0.12,ω_2=0.26,ω_3=0.40	150	416
ω_1=0.05,ω_2=0.49,ω_3=0.99	150	416

Table 7: Cylinder: Time consumption for ω fixed and variable

2.8.2 Flat Plate

As viscous benchmark has been chosen a flat plate, represented in fig 16, run over by a current of perfect gas without incidence. The number of points along the three indices are: 40 along i, 73 along j and 2 along k. i is oriented like x, j like y and k like z. The inlet is at face $i = 1$ and the outlet at face $i = ni$. The plate starts at $i = 17$ instead of $i = 1$ to avoid the double boundary condition at line $i = 1$. A pressure difference is imposed on the field. In detail:

$$\begin{aligned} at \quad inlet & \\ & P_{tot} &= 29041.75 Pa \\ at \quad outlet & \\ & P &= 28831. Pa \end{aligned}$$

That determines a Reynolds number of $1.76E5$ smaller than $5.E5$, the critic Reynolds number for a flat plate, so it's a laminar flow. The mach is slightly bigger than 0.1 and the preconditioner has not been used.

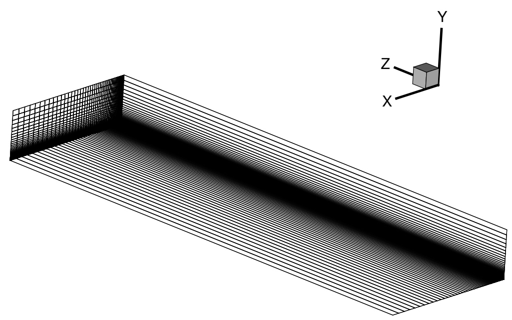

Figure 16: Flate Plate: Sketch of mesh used

Figure 17 shows the x-momentum convergence of the explicit, implicit point red/black and implicit line zebra schemes.

In figure 18 the fraction $\frac{V}{V_e}$ is plotted versus η. η is the adimensional vertical coordinate defined by:

$$\eta = \frac{y}{\delta}$$

and δ is the geometric boundary layer thickness defined by the value of y at which the vecloity in the boundary layer has reached $0.99V_e$,being $V_e = V_\infty$ the free stream velocity. The exact profile has been calculated by Blasius. That profile $\frac{V}{V_e}(\eta)$ has the feature of self-similarity along the coordinate x.

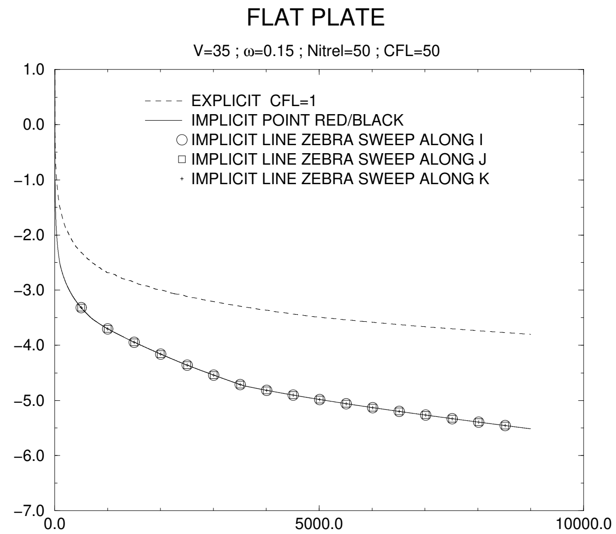

Figure 17: Flate Plate: x-momentum convergence

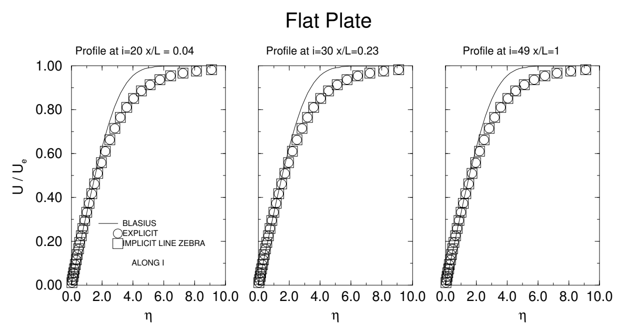

Figure 18: Flate Plate: Adimensional computed profiles versus Blasius profile

The geometric boundary layer thickness calculated by mean of the blasius law for the laminar plate:

$$\delta = \frac{5x}{\sqrt{Re_x}}$$

gives the value 0.00314 at $x = L$ while both explicit and implicit gives 0.00336. In section 5.3 are reminded some relations for the laminar boundary layer in a flat plate

3 Implicit Spalart-Allmaras

3.1 Governing equation

The Spalart & Allmaras is a one-equation model for the eddy or turbulent viscosity ν_t, which added to ν, the molecular or laminar viscosity, gives the total viscosity ν_g . In its general form is:

$$\frac{D\nu_t}{Dt} = c_{b1} S \nu_t + \frac{1}{\sigma}\left[\nabla \cdot (\nu_t \nabla \nu_t) + c_{b2} (\nabla \nu_t)^2\right] - c_{\omega 1} f_\omega \left(\frac{\nu_t}{d}\right)^2$$

that, multiplying both side by ρ and remembering:

$$\rho\frac{D\nu_t}{Dt} = \frac{\partial \rho \nu_t}{\partial t} + \frac{\partial \rho V_j \nu_t}{\partial x_j}$$

becomes

$$\frac{\partial \rho \nu_t}{\partial t} + \frac{\partial \rho u \nu_t}{\partial x} + \frac{\partial \rho v \nu_t}{\partial y} + \frac{\partial \rho w \nu_t}{\partial z} = \rho c_{b1} S \nu_t + \frac{1}{\sigma}\left[\nabla \cdot (\rho \nu_t \nabla \nu_t) + \rho c_{b2} (\nabla \nu_t)^2\right] - \rho c_{\omega 1} f_\omega \left(\frac{\nu_t}{d}\right)^2 \quad (41)$$

where:

$$\begin{array}{cl}
\nu_t & \textit{is the eddy viscosity} \\
S & \textit{is equal to the vorticity} \\
c_{b1}, \sigma, c_{b2} & \textit{are empirical constants} \\
d & \textit{is the distance to the wall} \\
f_\omega = & g\left[\dfrac{1 + {c_{\omega 3}}^6}{g^6 + {c_{\omega 3}}^6}\right]^{\frac{1}{6}} \\
g = & r + c_{\omega 2}\left(r^6 - r\right) \\
r = & \dfrac{\nu_t}{S \kappa^2 d^2} \\
c_{\omega 1}, c_{\omega 2}, c_{\omega 3} & \textit{are empirical constants} \\
\kappa & \textit{is the Karman constant}
\end{array}$$

To cope with the buffer layer and viscous sublayer, equation 41 can be arranged and expressed in terms of $\tilde{\nu}$ equal to ν_t except in those zones. In mathematical terms:

$$\tilde{\nu} = \frac{\nu_t}{f_{v1}}$$

with:

$$\begin{aligned}
f_{v1} &= \frac{\chi}{\chi^3 + {c_{v1}}^3} \\
\chi &= \frac{\tilde{\nu}}{\nu} = \kappa y^+ \\
y^+ &= \frac{d}{l_\tau} \\
l_\tau &= \frac{\nu}{V_\tau} \\
V_\tau &= \sqrt{\frac{\tau_p}{\rho}}
\end{aligned}$$

being τ_p the viscous stress evaluated at wall boundary. This leads to the following equation for $\tilde{\nu}$:

$$\frac{\partial\rho\tilde{\nu}}{\partial t}+\frac{\partial\rho u\tilde{\nu}}{\partial x}+\frac{\partial\rho v\tilde{\nu}}{\partial y}+\frac{\partial\rho w\tilde{\nu}}{\partial z}=\rho c_{b1}\tilde{S}\tilde{\nu}+\frac{1}{\sigma}\left[\nabla\cdot\left(\left(\rho\nu+\rho\tilde{\nu}\right)\nabla\tilde{\nu}\right)+\rho c_{b2}\left(\nabla\tilde{\nu}\right)^2\right]-\rho c_{\omega 1}f_\omega\left(\frac{\tilde{\nu}}{d}\right)^2 \quad (42)$$

where:

$$\begin{aligned}\tilde{S} &= S+\frac{\tilde{\nu}}{\kappa^2 d^2}f_{v2}\\ f_{v2} &= 1-\frac{\chi}{1+\chi f_{v1}}\end{aligned}$$

3.2 Discretization

The governing equation 42 is spatially discretized following the indication of section 2.3. Equation 42 becomes

$$\Delta Vol_{i,j,k}\frac{\partial\rho\tilde{\nu}}{\partial t}+\sum_{nfaces}^{jj=1}L_{n_{jj}}\Delta S_{jj}=\Delta Vol_{i,j,k}\left(PR-DE\right) \quad (43)$$

being $L_{n_{jj}}$ the sum of convective and viscous fluxes and calling PR and DE the production and destruction term respectively. They have the following form:

$$\begin{aligned}PR &= \rho c_{b1}\tilde{S}\tilde{\nu}\\ DE &= \rho c_{\omega 1}f_\omega\left(\frac{\tilde{\nu}}{d}\right)^2\end{aligned}$$

In a compact way introducing the residual $R_{i,j,k}(\tilde{\nu})$:

$$\Delta Vol_{i,j,k}\frac{\partial\rho\tilde{\nu}}{\partial t}=-R_{i,j,k}(\tilde{\nu}) \quad (44)$$

As said before, depending on whether the residual $R_{i,j,k}(\tilde{\nu})$ is evaluated at the previous temporal step or at the current one, an explicit or implicit formulation respectively is created.

3.3 Implicit vs Explicit formulation

3.3.1 Explicit

The equation 44, with an explicit Euler time-marching method, can be written, dropping the subscripts (i,j,k) for convenience and making the approximation

$$\rho^{n+1}\tilde{\nu}^{n+1}-\rho^n\tilde{\nu}^n\approx\rho^{n+1}\left(\tilde{\nu}^{n+1}-\tilde{\nu}^n\right)$$

it is possible to write:

$$\rho\Delta Vol\frac{\tilde{\nu}^{n+1}-\tilde{\nu}^n}{\Delta t}=-R(\tilde{\nu}^n)$$

or

$$\left(\frac{\Delta Vol}{\Delta t}I\right)\Delta\tilde{\nu}=-\frac{R(\tilde{\nu}^n)}{\rho} \quad (45)$$

3.3.2 Implicit

The equation 44, with an implicit Euler time-marching method, can be written, dropping the subscripts (i, j, k) for convenience and making the approximation

$$\rho^{n+1}\tilde{\nu}^{n+1} - \rho^n\tilde{\nu}^n \approx \rho^{n+1}\left(\tilde{\nu}^{n+1} - \tilde{\nu}^n\right)$$

it is possible to write:

$$\rho\Delta Vol\frac{\tilde{\nu}^{n+1} - \tilde{\nu}^n}{\Delta t} = -R(\tilde{\nu}^{n+1})$$

or

$$\left(\rho\frac{\Delta Vol}{\Delta t}I\right)\Delta\tilde{\nu} = -R(\tilde{\nu}^{n+1}) \tag{46}$$

Expanding the rightanside in Taylor series about time level n:

$$R(\tilde{\nu}^{n+1}) = R(\tilde{\nu}^n) + \left(\frac{\partial R}{\partial\tilde{\nu}}\right)^n\Delta\tilde{\nu} + O(\Delta\tilde{\nu}^2) \tag{47}$$

and substituting equation 47 in equation 46 , dropping the superscript n and the second order terms:

$$\left(\rho\frac{\Delta Vol}{\Delta t}I\right)\Delta\tilde{\nu} = -R(\tilde{\nu}) - \left(\frac{\partial R}{\partial\tilde{\nu}}\right)\Delta\tilde{\nu}$$

and :

$$\left[\rho\frac{\Delta Vol}{\Delta t}I + \left(\frac{\partial R}{\partial\tilde{\nu}}\right)\right]\Delta\tilde{\nu} = -R(\tilde{\nu}) \tag{48}$$

It is possible to evaluate the residual jacobian $\left(\frac{\partial R}{\partial\tilde{\nu}}\right)$ and the residual $R(\tilde{\nu})$ with different approximations.

In fact at present only the destructive and convective terms have been derived , so equation 48 should be written as

$$\left[\rho\frac{\Delta Vol}{\Delta t}I + \left(\frac{\partial R_{conv+destr}}{\partial\tilde{\nu}}\right)\right]\Delta\tilde{\nu} = -R(\tilde{\nu}) \tag{49}$$

3.4 Implicit Solution Schemes

Once called :

$$K = \left[\rho\frac{\Delta Vol}{\Delta t}I + \left(\frac{\partial R_{conv+destr}}{\partial\tilde{\nu}}\right)\right]$$

equation 49 can be written as

$$K\Delta\tilde{\nu} = -R$$

which represents a linear equation and can be interpreted as an iterative method like the Newton-Raphson method applied to the equation 45. At each step the linear system is solved (relaxation step) and then the solution is updated (Newton-Raphson step). $\Delta\tilde{\nu}$ is a vector of dimension $ncells$ and contains all the contributions from the neigbouring cells.

3.4.1 Red/Black Point Zebra Scheme

Separating the neighbours contributions and highlighting the indeces:

$$\begin{aligned}ASA_{i,j,k}\Delta\tilde{\nu}_{i,j,k} = -R_{i,j,k} - BSA_{I_{i,j,k}}\Delta\tilde{\nu}_{i-1,j,k} - BSA_{J_{i,j,k}}\Delta\tilde{\nu}_{i,j-1,k} - BSA_{K_{i,j,k}}\Delta\tilde{\nu}_{i,j,k-1} + \\ -CSA_{I_{i,j,k}}\Delta\tilde{\nu}_{i+1,j,k} - CSA_{J_{i,j,k}}\Delta\tilde{\nu}_{i,j+1,k} - CSA_{K_{i,j,k}}\Delta\tilde{\nu}_{i,j,k+1}\end{aligned} \tag{50}$$

where ASA,BSA_I,BSA_J,BSA_K,CSA_I,CSA_J,CSA_K are arrays, and for a single cell scalars.

In other words ASA is the sum of $\rho\frac{\Delta Vol}{\Delta t}I$ and the residuals jacobian which acts on the cell (i, j, k).

For any cell the previous equation can be written as :

$$\Delta\tilde{\nu}_{i,j,k} = ASA^{-1}_{i,j,k}RHS(\Delta\tilde{\nu}) \tag{51}$$

A given number of relaxation steps is performed and the resulting $\Delta\tilde{\nu}$ is used to update the viscous variables through a ρ weighed formula:

$$\tilde{\nu}_{new} = \tilde{\nu}_{old}\frac{\rho_{old}}{\rho_{new}} + 0.15\Delta\tilde{\nu} \tag{52}$$

The relaxation steps follow a red/black point zebra scheme. Two main sweeps are performed at each relaxation step. The first one solves equation 51 every two cells in each index direction (figure 1).The second one solves equation 51 on the remaining cells. At the first iteration of relaxation all the $\Delta\tilde{\nu}$ are zero and then are updated at every sweep.

Table 8 resumes the steps of the solution algorithm.

1	solve the NS equation and find ρ_{new}
2	start with initial solution of $\tilde{\nu}$
3	calculate $ASA_{i,j,k}$,$BSA_{I_{i,j,k}}$,$BSA_{J_{i,j,k}}$,$BSA_{K_{i,j,k}}$,$CSA_{I_{i,j,k}}$,$CSA_{J_{i,j,k}}$,$CSA_{K_{i,j,k}}$
4	calculate $R_{i,j,k}$
5	invert $ASA_{i,j,k}$
6	perform relaxation steps
	a) first sweep on half of the cells (eq 51)
	b) second sweep on the remaining half (eq 51)
7	update $\tilde{\nu}$ (eq 52)

Table 8: Steps for solution of SA equation on any internal cell

3.4.2 Line Zebra Scheme

In the line zebra scheme are separated all the contributions from neighbouring cells but those of the ones standing on the chosen index line. So, if i has been chosen,for instance, the equation 49 becomes:

$$ABCSA_{j,k}\Delta\tilde{\nu}_{j,k} = -R_{j,k} - BSA_{J_{j,k}}\Delta\tilde{\nu}_{j-1,k} - BSA_{K_{j,k}}\Delta\tilde{\nu}_{j,k-1} - CSA_{J_{j,k}}\Delta\tilde{\nu}_{j+1,k} - CSA_{K_{j,k}}\Delta\tilde{\nu}_{j,k+1} \tag{53}$$

where $ABCSA$ is a tridiagonal and encloses, for a given j, k, all the $ASA_{i,j,k}$, $BSA_{I_{i,j,k}}$, $CSA_{I_{i,j,k}}$ with i that goes from 1 to $ni - 1$. Figure 2 shows how $ABCSA$ is assembled.

$R_{j,k}$, $BSA_{J_{j,k}}$, $BSA_{K_{j,k}}$, $CSA_{J_{j,k}}$, $CSA_{K_{j,k}}$ are ni arrays. They are created, for a given j, k, respectively by all the $R_{i,j,k}$, $BSA_{J_{i,j,k}}$,$BSA_{K_{i,j,k}}$,$CSA_{J_{i,j,k}}$,$CSA_{K_{i,j,k}}$ with i that goes from 1 to $ni - 1$.

In order to solve equation 53 firstly the LU decomposition of $ABCSA$ is calculated.

A given number of relaxation steps is performed and the resulting $\Delta\tilde{\nu}$ is used to update the viscous variables following the equation 52.

The relaxation steps follow a line zebra scheme. Two main sweeps are performed at each relaxation step. The first one solves equation 53, with $ABCSA$ factorized, every two line $i = constant$ in both direction j and k (figure 3). The second one solves equation 53, with $ABCSA$ factorized, on any other line $i = constant$.

Table 9 resumes the steps of the solution algorithm.

1	solve the NS equation and find ρ_{new}
3	start with initial solution of $\tilde{\nu}$
4	calculate $ASA_{i,j,k}$,$BSA_{I_{i,j,k}}$,$BSA_{J_{i,j,k}}$,$BSA_{K_{i,j,k}}$,$CSA_{I_{i,j,k}}$,$CSA_{J_{i,j,k}}$,$CSA_{K_{i,j,k}}$
5	calculate $R_{i,j,k}$
6	calculate $ABCSA_{i,j,k}$,$R_{j,k}$,$BSA_{J_{j,k}}$,$BSA_{K_{j,k}}$,$CSA_{J_{j,k}}$,$CSA_{K_{j,k}}$
5	compute the LU factorization of $ABCSA_{i,j,k}$
6	perform relaxation steps
	a) first sweep on half of the cells (eq 53 with $ABCSA$ factorized)
	b) second sweep on the remaining half (eq 53 with $ABCSA$ factorized)
7	update $\tilde{\nu}$ (eq 52)

Table 9: Steps for solution of SA equation on any line $i = constant$ of internal cells

3.5 Code Speedup

The considerations are the same of section 2.6, exchanging Q_v with $\tilde{\nu}$. Moreover ω_{opt} is taken from the previous calculation (2.6).

3.6 Residuals Jacobian Building

In order to build the SA residual jacobian it has been considered only the contribution from the inviscid or convective and the destruction or sink part of residuals. This choice follows the consideration that for stability purposes only the residuals which are negative on the lefthandside of the equation have to be derived.

The residuals jacobian is calculated with respect to the variable $\tilde{\nu}$. Moreover it is useful to consider the part of Jacobian belongin to the cell , on which calculation is performed , and the part that goes to the neighbours. That's why are introduced the arrays ASA, BSA_I, CSA_I, BSA_J, CSA_J, BSA_K, CSA_K.

In the following considerations figure 4 is adopted.

3.6.1 Inviscid Part of Residuals jacobian

The inviscid fluxes appearing in 43 are evaluated by a Van Leer MUSCL method. The convective flux at boundary face jj are:

$$L_{conv_{jj}} = \frac{1}{2}\left(L_{conv_{i,j,k}} + L_{conv_{i-1,j,k}}\right) - \frac{1}{2} \mid V_m \mid (\rho_{i,j,k}\tilde{\nu}_{i,j,k} - \rho_{i-1,j,k}\tilde{\nu}_{i-1,j,k}) \tag{54}$$

where the convective flux at cell (i,j,k), $L_{conv_{i,j,k}}$, is equal to

$$L_{conv_{i,j,k}} = V_{n_{i,j,k}}\rho_{i,j,k}\tilde{\nu}_{i,j,k}$$

and V_m is the average velocity of the two cells sharing the boundary face jj. In order to obtain the convective residuals jacobian equation 54 is derived. The jacobian at boundary face jj is

$$\frac{\partial L_{conv_{jj}}}{\partial \tilde{\nu}} = \frac{1}{2}\left(V_{n_{i,j,k}}\rho_{i,j,k} + V_{n_{i-1,j,k}}\rho_{i-1,j,k}\right) - \frac{1}{2} \mid V_{m_{jj}} \mid (\rho_{i,j,k} - \rho_{i-1,j,k}) \tag{55}$$

Separating the contribution to the right and to the left cell at boundary jj, the first one is

$$\frac{\partial L_{conv_{jj}}}{\partial \tilde{\nu}_i} = \frac{1}{2}V_{n_{i,j,k}}\rho_{i,j,k} - \frac{1}{2} \mid V_{m_{jj}} \mid \rho_{i,j,k} \tag{56}$$

and the second one is

$$\frac{\partial L_{conv_{jj}}}{\partial \tilde{\nu}_{i-1}} = \frac{1}{2}V_{n_{i-1,j,k}}\rho_{i-1,j,k} - \frac{1}{2} \mid V_{m_{jj}} \mid \rho_{i-1,j,k} \tag{57}$$

Writing the two contribution at the boundary face $JJ+1$:

$$\frac{\partial L_{conv_{jj+1}}}{\partial \tilde{\nu}_i} = \frac{1}{2}V_{n_{i,j,k}}\rho_{i,j,k} + \frac{1}{2} \mid V_{m_{jj+1}} \mid \rho_{i,j,k} \tag{58}$$

$$\frac{\partial L_{conv_{jj+1}}}{\partial \tilde{\nu}_{i+1}} = \frac{1}{2}V_{n_{i+1,j,k}}\rho_{i+1,j,k} - \frac{1}{2} \mid V_{m_{jj+1}} \mid \rho_{i+1,j,k} \tag{59}$$

Assuming that a flux is positive when it is directed outward , with respect to the chosen cell, the flux $L_{conv_{jj+1}}$ is positive and $L_{conv_{jj}}$ is negative.

Now recalling that:

$$\begin{aligned}
ASA_{i,j,k} &= \sum_{boundaries} \frac{\partial L_{conv_{boundary}}}{\partial \tilde{\nu}_i} \Delta S_{boundary} \\
BSA_{I_{i,j,k}} &= \sum_{boundaries} \frac{\partial L_{conv_{boundary}}}{\partial \tilde{\nu}_{i-1}} \Delta S_{boundary} \\
CSA_{I_{i,j,k}} &= \sum_{boundaries} \frac{\partial L_{conv_{boundary}}}{\partial \tilde{\nu}_{i+1}} \Delta S_{boundary}
\end{aligned}$$

and adding the contributions from the boundary faces jj and $jj+1$, the arrays $ASA_{i,j,k}$, $BSA_{I_{i,j,k}}$, $CSA_{I_{i,j,k}}$ can be obtained:

$$\begin{aligned}
ASA_{i,j,k} &= ASA_{i,j,k} - \left(V_{n_{i,j,k}} \rho_{i,j,k} - \mid V_{m_{jj}} \mid \rho_{i,j,k}\right) \frac{\Delta S_{jj}}{2} + \left(V_{n_{i,j,k}} \rho_{i,j,k} + \mid V_{m_{jj+1}} \mid \rho_{i,j,k}\right) \frac{\Delta S_{jj+1}}{2} \\
BSA_{I_{i,j,k}} &= BSA_{I_{i,j,k}} - \left(V_{n_{i-1,j,k}} \rho_{i-1,j,k} - \mid V_{m_{jj}} \mid \rho_{i-1,j,k}\right) \frac{\Delta S_{jj}}{2} \\
CSA_{I_{i,j,k}} &= CSA_{I_{i,j,k}} + \left(V_{n_{i+1,j,k}} \rho_{i+1,j,k} - \mid V_{m_{jj+1}} \mid \rho_{i+1,j,k}\right) \frac{\Delta S_{jj+1}}{2}
\end{aligned}$$

Examining the expressions just written , recursive contributions can be deducted. At each boundary face jj four contribution are added:

$$\begin{array}{llll}
V_{n_{i,j,k}} \rho_{i,j,k} \dfrac{\Delta S_{jj}}{2} & is \quad given \quad to & ASA_{i,j,k} & with \quad sign \quad minus \\
 & and \quad to & CSA_{I_{i-1,j,k}} & with \quad sign \quad plus \\
 & & & \\
\mid V_{m_{jj}} \mid \rho_{i,j,k} \dfrac{\Delta S_{jj}}{2} & is \quad given \quad to & ASA_{i,j,k} & with \quad sign \quad plus \\
 & and \quad to & CSA_{I_{i-1,j,k}} & with \quad sign \quad minus \\
 & & & \\
V_{n_{i-1,j,k}} \rho_{i-1,j,k} \dfrac{\Delta S_{jj}}{2} & is \quad given \quad to & BSA_{I_{i,j,k}} & with \quad sign \quad minus \\
 & and \quad to & ASA_{i-1,j,k} & with \quad sign \quad plus \\
 & & & \\
\mid V_{m_{jj}} \mid \rho_{i-1,j,k} \dfrac{\Delta S_{jj}}{2} & is \quad given \quad to & BSA_{I_{i,j,k}} & with \quad sign \quad minus \\
 & and \quad to & ASA_{i-1,j,k} & with \quad sign \quad plus
\end{array} \tag{60}$$

3.6.2 Destruction term of Residuals Jacobian

The destruction term appearing in equation 42 is

$$\rho c_{\omega 1} f_{\omega} \left(\frac{\tilde{\nu}}{d}\right)^2$$

that discretized for cell (i, j, k) becomes

$$DE = \rho_{i,j,k} c_{\omega 1} f_\omega \left(\frac{\tilde{\nu}_{i,j,k}}{d_{i,j,k}} \right)^2 \tag{61}$$

and remembering the LHS of equation 43, that is $\Delta Vol_{i,j,k} \left(PR - DE \right)$, this term goes on the righthandside with sign plus. To understand better this statement, the residual due to source and destruction terms is :

$$R_{i,j,k}(\tilde{\nu}) = -\Delta Vol_{i,j,k} \left(PR - DE \right)$$

and what goes to the RHS is $-R$ so the RHS becomes:

$$-R_{i,j,k}(\tilde{\nu}) = \Delta Vol_{i,j,k} \left(PR - DE \right)$$

Finally the destruction term of residuals jacobian comes from the derivation of equation 61:

$$\frac{\partial \left(\Delta Vol_{i,j,k} DE \right)}{\partial \tilde{\nu}_i} = 2 \rho_{i,j,k} \Delta Vol_{i,j,k} c_{\omega 1} f_\omega \frac{\tilde{\nu}_{i,j,k}}{{d_{i,j,k}}^2} \tag{62}$$

This contribution is then added to $ASA_{i,j,k}$.

3.7 Results

In order to test the SA implicit code the backward facing step case has been chosen. The NS part has been treated without viscous contribution to the jacobian. The comparison is made on convergence between explicit and implicit schemes on a fine and on a one level coarser grid. A result of Fluent convergence is also displayed. The characteristics of the three mesh are sketched in table 10

Case	Grid Type	Number of Elements	First cell center distance to the wall
BFS	Structured	9600	5.E-6
BFS Coarse	Structured	2400	1.E-5
Fluent	Unstructured	9600	5.E-5

Table 10: Mesh details

All Karalis computations have been made with $CFL = 5$, which was the upper reachable CFL without incurring in divergence. Figures 19 and 20 show the convergences for the fine grid while figure 21 shows the Fluent convergences. In fact it is difficult to compare the Fluent convergence with both BFS and BFS coarse convergences because the smaller cell size in Fluent is still considerely larger than those of BFS's grids and that leads to a bigger timestep, i.e. less iterations. The comparison between implicit and explicit schemes can't ignore the different time consumption of the two. Table 11 shows this factor. $Nitrel$ is the number of iterations of ralxation. Adding all these data it can be said that the implicit point red/black is twice slower than explicit and leads to a worst convergence. The line zebra along i is three times slower than explicit but reaches $1.E-5$ in half number of iterations.

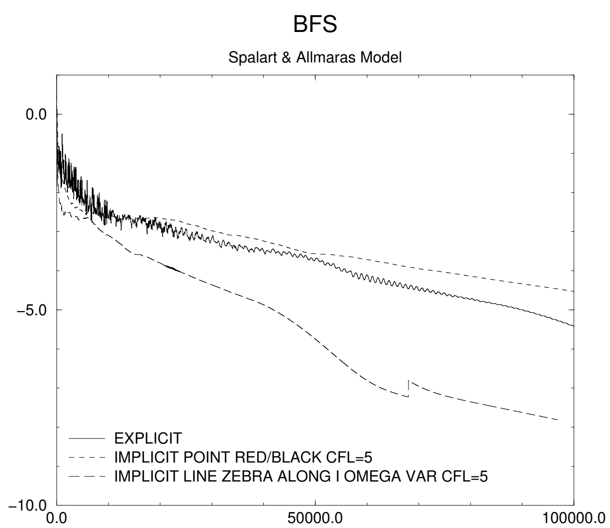

Figure 19: BFS: x-momentum convergence

More informations can be understood by analysing the results on the BFS coarse grid (figures 22, 23 and table 12). The difference between sweep along i and along j is not much marked. What can be said is that a good convergence is obtained yet for a number of relaxation iterations of 10. The line zebra along i is with $nitrel = 10$ is 1.75 times slower than explicit but reaches

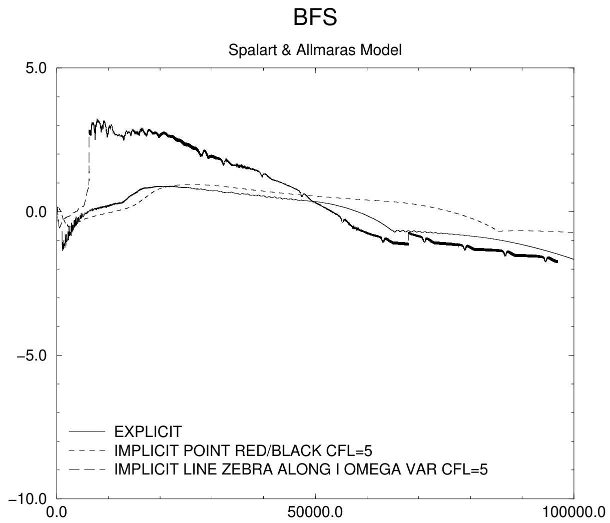

Figure 20: BFS: SA-equation convergence

Scheme	Nitrel	Time (s)
Explicit	-	582000
Implicit red/black	50	953000
Implicit line zebra along I	50	1735093

Table 11: BFS: Different schemes time consumption for 90000 iterations

$1.E-5$ in three times less iterations. The result is that the implicit scheme with $nitrel = 10$ is about 1.7 times faster than explicit, on the coarse grid. What is expected is that for coarser grids this gap widens out. A test have been conducted with the viscous contributions added to the jacobian but that yelds a slight worse convergence.

Scheme	Nitrel	Time (s)
Explicit	-	136000
Implicit red/black	50	252000
Implicit line zebra along I	50	465000
Implicit line zebra along I	10	240000
Implicit line zebra along J	50	485000

Table 12: BFS coarse : Different schemes time consumption for 99000 iterations

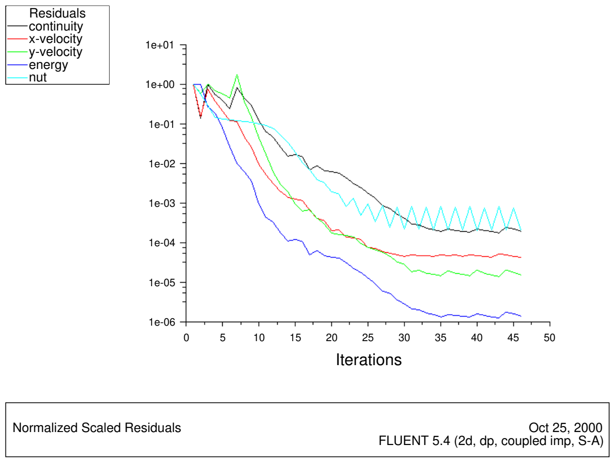

Figure 21: Fluent convergences

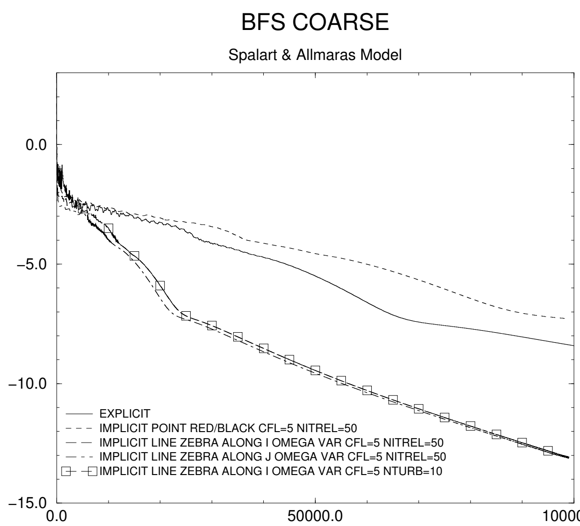

Figure 22: BFS coarse: x-momentum convergence

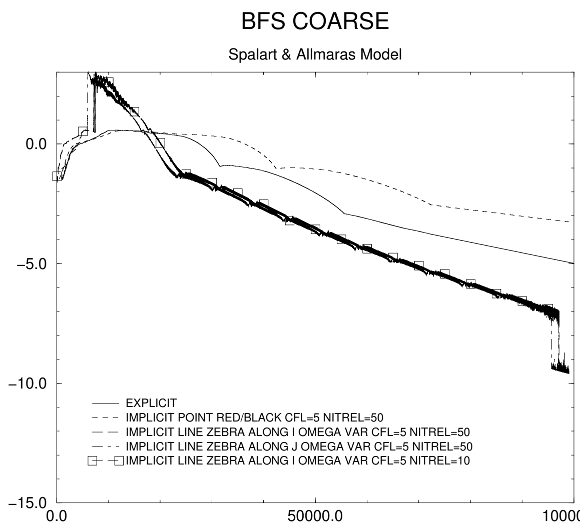

Figure 23: BFS coarse: SA-equation convergence

4 Brief summary of subroutines

4.1 Modified routines

The routines modified with respect to the release of Karalis4 in $/u/cfdc/CFDC\_dir/karalis4.0$ are:

`alloc.f`	`conflx5.f`	`convsa.f`	`readbc.f`	`solver.f`	`tvdflux.f`	`visresk.f`
`calcsousa.f`	`conflxmg.f`	`grad5.f`	`readdat.f`	`solvermg.f`	`visresi.f`	
`chkpnt.f`	`conflxsa.f`	`main.f`	`smoothjm.f`	`sourcesa.f`	`visresj.f`	

Follows the description of the added part of each subroutine.

alloc.f

It allocates all the matrices for implicit.

- point red/black scheme
  Three pointers are allocated:

  $IMATRX(IGRID, IBLOC)$ pointer to $7 \times 5 \times 5 \times NCELL(IGRID, IBLOC)$ locations occupied by matrices A,B_I,B_J,B_K,C_I,C_J,C_K ;

  $IRHSIDE(IGRID, IBLOC)$ pointer to $5 \times NCELL(IGRID, IBLOC)$ locations occupied by vector RHS ;

  $IDELTAQ(IGRID, IBLOC)$ pointer to $5 \times NCELL(IGRID, IBLOC)$ locations occupied by vector $DELTAQ$;

  For a Spalart & Allmaras calculation three more pointers are needed:

  $IMATSA(IGRID, IBLOC)$ pointer to $7 \times NCELL(IGRID, IBLOC)$ locations occupied by matrices ASA,BSA_I,BSA_J,BSA_K,CSA_I,CSA_J,CSA_K ;

  $IRHSSA(IGRID, IBLOC)$ pointer to $NCELL(IGRID, IBLOC)$ locations occupied by vector RHS ;

  $IDELTASA(IGRID, IBLOC)$ pointer to $NCELL(IGRID, IBLOC)$ locations occupied by vector $IDQSA$.

- line zebra scheme
  $NCMAX$ maximum number of cells, minus ghost cells, is found. This is useful for future allocation of integer vectors $IPIV$ and $IPIVSA$. Four pointers are here allocated:

  $IMATRX(IGRID, IBLOC)$ pointer to $7 \times 5 \times 5 \times NCELL(IGRID, IBLOC)$ locations occupied by matrices A,B_I,B_J,B_K,C_I,C_J,C_K ;

  $IRHSIDE(IGRID, IBLOC)$ pointer to $5 \times NCELL(IGRID, IBLOC)$ locations occupied by vector RHS ;

  $IDELTAQ(IGRID, IBLOC)$ pointer to $5 \times NCELL(IGRID, IBLOC)$ locations occupied by vector $DELTAQ$;

$IZEBRA(IGRID, IBLOC)$ pointer to $28 \times 5 \times NCELL(IGRID, IBLOC)$ locations occupied by matrice $ZEBRA$.

For a Spalart & Allmaras calculation four more pointers are needed:

$IMATSA(IGRID, IBLOC)$ pointer to $7 \times NCELL(IGRID, IBLOC)$ locations occupied by matrices ASA,BSA_I,BSA_J,BSA_K,CSA_I,CSA_J,CSA_K ;

$IRHSSA(IGRID, IBLOC)$ pointer to $NCELL(IGRID, IBLOC)$ locations occupied by vector RHS ;

$IDELTASA(IGRID, IBLOC)$ pointer to $NCELL(IGRID, IBLOC)$ locations occupied by vector $IDQSA$;

$IZEBRASA(IGRID, IBLOC)$ pointer to $4 \times NCELL(IGRID, IBLOC)$ locations occupied by matrice $ZEBRASA$.

calcsousa.f

It calculates the source contribution to the jacobian of residuals ASA for the Spalart & Allmaras equation.

chkpnt.f

It checks the *.dat* file and prevents the use of implicit with multigrid ,preconditioning or central differencing flux.

conflx5.f

For implicit it calls the routine $tvdflux$ with pointers to matrices A,B_I,B_J,B_K,C_I,C_J,C_K.

conflxmg.f

For multigrid implicit it calls the routine $tvdflux$ with pointers to matrices A,B_I,B_J,B_K,C_I,C_J,C_K.

conflxsa.f

For implicit it calls the routine *convsa* with pointers to matrices ASA,BSA_I,BSA_J,BSA_K,CSA_I,CSA_J,CSA_K.

convsa.f

For implicit Spalart & Allmaras it calculates the jacobian of inviscid residuals and adds it to the matrices ASA,BSA_I,BSA_J,BSA_K,CSA_I,CSA_J,CSA_K.

grad5.f

For implicit it calls the routines $visresi$,$visresj$ and $visresk$ with pointers to matrices A,B_I,B_J,B_K,C_I,C_J,C_K.

main.f

It calls the routine *ctstep* also in case of point red black and line zebra constant time step.

readbc.f

It keeps $NDIM = 3$ also for bidimensional cases.

readdat.f

It adds three new input lines in files *.dat*:

line 22 It asks for the number of iterations of relaxation for each level of grid

line 23 It asks for the directions on which performing line zebra sweep through a single value from 1 to 7 ($1 = i$, $2 = j$, $3 = k$, $4 = ij$, $5 = jk$, $6 = ik$, $7 = ijk$)

line 24 It asks for the values of ω_{SLOR},ω_1,ω_2,ω_3. If $\omega_{SLOR} = 1$ no $SLOR$ is applied. If $\omega_1 = 0$ a fixed $\omega = 0.15$ is taken. If ω_1 is negative the absolute value of ω_1 is taken as ω fixed.

smoothjm.f

It avoids the cycle on k if $nk = 2$.

solver.f

For implicit line zebra it allocates dynamically the vector $IPIV$ and for Spalart & Allmaras implicit line zebra also the vector $IPIVSA$. It calls the routine *koala* for implicit point red/black and *sloth* for implicit line zebra.

solvermg.f

It calls the routines $koalamgf$ and $koalamgc$ for implicit point multigrid.

sourcesa.f

It calls the routine *calcsousa* with pointer to ASA.

tvdflux.f

It calculates the inviscid residual contribution to the matrices A,B_I,B_J,B_K,C_I,C_J,C_K.

visresi.f,visresj.f,visresk.f

For implicit viscous calculation they call $visjack$ with pointers to A,B_I,B_J,B_K,C_I,C_J,C_K.

4.2 New routines

The routines not present in the release of Karalis4 in $/u/cfdc/CFDC\_dir/karalis4.0$ are:

`absa.f`	`invert.f`	`koamatprod6.f`	`setsweep.f`	`slothrelaxsa.f`
`absap.f`	`invertsa.f`	`koamatsa.f`	`setzeroi.f`	`slothupdateimp.f`
`comprms.f`	`koala.f`	`koarelax.f`	`sloth.f`	`slothupdatesa.f`
`euljack.f`	`koalamgc.f`	`koarelaxsa.f`	`slothfactorize.f`	`visjack.f`
`evoptomg.f`	`koalamgf.f`	`koaupdateimp.f`	`slothfactorizesa.f`	
`glob_con_2.f`	`koamatmilan.f`	`koaupdatesa.f`	`slothmatprod.f`	
`invers.f`	`koamatprod4.f`	`setmatomega.f`	`slothmatprodsa.f`	
`inversomega.f`	`koamatprod4sa.f`	`setres2.f`	`slothrelax.f`	

Follows a very brief description of each subroutine.

absa.f

It computes the third contributions of 32:

$$| X_{jj} | \frac{\Delta S_{jj}}{2}$$

That's divided in three loops.

- the first one calculates a matrix called here R:

$$R = \frac{\Delta S_{jj}}{2} * MR_v \mid \Lambda \mid$$

- the second one calculates a matrix called here L:

$$L = L_v$$

- the third one calculates a matrix called here A:

$$A = RL$$

This routines works for both the point red/black and the line zebra schemes.

absap.f

Likewise *absa.f* it computes the third contributions of 32 for preconditioned system:

$$| X_{jj} | \frac{\Delta S_{jj}}{2}$$

That's divided in three loops.

- the first one calculates a matrix called here R:

$$R = \frac{\Delta S_{jj}}{2} * M_m R_{P_v} \mid \Lambda \mid$$

- the second one calculates a matrix called here L:

$$L = L_{P_v}$$

- the third one calculates a matrix called here A:

$$A = RL$$

This routines works for both the point red/black and the line zebra schemes.

comprms.f

It calculates the maximum and RMS of the residual chosen by user. This routine is called by the line zebra scheme.

euljack.f

It computes, in three sweeps over (i, j, k), the matrices AI and AM which correspond respectively to the first and second contribution of 32, that for a i face are:

$$J_{F_{i,j,k}} \frac{\Delta S_{jj}}{2}$$
$$J_{F_{i-1,j,k}} \frac{\Delta S_{jj}}{2}$$

Finally AI and AM are added to A,B_I,B_J,B_K,C_I,C_J,C_K following indications of 32. This routines works for both the point red/black and the line zebra schemes.

evoptomg.f

It evaluates ω_{opt} according what is written in section 2.6. Finally ω_{opt} is compared with ω_3 and the minimum is taken and with 0.15 and the maximum is taken. This routine is called by the line zebra scheme.

glob_con_2.f

It tries to emulate *glob_con.f*. It should find the maximum and RMS of the residual chosen by user between blocks assigned to different processors. This routine is called by the line zebra scheme.

invers.f

Invert, by Gauss-Jordan elimination method, a 5×5 matrix A given in input and the result is stored in $HELP$ matrix. It's used in point red/black scheme to invert the matrix $A(i, j, k)$. This routine is called by the point red/black scheme.

inversomega.f

Invert, by Gauss-Jordan elimination method, a 3×3 matrix A given in input and the result is stored in $HELP$ matrix. It's used in line zebra scheme to invert only once the ω's matrix (section 2.6). This routine is called by the line zebra scheme.

invert.f

It calls $invers.f$ for each i, j, k cell and then put the contents of $HELP$ in A. This routine is called by the point red/black scheme.

invertsa.f

It inverts scalar quantity ASA and store it in itself. This routine is called by the point red/black scheme.

koala.f

It's the point red/black solver subroutine. Resuming the main subroutine instructions:

- it saves dependent variables in array old
- it computes timestep ($tstep$)
- it sets boundary conditions and calculates boundary gradients ($setbc$,$setgrad$)
- it adds $M^{-}1\frac{\Delta Vol}{\Delta t}$ to $A(i,j,k)$ and sets to zero B_I,B_J,B_K,C_I,C_J,C_K ($koamatmilan$)
- it sets to zero ASA,BSA_I,BSA_J,BSA_K,CSA_I,CSA_J,CSA_K ($setzero$)
- it calculates residuals and matrices A,B_I,B_J,B_K,C_I,C_J,C_K ($setres2$)
- it inverts matrix $A(i,j,k)$ and store it in itself ($invert$)
- it sets to zero ΔQ_v ($setzero$)
- point red/black relaxation loop ($koarelax$)
- it updates dependent variables ($koaupdateimp$)
- it adds $\rho\frac{\Delta Vol}{\Delta t}$ to $ASA(i,j,k)$ ($koamatsa$)
- it inverts $ASA(i,j,k)$ and store it in itself ($invertsa$)
- it sets to zero $\Delta\tilde{\nu}$ ($setzero$)
- point red/black relaxation loop ($koarelaxsa$)
- it updates $\tilde{\nu}$ ($koaupdatesa$)

koalamgc.f

Old version of $koala$ for multigrid coarse levels.

koalamgf.f

Old version of $koala$ for multigrid fine levels.

koamatmilan.f

It adds $M^{-}1\frac{\Delta Vol}{\Delta t}$ to $A(i,j,k)$ or $M_m{}^{-}1\frac{\Delta Vol}{\Delta t}$ in case of preconditioning. It also sets to zero the matrices B_I,B_J,B_K,C_I,C_J,C_K. This routine is called by the point red/black scheme.

koamatprod4.f

It calculates the righthandside RHS of equation 17 that, for any (i,j,k) given, is:

$$
\begin{aligned}
RHS_{i,j,k} = -R_{i,j,k} - B_{I_{i,j,k}}\Delta Q_{v_{i-1,j,k}} - B_{J_{i,j,k}}\Delta Q_{v_{i,j-1,k}} - B_{K_{i,j,k}}\Delta Q_{v_{i,j,k-1}} + \\
-C_{I_{i,j,k}}\Delta Q_{v_{i+1,j,k}} - C_{J_{i,j,k}}\Delta Q_{v_{i,j+1,k}} - C_{K_{i,j,k}}\Delta Q_{v_{i,j,k+1}}
\end{aligned}
$$

This routine is called by the point red/black scheme.

koamatprod4sa.f

It calculates the righthandside $RHSSA$ of equation 51 that, for any (i, j, k) given, is:

$$RHSSA_{i,j,k} = -R_{i,j,k} - BSA_{I_{i,j,k}}\Delta\tilde{\nu}_{i-1,j,k} - BSA_{J_{i,j,k}}\Delta\tilde{\nu}_{i,j-1,k} - BSA_{K_{i,j,k}}\Delta\tilde{\nu}_{i,j,k-1} + -CSA_{I_{i,j,k}}\Delta\tilde{\nu}_{i+1,j,k} - CSA_{J_{i,j,k}}\Delta\tilde{\nu}_{i,j+1,k} - CSA_{K_{i,j,k}}\Delta\tilde{\nu}_{i,j,k+1}$$

This routine is called by the point red/black scheme.

koamatprod6.f

It multiplies the matrix ${A_{i,j,k}}^{-1}$ by $RHS_{i,j,k}$. Here ${A_{i,j,k}}^{-1}$ is denoted by H and $RHS_{i,j,k}$ by R. This routine is called by the point red/black scheme.

koamatsa.f

It adds $\rho\frac{\Delta Vol}{\Delta t}$ to $ASA(i, j, k)$. This routine is called by the point red/black scheme.

koarelax.f

It performs a step of point red/black relaxation. The first four couple of loops solve on the first half of the cells following the figure 1 . The latter four couple of loops solve on the remaining cells. This routine is called by the point red/black scheme.

koarelaxsa.f

It performs a step of point red/black relaxation on Spalart & Allmaras equation. The first four couple of loops solve on the first half of the cells following the figure 1 . The latter four couple of loops solve on the remaining cells. This routine is called by the point red/black scheme.

koaupdateimp.f

It updates primitives variables by mean of ΔQ_v. ρ is then deduced. This routine is called by the point red/black scheme.

koaupdatesa.f

It updates $\tilde{\nu}$ by mean of $\Delta\tilde{\nu}$ according to equation 52 with ω instead of 0.15, its value depending on the user choice. This routine is called by the point red/black scheme.

setmatomega.f

It inverts ω's matrix and store it in itself. This routine is called by the line zebra scheme.

setres2.f

Similar to *setres* it does'nt copy the dependent variables in old array nor call preconditioning routine *vprec*. This routines works for both the point red/black and the line zebra schemes.

setsweep.f

It sets the indices $N1$,$N2$,$N3$,$I1$,$I2$,$I3$ according to the chosen sweep index. This routine is called by the line zebra scheme. Those indices are used in order to avoid multiple if statements.

setzeroi.f

It sets to zero an array of integers. Equivalent of *setzero* for integers. This routine is called by the line zebra scheme.

<u>sloth.f</u>

It's the line zebra solver subroutine. Resuming the main subroutine instructions:

- it saves dependent variables in array old
- it computes timestep ($tstep$)
- it sets boundary conditions and calculates boundary gradients ($setbc$,$setgrad$)
- it adds $M^{-}1\frac{\Delta Vol}{\Delta t}$ to $A(i,j,k)$ and sets to zero B_I,B_J,B_K,C_I,C_J,C_K ($koamatmilan$)
- it sets to zero ASA,BSA_I,BSA_J,BSA_K,CSA_I,CSA_J,CSA_K ($setzero$)
- it calculates residuals and matrices A,B_I,B_J,B_K,C_I,C_J,C_K ($setres2$)
- it sets to zero vector ΔQ_v ($setzero$)
- it sets to zero matrix $ZEBRA$ which corresponds to matrix $ABC_{j,k}$ for a i line sweep ($setzero$)
- it sets to zero vector of integers $IPIV$ ($setzeroi$)
- it prepares the indices $N1$,$N2$,$N3$,$I1$,$I2$,$I3$ ($setsweep$)
- it assembles and factorizes the matrix $ZEBRA$ ($slothfactorize$)
- line zebra relaxation loop ($slothrelax$)
- for three omegas it updates dependent variables ($slothupdateimp$)
- it sets boundary conditions and calculates boundary gradients ($setbc$,$setgrad$)
- it calculates residuals ($setres2$)
- it calculates the maximum and RMS of the chosen residual ($comprms$)
- if one of them is not valid cut ω in two and repeat from updating
- else invert omega matrix and store it in itself ($setmatomega$)
- it calculates the optimum ω
- it performs an update with $\Delta Q_v = 0$ ($slothupdateimp$)
- it sets boundary conditions and calculates boundary gradients ($setbc$,$setgrad$)
- it performs an update with ΔQ_v and ω_{opt} ($slothupdateimp$)
- it adds $\rho\frac{\Delta Vol}{\Delta t}$ to $ASA(i,j,k)$ ($koamatsa$)
- it sets to zero vector $\Delta\tilde{\nu}$ ($setzero$)
- it sets to zero matrix $ZEBRASA$ which corresponds to matrix $ABCSA_{j,k}$ for a i line sweep ($setzero$)
- it sets to zero vector of integers $IPIVSA$ ($setzeroi$)
- it prepares the indices $N1$,$N2$,$N3$,$I1$,$I2$,$I3$ ($setsweep$)
- it assembles and factorizes the matrix $ZEBRASA$ ($slothfactorizesa$)
- line zebra relaxation loop ($slothrelaxsa$)
- it performs an update with $\Delta\tilde{\nu}$ and ω_{opt} ($slothupdatesa$)

slothfactorize.f

It assembles the matrix $ZEBRA$ and then factorizes it by mean of routine $dgbtrf$, part of $LAPACK$ package. This routine is called by the line zebra scheme.

slothfactorizesa.f

It assembles the matrix $ZEBRASA$ and then factorizes it by mean of routine $dgttrf$, part of $LAPACK$ package. This routine is called by the line zebra scheme.

slothmatprod.f

It calculates the RHS which appears in equation 16. In fact for a sweep along i:

$$\begin{aligned} RHS = -R_{i,j,k} - B_{I_{i,j,k}} \Delta Q_{v_{i-1,j,k}} - B_{J_{i,j,k}} \Delta Q_{v_{i,j-1,k}} - B_{K_{i,j,k}} \Delta Q_{v_{i,j,k-1}} + \\ -C_{I_{i,j,k}} \Delta Q_{v_{i+1,j,k}} - C_{J_{i,j,k}} \Delta Q_{v_{i,j+1,k}} - C_{K_{i,j,k}} \Delta Q_{v_{i,j,k+1}} \end{aligned}$$

This routine is called by the line zebra scheme.

slothmatprodsa.f

It calculates the RHS which appears in equation 53. In fact for a sweep along i:

$$RHSSA = -R_{j,k} - BSA_{J_{j,k}} \Delta\tilde{\nu}_{j-1,k} - BSA_{K_{j,k}} \Delta\tilde{\nu}_{j,k-1} - CSA_{J_{j,k}} \Delta\tilde{\nu}_{j+1,k} - CSA_{K_{j,k}} \Delta\tilde{\nu}_{j,k+1}$$

This routine is called by the line zebra scheme.

slothrelax.f

It performs a step of line zebra relaxation. The first two loops solve on the first half of the cells following the figure 3 . The latter two loops solve on the remaining cells. This routine is called by the line zebra scheme. It uses the routine $dgbtrs$ from $LAPACK$ package.

slothrelaxsa.f

It performs a step of line zebra relaxation on the Spalart & Allmaras equation. The first two loops solve on the first half of the cells following the figure 3 . The latter two loops solve on the remaining cells. This routine is called by the line zebra scheme. It uses the routine $dgttrs$ from $LAPACK$ package.

slothupdateimp.f

It updates primitives variables by mean of ΔQ_v. ρ is then deduced. This routine is called by the line zebra scheme.

slothupdatesa.f

It updates $\tilde{\nu}$ by mean of $\Delta\tilde{\nu}$ according to equation 52 with ω instead of 0.15, its value depending on the user choice. This routine is called by the line zebra scheme.

visjack.f

It computes the viscous addition to A,B_I,B_J,B_K,C_I,C_J,C_K following what is written in 5.2. This routines works for both the point red/black and the line zebra schemes.

5 Appendix

5.1 Transformation Matrices

From viscous primitives to conservatives variables:

$$M = \left(\frac{\partial Q}{\partial Q_v}\right) = \begin{bmatrix} \rho_P & 0 & 0 & 0 & \rho_T \\ u\rho_P & \rho & 0 & 0 & u\rho_T \\ v\rho_P & 0 & \rho & 0 & v\rho_T \\ w\rho_P & 0 & 0 & \rho & w\rho_T \\ H\rho_P - H_P & \rho u & \rho v & \rho w & H\rho_T + \rho h_T \end{bmatrix}$$

From conservatives to viscous primitives variables:

$$M^{-1} = \left(\frac{\partial Q_v}{\partial Q}\right) = \begin{bmatrix} \frac{\rho_T}{d}\left(h - \frac{V^2}{2}\right) + \Delta^2 & \frac{\rho_T}{d}u & \frac{\rho_T}{d}v & \frac{\rho_T}{d}w & -\frac{\rho_T}{d} \\ -\frac{u}{\rho} & \frac{1}{\rho} & 0 & 0 & 0 \\ -\frac{v}{\rho} & 0 & \frac{1}{\rho} & 0 & 0 \\ -\frac{w}{\rho} & 0 & 0 & \frac{1}{\rho} & 0 \\ -\frac{\rho_P}{d}\left(h - \frac{V^2}{2}\right) + \frac{H_P}{d} & -\frac{\rho_P}{d}u & -\frac{\rho_P}{d}v & -\frac{\rho_P}{d}w & \frac{\rho_P}{d} \end{bmatrix}$$

where :

$$\begin{aligned} H_P &= 1 - \rho h_P \\ d &= \rho\rho_P h_T + \rho_T H_P \\ \Delta &= \sqrt{\frac{\rho h_T}{d}} \end{aligned}$$

Preconditioned transformation matrix from viscous primitives to conservatives variables:

$$M_m = \left(\frac{\partial Q}{\partial Q_v}\right)_P = \begin{bmatrix} M_P & 0 & 0 & 0 & M_T \\ uM_P & \rho & 0 & 0 & uM_T \\ vM_P & 0 & \rho & 0 & vM_T \\ wM_P & 0 & 0 & \rho & wM_T \\ HM_P - H_P & \rho u & \rho v & \rho w & HM_T + \rho h_T \end{bmatrix}$$

Preconditioned transformation matrix from conservatives to viscous primitives variables:

$$M^{-1} = \left(\frac{\partial Q_v}{\partial Q}\right)_P = \begin{bmatrix} \frac{M_T}{d^*}\left(h - \frac{V^2}{2}\right) + \Delta^{*2} & \frac{M_T}{d^*}u & \frac{M_T}{d^*}v & \frac{M_T}{d^*}w & -\frac{M_T}{d^*} \\ -\frac{u}{\rho} & \frac{1}{\rho} & 0 & 0 & 0 \\ -\frac{v}{\rho} & 0 & \frac{1}{\rho} & 0 & 0 \\ -\frac{w}{\rho} & 0 & 0 & \frac{1}{\rho} & 0 \\ -\frac{M_P}{d^*}\left(h - \frac{V^2}{2}\right) + \frac{H_P}{d^*} & -\frac{M_P}{d^*}u & -\frac{M_P}{d^*}v & -\frac{M_P}{d^*}w & \frac{M_P}{d^*} \end{bmatrix}$$

where :

$$\begin{aligned} d^* &= \rho M_P h_T + M_T H_P \\ \Delta^* &= \sqrt{\frac{\rho h_T}{d^*}} \end{aligned}$$

and M_P,M_T are the correction given by Merkle.

5.2 Viscous Fluxes

In section 2.7.2 have been introduced the $R_{\alpha\beta}$, which multiplied by the viscous variable gradient and the face surface give the residuals at a given boundary face jj. For the continuity equation they are all zero. For the x-momentum equation:

$$\begin{aligned}
R_{xx} &= \frac{4}{3}\mu n_x \\
R_{xy} &= R_{yx} = \mu n_y \\
R_{xz} &= R_{zx} = \mu n_z \\
R_{yy} &= -\frac{2}{3}\mu n_x \\
R_{yz} &= R_{zy} = 0 \\
R_{zz} &= -\frac{2}{3}\mu n_x
\end{aligned}$$

Likewise for the y-momentum equation:

$$\begin{aligned}
R_{xx} &= -\frac{2}{3}\mu n_y \\
R_{xy} &= R_{yx} = \mu n_x \\
R_{xz} &= R_{zx} = 0 \\
R_{yy} &= \frac{4}{3}\mu n_y \\
R_{yz} &= R_{zy} = \mu n_z \\
R_{zz} &= -\frac{2}{3}\mu n_y
\end{aligned}$$

and for the z-momentum equation:

$$\begin{aligned}
R_{xx} &= -\frac{2}{3}\mu n_z \\
R_{xy} &= R_{yx} = 0 \\
R_{xz} &= R_{zx} = \mu n_x \\
R_{yy} &= -\frac{2}{3}\mu n_z \\
R_{yz} &= R_{zy} = \mu n_y \\
R_{zz} &= \frac{4}{3}\mu n_z
\end{aligned}$$

The energy equation has these $R_{alpha\beta}$:

$$\begin{aligned}
R_{xx} &= \frac{2}{3}\mu\left(2un_x - vn_y - wn_z\right) \\
R_{xy} &= R_{yx} = \mu\left(vn_x + un_y\right) \\
R_{xz} &= R_{zx} = \mu\left(wn_x + un_z\right)
\end{aligned}$$

$$\begin{aligned}
R_{yy} &= \frac{2}{3}\mu\left(2vn_y - un_x - wn_z\right)\\
R_{yz} &= R_{zy} = \mu\left(wn_y + vn_z\right)\\
R_{zz} &= \frac{2}{3}\mu\left(2wn_z - un_x - vn_y\right)\\
R_{tx} &= -\kappa n_x\\
R_{ty} &= -\kappa n_y\\
R_{tz} &= -\kappa n_z
\end{aligned}$$

5.3 Flat plate boundary Layer: Relation between η,ϕ,V^+,y^+

In the laminar boundary layer for a flat plate rules:

$$\phi = \frac{V}{V_e} = \frac{y}{\delta} = \eta \tag{63}$$

being y the distance from the wall.

In terms of friction velocity:

$$\frac{V}{V_e} = \frac{V}{V_e V_\tau} = \frac{y}{\delta V_\tau}$$

or

$$\frac{V}{V_\tau} = \frac{y V_e}{\delta V_\tau}$$

but for a flat plate in laminar flow:

$$\tau_p = \frac{\mu V_e}{\delta}$$

Some algebra:

$$V^+ = \frac{V}{V_\tau} = \frac{y}{\mu V_\tau}\frac{muV_e}{\delta}$$

remembering $\tau_p = \frac{muV_e}{\delta}$:

$$V^+ = \frac{y\tau_p}{\mu V_\tau} = \frac{y\rho}{\mu V_\tau}\frac{\tau_p}{\rho}$$

and from $V_\tau = \frac{\tau_p}{\rho}$ and $\nu = \frac{\mu}{\rho}$:

$$V^+ = \frac{y\rho}{\mu V_\tau}{V_\tau}^2 = y\frac{V_\tau}{\nu}$$

and finally from the definition of l_τ, $l_\tau = \frac{\nu}{V_\tau}$, that leads to:

$$V^+ = y^+ \tag{64}$$

So for a laminar flow on a flat plate:

$$y^+ = \eta\frac{V_\tau}{V_e} = \eta\frac{\phi}{V^+}$$