

Evaluating the LCD algorithm for solving linear systems of equations arising from implicit SUPG formulation of compressible flows

Lucia Catabriga¹, Alvaro L.G.A. Coutinho^{2,*} and Leopoldo P. Franca³

¹ Department of Computer Science, Federal University of Espírito Santo, Brazil

² Center for Parallel Computations and Dept. of Civil Engineering, Federal University of Rio de Janeiro, Brazil

³ Department of Mathematics, University of Colorado at Denver, USA

SUMMARY

In this work we evaluate the performance of the Left Conjugate Direction method recently introduced by Yuan, Golub, Plemmons and Cicilio for the solution of nonsymmetric systems of linear equations arising from the implicit semi-discrete SUPG finite element formulation of advective-diffusive and inviscid compressible flows. We extend the original algorithm to accommodate restarts and typical element-by-element preconditioners. We also show how to select the first left conjugate vector to start LCD. Several problems are solved, accessing performance parameters such as number of iterations, memory requirements and CPU times, and results are compared with other algorithms, such as GMRES, TFQMR and Bi-CGSTAB. Copyright © 2003 John Wiley & Sons, Ltd.

KEY WORDS: Stabilized finite elements; Left Conjugate Direction method; Krylov space methods; compressible flows

1. INTRODUCTION

Implicit finite element strategies for compressible flow problems in science and engineering often requires repeated solution of nonlinear systems of equations involving millions of unknowns. After some form of linearization, these systems are usually solved by generalizations of the conjugate gradient method, GMRES or QMR and its variants [1]. The success of this solution strategy requires an efficient implementation of matrix-vector products and the choice of a suitable preconditioner. Within the finite element method, implementations of global matrix-vector products are performed by element level products followed by global assembly, which forms the core of element- by-element strategies as introduced in the finite element simulation of compressible flow by Shakib, Hughes and Johan [2]. Several different parallel element-by-

*Correspondence to: COPPE/Federal University of Rio de Janeiro, P.O. Box 68506, RJ 21945-970, Rio de Janeiro, Brazil, e-mail: alvaro@nacad.ufrj.br

Received 10 March 2003

element preconditioners, using the same data structures of the matrix-vector product, have been proposed earlier [2, 3].

Recently Yuan, Golub, Plemmons and Cecilio [4] introduced a new algorithm for solving nonsymmetric, nonsingular linear systems, the Left Conjugate Direction (LCD for short) method. This method is based on the concept of left and right conjugate vectors for nonsymmetric and nonsingular matrices and possesses several theoretical advantages: (i) it has a finite termination property; (ii) breakdown for general matrices can be avoided and (iii) there is a connection between LCD and LU decomposition. Initial experiments in [4] using a MATLAB implementation have shown that LCD has attractive convergence rates when compared to Bi-CGSTAB, QMR and GMRES algorithms. That motivated us to investigate the performance of LCD. Therefore, in this work we evaluate the performance of LCD in the solution of nonsymmetric systems of linear equations arising from the implicit semi-discrete SUPG finite element formulation for advection-diffusion and inviscid compressible flows described in [5, 6]. We extend the original algorithm to accommodate restarts and typical finite element preconditioners very much in the same manner Shakib, Hughes and Johan [2] did for GMRES. We also study how to select the first left conjugate vector to start LCD.

The remainder of this work is organized as follows. In the next section we review the stabilized finite element formulations for advection-diffusion and the two-dimensional Euler equations. In section 3 we describe LCD algorithm, with particular emphasis on element-by-element matrix-vector products and preconditioners, the introduction of restarts and the options to select the first left conjugate vector. Section 4 shows several numerical experiments, where we compare the performance of LCD with GMRES, Bi-CGSTAB and TFQMR in advective-diffusive and compressible flow problems. Finally the paper ends with a summary of our main conclusions.

2. Governing equations and finite element formulation

2.1. Advection-diffusion equation

Let us consider the following time-dependent advection-diffusion equation in conservative form defined in a domain Ω with boundary Γ :

$$u_{,t} + \sigma_{i,i}^a + \sigma_{i,i}^d = f \quad (1)$$

where u represents the quantity being transported (e.g. temperature, concentration), $\sigma_i^a = \beta_i u$ is the advective flux, β_i is the flow with $\nabla \cdot \beta = 0$, $\sigma_i^d = -k_{ij} u_{,i}$ is the diffusive flux and k_{ij} is the volumetric diffusivity given as,

$$\mathbf{k} = \begin{bmatrix} k_x & 0 \\ 0 & k_y \end{bmatrix} \quad (2)$$

Substituting the divergence free condition, equation (1) can be rewritten as

$$\frac{\partial u}{\partial t} + \beta \cdot \nabla u - \nabla \cdot (\mathbf{k} \nabla u) = f \quad (3)$$

The essential and natural boundary conditions appended to equation (3) are:

$$u = g \quad \text{on } \Gamma_g \quad (4)$$

$$\mathbf{n} \cdot \mathbf{k} \nabla u = h \quad \text{on } \Gamma_h \quad (5)$$

where g and h are given functions of $\mathbf{x} = (x, y)$ and t , \mathbf{n} is the unit outward normal vector at the boundary, Γ_g and Γ_h are the complementary subsets of Γ where boundary conditions are prescribed. Initial conditions are given by,

$$u(\mathbf{x}, 0) = u_o(\mathbf{x}) \quad \text{on } \Omega \quad (6)$$

Consider a finite element discretization of Ω into elements Ω_e , $e = 1, \dots, n_{el}$, where n_{el} is the number of elements. We consider piecewise linear basis spanning finite-dimensional trial solution and test function spaces \mathcal{S}^h and \mathcal{V}^h . The stabilized finite element formulation of equation (3) can then be written as follows. Find $u^h \in \mathcal{S}^h$ such that $\forall w^h \in \mathcal{V}^h$:

$$\begin{aligned} & \int_{\Omega} \left(w^h \frac{\partial u^h}{\partial t} + w^h \beta^h \cdot \nabla u^h - \nabla w^h \cdot \mathbf{k} \nabla u^h \right) d\Omega + \\ & \sum_{e=1}^{n_{el}} \int_{\Omega^e} \tau_{SUPG} \beta^h \cdot \nabla w^h \left(\frac{\partial u^h}{\partial t} + \beta^h \cdot \nabla u^h \right) d\Omega = \\ & \int_{\Omega} w^h f d\Omega + \sum_{e=1}^{n_{el}} \int_{\Omega^e} \tau_{SUPG} \beta^h \cdot \nabla w^h f d\Omega \end{aligned} \quad (7)$$

where τ_{SUPG} is the *SUPG* stabilization parameter which may be computed as suggested in [7] and [8]. Let the standard finite element approximation be given as follows:

$$u^h(\mathbf{x}) \cong \sum_{i=1}^{n_{nodes}} N_i(\mathbf{x}) u_i \quad (8)$$

where n_{nodes} is the number of the nodes, N_i is a shape function corresponding to node i and u_i are the nodal values of u . Then, substituting (8) into (7) we arrive at a system of ordinary differential equations,

$$\mathbf{M} \mathbf{a} + \mathbf{K} \mathbf{v} = \mathbf{F} \quad (9)$$

where $\mathbf{v} = \{u_1, u_2, \dots, u_{n_{nodes}}\}^t$ is the vector of nodal values of u , \mathbf{a} is its time derivative, \mathbf{M} is called the “mass” matrix, \mathbf{K} is called the “stiffness” matrix and \mathbf{F} is called the “load” vector. For linear triangles the interpolation within an element is simply,

$$u^e(\mathbf{x}) \cong \sum_{i=1}^3 N_i(\mathbf{x}) u_i \quad (10)$$

where N_1, N_2 and N_3 are the conventional shape functions [9]. Proceeding in the standard manner, matrix \mathbf{M} and \mathbf{K} are built from element contributions and it is convenient to identify their terms:

$$\begin{aligned}
\mathbf{M} &= \mathbf{A}_{e=1}^{nel}(\mathbf{m}^e) \\
\mathbf{m}^e &= \mathbf{m}_g^e + \mathbf{m}_{pg}^e = \int_{\Omega_e} \mathbf{N}^T \mathbf{N} d\Omega^e + \int_{\Omega_e} \tau_{SUPG} \mathbf{B}^T \boldsymbol{\beta}^h \mathbf{N} d\Omega^e
\end{aligned} \tag{11}$$

$$\begin{aligned}
\mathbf{K} &= \mathbf{A}_{e=1}^{nel}(\mathbf{k}^e) \\
\mathbf{k}^e &= \mathbf{k}_g^e + \mathbf{k}_{pg}^e
\end{aligned} \tag{12}$$

$$= \int_{\Omega_e} (\mathbf{N}^T (\boldsymbol{\beta}^h)^T \mathbf{B} + \mathbf{B}^T \mathbf{k} \mathbf{B}) d\Omega^e + \int_{\Omega_e} \tau_{SUPG} \mathbf{B}^T (\boldsymbol{\beta}^h) (\boldsymbol{\beta}^h)^T \mathbf{B} d\Omega^e \tag{13}$$

where \mathbf{A} is the assembling operator, $\mathbf{N} = \{N_1, N_2, N_3\}^t$ is a vector containing the shape functions and \mathbf{B} is a matrix containing the derivatives of \mathbf{N} with respect to the spatial coordinates. The subscripts g and pg and in equations (11) and (12) identify the terms of the element matrix emanating respectively from the Galerkin and SUPG integrals.

2.2. Compressible flows - Euler Equation

The system of conservation laws governing inviscid, compressible fluid flow are the Euler equations. These equations, restricted to two spatial dimensions, may be written in terms of conservation variables $\mathbf{U} = (\rho, \rho u, \rho v, \rho e)$, as

$$\mathbf{U}_{,t} + \mathbf{F}_{x,x} + \mathbf{F}_{y,y} = \mathbf{0} \quad \text{on } \Omega \times [0, T] \tag{14}$$

where \mathbf{F}_x and \mathbf{F}_y are the Euler fluxes given elsewhere [10], Ω is a domain in \mathbb{R}^2 and T is a positive real number. We denote the spatial and temporal coordinates respectively by $\mathbf{x} = (x, y) \in \bar{\Omega}$ and $t \in [0, T]$, where the superimposed bar indicates set closure, and Γ is the boundary of domain Ω . Here ρ is the fluid density; $\mathbf{u} = (u, v)^T$ is the velocity vector; e is the total energy per unit mass. We add to equation (14) the ideal gas assumption, relating pressure with the total energy per unit mass and kinetic energy. Alternatively, equation (14) may be written as,

$$\mathbf{U}_{,t} + \mathbf{A}_x \mathbf{U}_{,x} + \mathbf{A}_y \mathbf{U}_{,y} = \mathbf{0} \quad \text{on } \Omega \times [0, T] \tag{15}$$

where $\mathbf{A}_i = \frac{\partial \mathbf{F}_i}{\partial \mathbf{U}}$. Associated to equation (15) we have proper boundary and initial conditions.

Considering a standard discretization of Ω into finite elements, the SUPG formulation for the Euler equations in conservation variables introduced by [11] and [12] is written as,

$$\begin{aligned}
& \int_{\Omega} \mathbf{W}^h \cdot \left(\frac{\partial \mathbf{U}^h}{\partial t} + \mathbf{A}_i^h \frac{\partial \mathbf{U}^h}{\partial x_i} \right) d\Omega + \\
& \sum_{e=1}^{nel} \int_{\Omega^e} (\mathbf{A}_k^h)^t \boldsymbol{\tau} \left(\frac{\partial \mathbf{W}^h}{\partial x_k} \right) \cdot \left[\frac{\partial \mathbf{U}^h}{\partial t} + \mathbf{A}_i^h \frac{\partial \mathbf{U}^h}{\partial x_i} \right] d\Omega + \\
& \sum_{e=1}^{nel} \int_{\Omega^e} \delta \frac{\partial \mathbf{W}^h}{\partial x_i} \cdot \frac{\partial \mathbf{U}^h}{\partial x_i} d\Omega = 0
\end{aligned} \tag{16}$$

where \mathbf{W}^h and \mathbf{U}^h are respectively, the discrete weighting and test functions, defined on standard finite element spaces. In (16) the first integral corresponds to the Galerkin formulation, the first series of element-level integrals are the SUPG stabilization terms, and the second series of element-level integrals are the shock-capturing terms added to the variational formulation to prevent spurious oscillations around shocks. The SUPG parameter used here is given as a stabilization matrix $\boldsymbol{\tau} = \tau \mathbf{I}$, where \mathbf{I} is the identity tensor (see Aliabadi and Tezduyar [13]). The shock-capturing parameter, δ , is evaluated here using the approach proposed by [14].

Considering the standard finite element approximation we have:

$$\mathbf{U}^h = \mathbf{N}\mathbf{v} \quad (17)$$

$$\mathbf{W}^h = \mathbf{N}\mathbf{c} \quad (18)$$

$$\mathbf{U}_{,t}^h = \mathbf{N}\mathbf{a} \quad (19)$$

where \mathbf{v} is the vector of nodal values of \mathbf{U} (depending on time only), \mathbf{c} is a vector of arbitrary constants, \mathbf{a} is the time derivate of \mathbf{v} ($\mathbf{a} = \frac{d\mathbf{v}}{dt}$) and \mathbf{N} is a matrix containing the space dependent shape functions. Restricting ourselves to linear triangles, the element shape functions may be represented in matrix form as,

$$\mathbf{N}^e = [N_1 \mathbf{I} \quad N_2 \mathbf{I} \quad N_3 \mathbf{I}] \quad (20)$$

where N_1, N_2 e N_3 are usual element shape functions [9] and \mathbf{I} is the identity matrix of order 4. Using theses approximations the spatial discretization of equation (16) leads to a set of coupled non-linear ordinary differential equations,

$$\mathbf{M}\mathbf{a} + \mathbf{C}(\mathbf{v}) = \mathbf{0} \quad (21)$$

where \mathbf{M} is the generalized "mass" matrix and \mathbf{C} is a non-linear vector function of \mathbf{v} . To solve the system of non-linear ordinary differential equations (21) towards steady-state we employ here the implicit predictor/multicorrector scheme described in detail in Hughes and Tezduyar [15]. In this scheme at each non-linear iteration (or multicorrection) we have to solve the following non-symmetric algebraic system of equations,

$$\mathbf{M}^* \Delta \mathbf{a} = \mathbf{R} \quad (22)$$

where $\mathbf{M}^* = \mathbf{M} + \alpha \Delta t \mathbf{K}$ is a $N \times N$ sparse matrix, $\mathbf{R} = -[\mathbf{M}\mathbf{a}^* + \mathbf{K}\mathbf{v}^*]$ is the residual vector, function of the predicted values of \mathbf{v} and \mathbf{a} , that is, \mathbf{v}^* , \mathbf{a}^* and $\Delta \mathbf{a}$ is the correction in the nodal values of \mathbf{a} from an iteration to the next. We adopt here $\alpha = 0.5$, which is second-order accurate in time. Both \mathbf{M} and \mathbf{K} are built from element contributions and it is convenient to identify their terms:

$$\begin{aligned} \mathbf{M} &= \mathbf{A}_{e=1}^{nel}(\mathbf{m}^e) \\ \mathbf{m}^e &= \mathbf{m}_g^e + \mathbf{m}_{pg}^e = \int_{\Omega^e} \mathbf{N}^T \mathbf{N} d\Omega^e + \int_{\Omega^e} \tau (\mathbf{B}_x^T \mathbf{A}_x^h \mathbf{N} + \mathbf{B}_y^T \mathbf{A}_y^h \mathbf{N}) d\Omega^e \end{aligned} \quad (23)$$

$$\begin{aligned} \mathbf{K} &= \mathbf{A}_{e=1}^{nel}(\mathbf{k}^e) \\ \mathbf{k}^e &= \mathbf{k}_g^e + \mathbf{k}_{pg}^e + \mathbf{k}_{dc}^e \end{aligned} \quad (24)$$

$$\begin{aligned} &= \int_{\Omega^e} \left(\mathbf{N}^T \mathbf{A}_x^h \mathbf{B}_x + \mathbf{N}^T \mathbf{A}_y^h \mathbf{B}_y \right) d\Omega^e + \\ &\int_{\Omega^e} \tau \mathbf{B}^T \begin{bmatrix} \mathbf{A}_x^h \mathbf{A}_x^h & \mathbf{A}_x^h \mathbf{A}_y^h \\ \mathbf{A}_y^h \mathbf{A}_x^h & \mathbf{A}_y^h \mathbf{A}_y^h \end{bmatrix} \mathbf{B} d\Omega^e + \\ &\int_{\Omega^e} \delta \mathbf{B}^T \mathbf{B} d\Omega^e \end{aligned} \quad (25)$$

where \mathbf{A} is the assembly operator, \mathbf{N} is a matrix containing the shape functions and \mathbf{B}_x , \mathbf{B}_y denotes respectively the derivatives of \mathbf{N} with respect to the spatial coordinates. The subscripts g , pg and dc in equations (23) and (24) identify the terms of the element matrix emanating respectively from the Galerkin, SUPG and discontinuity capturing integrals.

3. The Left Conjugate Direction algorithm

The finite element discretization of the scalar advective-diffusive equation and the Euler equations described on the previous section leads to a linear or nonlinear time-marching problem, respectively given in (9) for the advection-diffusive equation or in (21) for the compressible Euler equation. In any case we have to solve at each time step or nonlinear iteration a system of linear equations of the form,

$$Ax = b \quad (26)$$

where A is a $N \times N$ nonsymmetric sparse matrix, x is the vector of nodal unknowns and b is the out-of-balance force or residual vector. Both A and b are constructed assembling the element contributions:

$$A = \mathbf{A}_{e=1}^{nel} A^e \quad (27)$$

$$b = \mathbf{A}_{e=1}^{nel} b^e \quad (28)$$

where \mathbf{A} is the assembly operator, nel is the number of elements and A^e and b^e are the element matrices described in detail earlier.

The LCD method was recently introduced by Yuan et al [4]. In this method vectors $p_1, p_2, \dots, p_N \in \mathbb{R}^N$ are called left conjugate gradient vectors of an $N \times N$ real nonsingular matrix A if

$$\begin{aligned} p_i^T A p_j &= 0 \quad \text{for } i < j \\ p_i^T A p_j &\neq 0 \quad \text{for } i = j \end{aligned} \quad (29)$$

Suppose that the solution of the system (26) is x^* , and $\{p_1, p_2, \dots, p_N\}$ are left conjugate gradient vectors of A . Then it follows that

$$x^* = x_0 + \sum_{i=1}^N \alpha_i p_i \quad (30)$$

for every fixed vector x_0 . If r denotes the residual vector then

$$r = r_0 - \sum_{i=1}^N \alpha_i A p_i \quad (31)$$

where r_0 is the initial residual vector. To determine α_i , since p_1, p_2, \dots, p_N are linearly independent, then take r orthogonal to all p_i , that is

$$p_i^T r = 0 \quad \forall i = 1, \dots, N \quad (32)$$

From (32) we obtain

$$\alpha_i = \frac{p_i^T r_{i-1}}{p_i^T A p_i} \quad (33)$$

We also can write

$$r_i = b - A x_i = r_{i-1} - \alpha_i A p_i \quad (34)$$

$$x_i = x_0 + \sum_{k=1}^i \alpha_k p_k = x_{i-1} + \alpha_i p_i \quad (35)$$

From (33), (34) and (35) we can implement the left conjugate direction method if we know the set of linearly independent vectors p_1, p_2, \dots, p_N such that they are left conjugate gradient vectors of A . There is still a recurrence relation among p_1, p_2, \dots, p_k and r_k to compute the left conjugate gradient vector p_{k+1} , given in [4]:

$$\begin{aligned} q_0 &= r_k \\ \beta_i &= -\frac{p_i^T A q_{i-1}}{p_i^T A p_i} \\ q_i &= q_{i-1} + \beta_i p_i \quad \text{for } i = 1, \dots, k \\ p_{k+1} &= q_k \end{aligned} \quad (36)$$

In this case we need to know the first vector p_1 such that $p_1^T A p_1 \neq 0$. Putting all together, Yuan et al [4] described the complete left conjugate direction method as follows:

Algorithm 3.1

1. Input x_0 , A , p_1 such that $p_1^T A p_1 \neq 0$ and b ;
2. Calculate $r_0 = b - A x_0$;
3. For $k = 1, \dots, N$ do
 - 3.1 $q_k = A^T p_k$,
 $\alpha_k = \frac{p_k^T r_{k-1}}{q_k^T p_k}$,
 $x_k = x_{k-1} + \alpha_k p_k$,
 $r_k = b - A x_k = r_{k-1} - \alpha_k A p_k$;
 - 3.2 $p_{k+1} = r_k$,
 $\beta_i = -\frac{q_i^T p_{k+1}}{q_i^T p_i}$,
 $p_{k+1} = p_{k+1} + \beta_i p_i$ for $i = 1, \dots, k$.

In Algorithm 3.1 we need to store N vectors p_k and N vectors q_k to obtain the solution x_N . In this paper we considered a similar algorithm but with restart as in the GMRES algorithm implemented by Shakib et al [2], resulting in the LCD(k) algorithm with restart given below:

Algorithm 3.2 - LCD(k)

1. Given x_0 , A , b , l_{max} , k and ϵ_{tol} ;
2. $r_0 = b - A x_0$;
3. $\epsilon = \epsilon_{tol} \|r\|$;
4. Choose p_1 such that $p_1^T A p_1 \neq 0$;
5. For $l = 1, \dots, l_{max}$ do
 - 5.1. For $i = 1, \dots, k$ do
 - 5.1.1. $q_i = A^T p_i$,
 $\alpha_i = \frac{p_i^T r_{i-1}}{q_i^T p_i}$,
 $x_i = x_{i-1} + \alpha_i p_i$,
 $r_i = r_{i-1} - \alpha_i A p_i$;
 - 5.1.2. if $\|r_i\| < \epsilon$ then Exit l loop and x_i is the solution;
 - 5.1.3. $p_{i+1} = r_i$,
 For $j = 1, \dots, i$ do
 $\beta_j = -\frac{q_j^T p_{i+1}}{q_j^T p_j}$,
 $p_{i+1} = p_{i+1} + \beta_j p_j$;
 - 5.2. Choose the new p_1 such that $p_1^T A p_1 \neq 0$;

where l_{max} is the maximum number of iterations, ϵ_{tol} is the user supplied tolerance and k is number of left conjugate directions considered in the restart.

Remarks:

1. We need to store $2k$ N -dimensional vectors ($\{p_1, \dots, p_k\}$ and $\{q_1, \dots, q_k\}$).
2. For each iteration l we need two matrix-vector products ($q_k = A^T p_k$ and $r_k = r_{k-1} - \alpha_k A p_k$ - step 5.2.1). Note that in GMRES we just need one matrix-vector product per iteration.
3. To start LCD(k), we have to choose p_1 (step 4 in the Algorithm 3.2). We have the following options: $p_1 = b$, $p_1 = \text{diag}(A)^{-1} b$ or $p_1 = \text{diag}(A)$.

4. In every restart l there is the need to choose a new p_1^l (step 5.2 in Algorithm 3.2). The options here are: $p_1^l = p_{k+1}^{l-1}$, $p_1^l = r^{l-1}$ or $p_1^l = x^{l-1}$.
5. Our numerical experiments will indicate which are the best options to select the initial vector in both cases.

3.1. LCD Element-by-Element Preconditioning

To accelerate convergence of the LCD algorithm, we have used an element-by-element preconditioning strategy. For the advective-diffusive equation the element-by-element Gauss-Seidel preconditioner begins by rewriting the system of linear equations (26) in scaled form,

$$\tilde{A}\tilde{x} = \tilde{b} \quad (37)$$

$\tilde{A} = W^{-1/2}AW^{-1/2}$, $\tilde{x} = W^{1/2}x$, $\tilde{b} = W^{1/2}b$ and $W = \text{diag}(A)$. Thus we define the preconditioned system of equations to be solved as,

$$\bar{A}\bar{x} = \bar{b} \quad (38)$$

where $\bar{A} = L^{-1}\tilde{A}U^{-1}$, $\bar{x} = U\tilde{x}$ and $\bar{b} = U^{-1}\tilde{b}$. The matrices L and U are respectively the left and right preconditioning, defined as

$$L = \mathbf{A}_{e=1}^{nel} L_e \quad (39)$$

$$U = \mathbf{A}_{e=1}^{nel} U_e \quad (40)$$

where element matrices L_e and U_e are the Gauss-Seidel factors of the regularized element array,

$$L_e + U_e = \tilde{A}_e - \text{diag}(\tilde{A}_e) + I \quad (41)$$

For the Euler equation a nodal block-diagonal preconditioner is used. In this case, $\tilde{A} = BD^{-1}A$, $\tilde{b} = BD^{-1}b$, $BD = \mathbf{A}_{ino=1}^{nnodes} BD_{ino}$ and BD_{ino} is the 4×4 block-diagonal matrix for each node.

4. Numerical Results

4.1. 2D advection-diffusion problem

We consider a pure convection of a scalar on a square domain, where the advection is skew to the mesh and the diffusivity is negligible. Figure 1 shows the problem set up. The domain is the unit square $\Omega = [0, 1] \times [0, 1]$ and the boundary conditions are

$$\begin{aligned}
 u &= 0.0 & \text{along} & \quad y = 0.0 \\
 u &= 0.0 & \text{along} & \quad x = 0.0 \text{ and } 0.0 < y < 0.25 \\
 u &= 1.0 & \text{along} & \quad x = 0.0 \text{ and } 0.25 < y < 1.0
 \end{aligned}
 \tag{42}$$

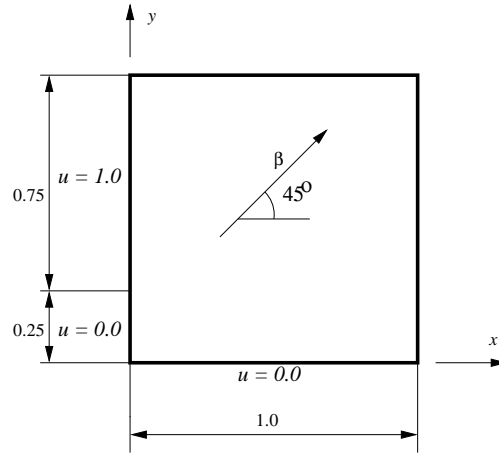


Figure 1. Pure convection of a scalar on a square domain - Problem set up

The diffusivity is $k = 1 \times 10^{-7}$, the flow direction is 45° from the x -axis and $\|\beta\| = 1$. The domain is discretized by grids of triangular elements with 64×64 , 128×128 , 256×256 and 512×512 cells. Each cell is subdivided into four triangles.

Figures 2 and 3 show, respectively, the solution with the GMRES(5) and LCD(5) solutions for the mesh with 64×64 cells. We observed that both solutions are virtually identical.

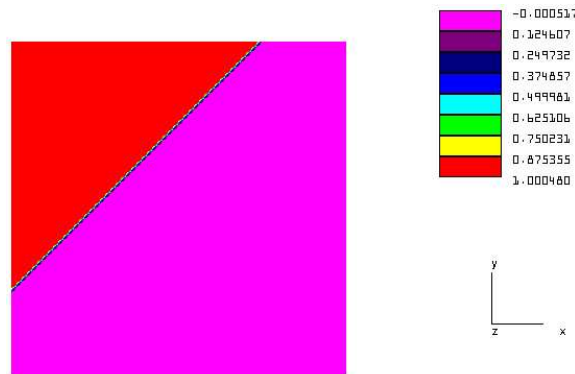


Figure 2. Pure convection of a scalar on a square domain - Mesh with 64×64 cells - GMRES(5) solution.

Figure 4(a) shows the LCD residual behavior for the three choices of the first left conjugate vector. The best choice is $p_1 = b$. In Figure 4(b), we can observe that $p_1^l = p_{k+1}^{l-1}$ is the best

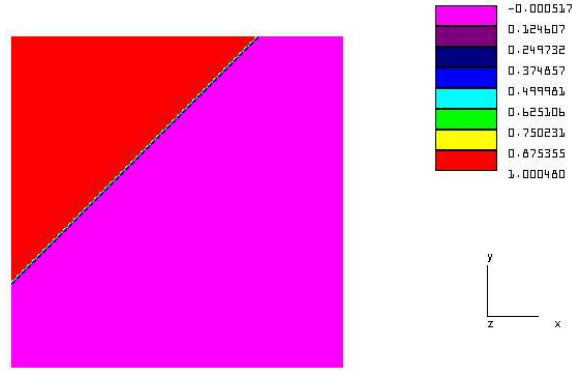


Figure 3. Pure convection of a scalar on a square domain - Mesh with 64×64 cells - LCD(5) solution.

option for the new left conjugate vector in every restart of the LCD algorithm. Therefore, from now on we are going to use these two options.

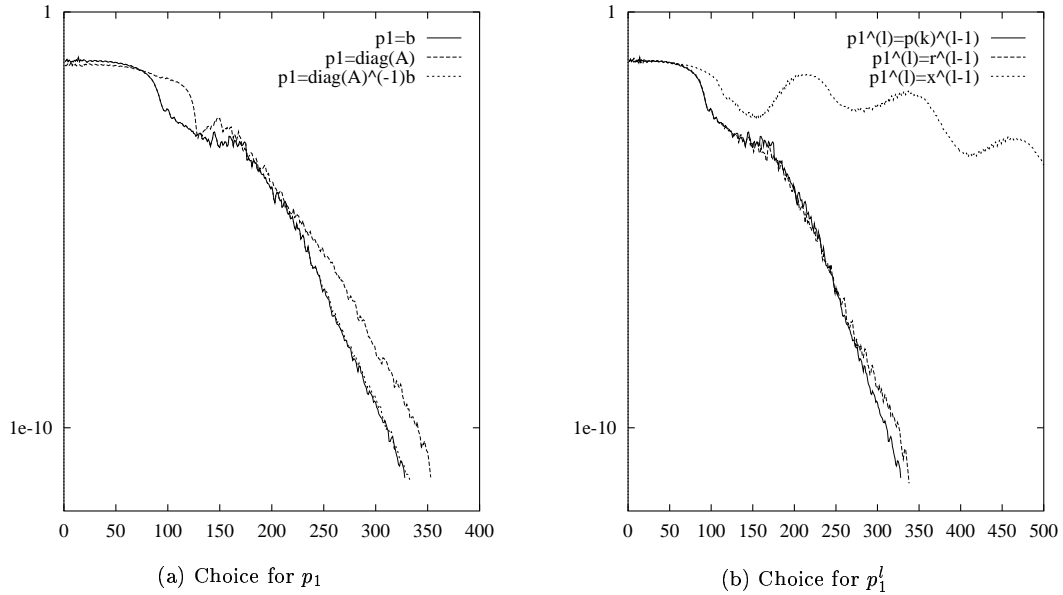


Figure 4. Pure convection of a scalar on a square domain - Mesh with 64×64 cells - Choice of the left conjugate starting vector on LCD algorithm.

Figure 5 compares the relative residual evolution for LCD(5) and GMRES(5) for the four meshes defined before. Iterations are halted when relative residual reaches a tolerance of 10^{-10} . Although relative residual in LCD(5) decreases more slowly than in GMRES(5) in the beginning of the process, the total number of LCD(5) iterations is smaller than the number of

GMRES(5) iterations. A similar behavior is observed for all meshes.

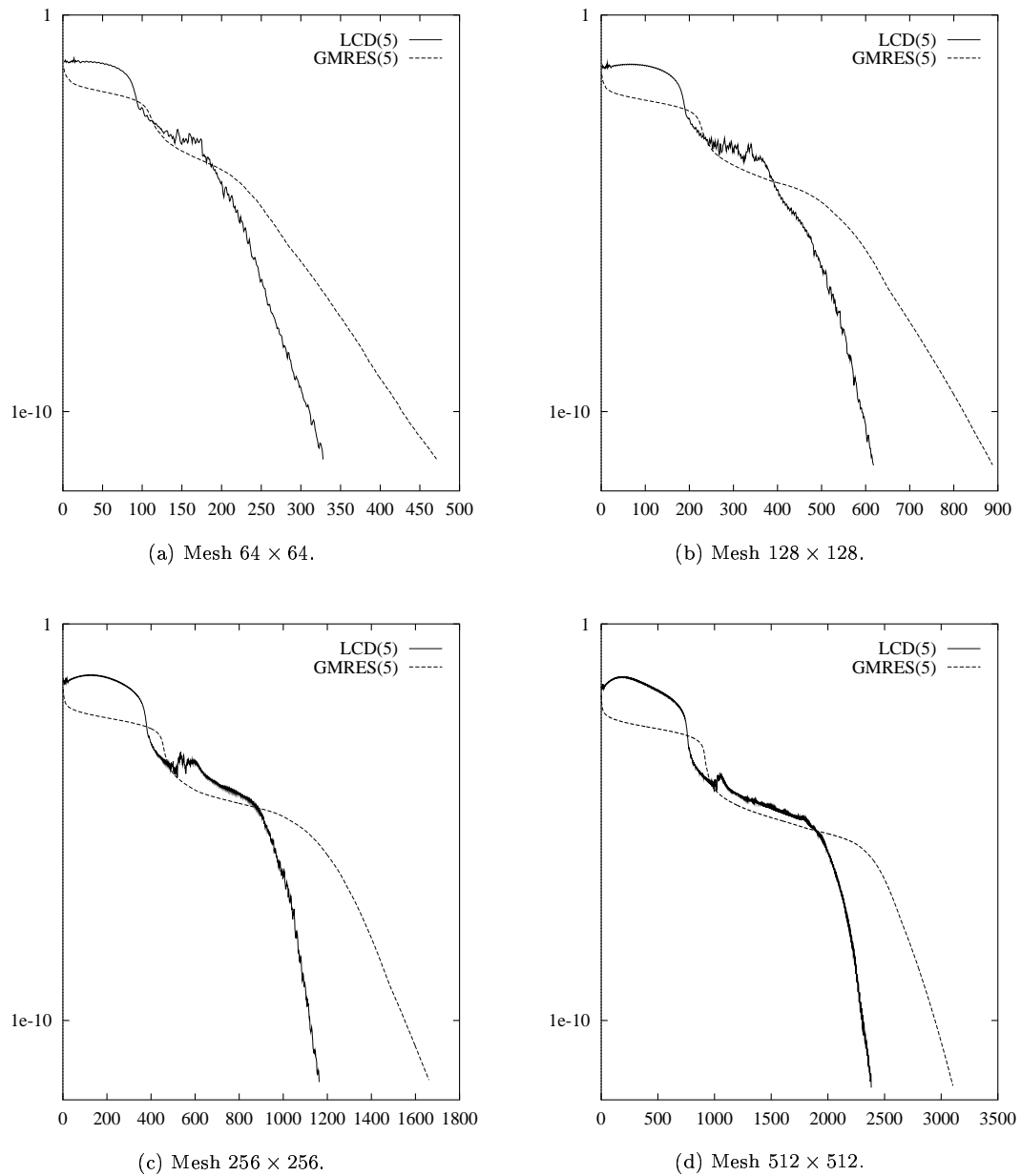


Figure 5. Pure convection of a scalar on a square domain - Relative residual evolution for LCD(5) and GMRES(5).

Figure 6 shows the performance of GMRES and LCD methods for $k=1, 5, 10$ and 20 considering the mesh of 128×128 cells. Results for the other meshes are similar. Table 1 shows the number of iterations ($Niter$) and CPU times for the GMRES and LCD methods using a relative residual tolerance of 10^{-10} . In this Table Neq is the number of the unknowns. We can observe that the LCD method converges with less iterations, however it is slower than GMRES method in all cases.

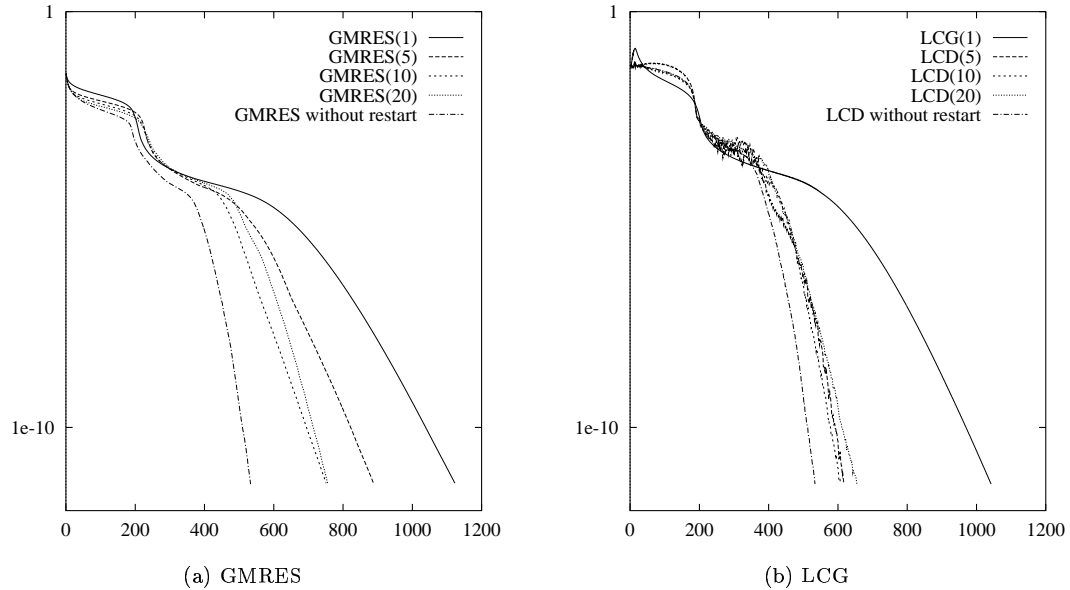


Figure 6. Pure convection of a scalar on a square domain - Relative residual evolution for LCD(k) and GMRES(k) - mesh of 128×128 cells.

Table 2 presents the required memory for each case using the mesh of 128×128 cells. We can observe that the case without restart converges with less iterations than the other cases, however it is slower and needs more memory than the others. We can also observe that as more vectors are used, more memory is necessary.

For each iteration the LCD method computes two matrix-vector products, one with A , the other involving A^t , while GMRES needs just one. In Table 3 we show for the mesh with 128×128 cells the time required for each matrix-vector product for both methods. Note that times for the product with the transpose are of the same order of magnitude of the Ap product, thanks to the element-by-element data structure.

Next we study the action of the Gauss-Seidel element-by-element preconditioner in LCD(5) and GMRES(5) using again the mesh with 128×128 and a residual tolerance of 10^{-10} . We compare in Figure 7 the relative residual evolution of LCD(5), GMRES(5) and two other popular members of the Krylov family of iterative methods, Bi-CGSTAB [16] and TFQMR [17], all using the Gauss-Seidel element-by-element preconditioner. We may note in Figure 7 that LCD and GMRES require more iterations than the other methods, but their convergence is

Table I. Computational costs

1 vector					
Mesh		GMRES(1)		LCD(1)	
Cells	Neq	Niter	Time(sec)	Niter	Time(sec)
64 × 64	8192	714	14	654	14
128 × 128	32768	1123	114	1042	114
5 Vectors					
Mesh		GMRES(5)		LCD(5)	
Cells	Neq	Niter	Time(sec)	Niter	Time(sec)
64 × 64	8192	471	6	328	7
128 × 128	32768	888	62	618	70
256 × 256	131072	1661	500	1163	580
512 × 512	524288	3104	4096	2384	4592
10 Vectors					
Mesh		GMRES(10)		LCD(10)	
Cells	Neq	Niter	Time(sec)	Niter	Time(sec)
64 × 64	8192	399	5	356	8
128 × 128	32768	751	56	608	77
256 × 256	131072	1479	490	1091	582
20 Vectors					
Mesh		GMRES(20)		LCD(20)	
Cells	Neq	Niter	Time(sec)	Niter	Time(sec)
64 × 64	8192	448	6	401	10
128 × 128	32768	756	70	655	95
256 × 256	131072	1383	554	1123	699
without restart					
Mesh		GMRES		LCD	
Cells	Neq	Niter	Time(sec)	Niter	Time(sec)
64 × 64	8192	261	11	262	20
128 × 128	32768	533	565	534	609

Table II. Memory requirements for the mesh of 128 × 128 cells

Number of vectors	GMRES(Mwords)	LCD (Mwords)
1	1.57	1.77
5	1.84	2.36
10	2.16	2.95
20	2.82	4.26
without restart	41.56	80.29

Table III. Time required for matrix-vector operations - mesh 128×128 cells

operation	Time (sec)
$A^t p$ in LCD	0.052
$A p$ in LCD or GMRES	0.050

smooth, showing an ever decreasing residual. However, as shown in Table 4, LCD is the slowest method. We may attribute this again to the fact that LCD is the only method requiring two matrix-vector products per iteration.

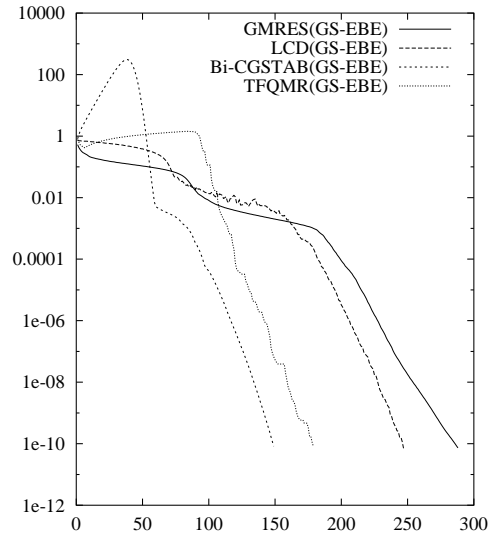


Figure 7. Pure convection of a scalar on a square domain - Mesh with 128×128 cells - Relative residual evolution for LCD(5), GMRES(5), Bi-CGSTAB and TFQMR with Gauss-Seidel EBE preconditioner.

Table 4 shows the solution times required for each mesh. We can observe the same conclusion above, that is, the LCD solutions converge with less iterations, however it is more slow than GMRES in all cases.

Table IV. Computational costs - Mesh with 128×128 cells - Gauss-Seidel EBE preconditioner

Method	Niter	CPU Time(sec)
LCD(5)	247	48
GMRES(5)	288	34
Bi-CGSTAB	149	41
TFQMR	180	34

4.2. Compressible Flow - Oblique Shock

This problem consists of a two-dimensional steady problem of a inviscid, Mach 2, uniform flow, over a wedge at an angle of -10° with respect to a horizontal wall, resulting in the occurrence of an oblique shock with an angle of 29.3° emanating from the leading edge of the wedge, as shown in Figure 1. Multicorrections are fixed to 3 and we considered a fixed time step of 0.01.

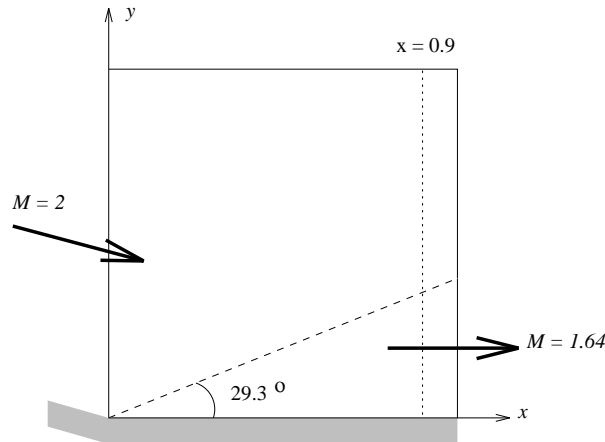


Figure 8. Oblique shock - problem description.

The computational domain is the square $0 \leq x \leq 1$ and $0 \leq y \leq 1$. Prescribing the following flow data at the inflow, i.e., on the left and top sides of the shock, results in the exact solution with the flow data past the shock:

$$\text{Inflow} \begin{cases} M &= 2.0 \\ \rho &= 1.0 \\ u_1 &= \cos 10^\circ \\ u_2 &= -\sin 10^\circ \\ p &= 0.17857 \end{cases} \quad \text{Outflow} \begin{cases} M &= 1.64052 \\ \rho &= 1.45843 \\ u_1 &= 0.88731 \\ u_2 &= 0.0 \\ p &= 0.30475 \end{cases} \quad (43)$$

where M is the Mach number, ρ is the flow density, u_1 and u_2 are the horizontal and vertical velocities respectively, and p is the pressure.

Four Dirichlet boundary conditions are imposed on the left and top boundaries; the slip condition $u_2 = 0$ is set at the bottom boundary; and no boundary conditions are imposed on the outflow (right) boundary. A 20×20 mesh with 800 linear triangles and 441 nodes is employed. All the solutions are initialized with free-stream values. The number of multicorrections is fixed to 3. Nodal block-diagonal preconditioning is used here for all methods.

Catabriga and Coutinho presented in [18] a procedure to detect convergence stagnation in the computation of inviscid flows forcing the solution to converge to steady-state. A simple heuristic is used to detect convergence stagnation and then to stop updating the shock-capturing term. As a consequence, the problem converges very fast towards a steady-state solution with no accuracy degradation. Figure 9(a) shows density along line $x = 0.9$ for LCD(5), GMRES(5),

Bi-CGSTAB and TFQMR solutions and Figure 9(b) shows the evolution of density residual for all methods. Within each multicorrection the linear systems tolerance is set to 0.1 for all methods. We may observe that all methods yield solutions with comparable accuracy, but the Bi-CGSTAB solution required less iterations than all the others to reach machine zero. In Table 5 we list the number of steps ($Nsteps$), the total number of iterations ($Niter$) and the CPU times to reach a relative density residual of 10^{-10} . The fastest solution was obtained by GMRES, although it required more steps and iterations than the Bi-CGSTAB and TFQMR solutions. The slowest solution was obtained by LCD.

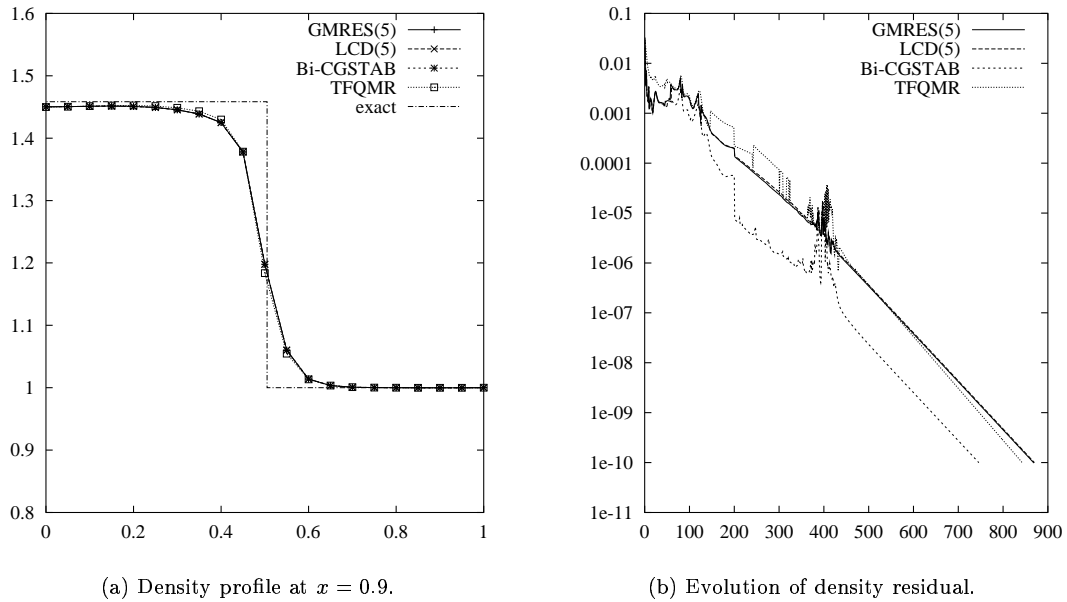


Figure 9. Oblique shock - Solution with both methods.

Table V. Oblique shock - Computational costs.

Method	Nsteps	Niter	Time(sec)
LCD(5)	870	5285	134
GMRES(5)	869	5226	117
Bi-CGSTAB	746	3520	128
TFQMR	843	4296	142

4.3. Compressible flow - Reflected Shock

This two-dimensional steady problem consists of three regions (R1, R2 and R3) separated by an oblique shock and its reflection from a wall, as shown in Figure 10. Prescribing the following Mach 2.9 flow data at the inflow, i.e., the first region on the left (R1), and requiring that the

incident shock to be at an angle of 29° , leads to the exact solution (R2 and R3):

$$\text{R1} \begin{cases} M = 2.9 \\ \rho = 1.0 \\ u_1 = 2.9 \\ u_2 = 0.0 \\ p = 0.714286 \end{cases} \quad \text{R2} \begin{cases} M = 2.3781 \\ \rho = 1.7 \\ u_1 = 2.61934 \\ u_2 = -0.50632 \\ p = 1.52819 \end{cases} \quad \text{R3} \begin{cases} M = 1.94235 \\ \rho = 2.68728 \\ u_1 = 2.40140 \\ u_2 = 0.0 \\ p = 2.93407 \end{cases} \quad (44)$$

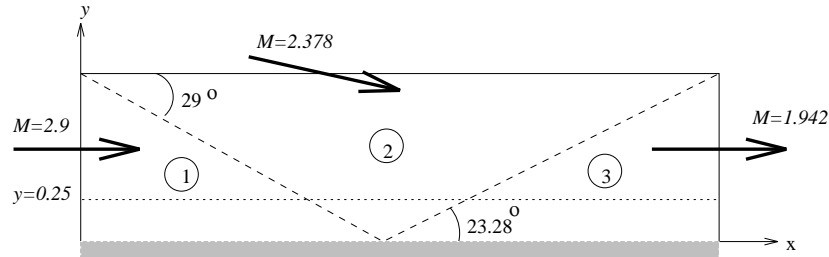


Figure 10. Reflected Shock - Problem Description.

We prescribe density, velocities and pressure on the left and top boundaries; the slip condition is imposed on the wall (bottom boundary); and no boundary conditions are set on the outflow (right) boundary. Multicorrections are fixed to 3 and we considered a fixed time step of 0.01. We consider a structured mesh with 60×20 cells, where each cell was divided into two triangles (1281 nodes and 2400 elements) and an unstructured mesh with 1,837 nodes and 3,429 elements covering the domain $0 \leq x \leq 4.1$ and $0 \leq y \leq 1$. The tolerance of the preconditioned LCD(5), GMRES(5) Bi-CGSTAB and TFQMR methods for each step is 0.1 and all the solutions are initialized with free-stream values.

Figure 11(a) shows the density along line $y = 0.25$ of the structured mesh for LCD(5), GMRES(5), Bi-CGSTAB and TFQMR solutions. Figure 11(b) shows the evolution of density residual when the convergence stagnation detection procedure is used. Table 6 shows the number of steps (*Nsteps*) and CPU times of all solutions to reach a relative density residual of 10^{-10} . In this case GMRES and Bi-CGSTAB solution are the fastest, although the latter solution required less steps and iterations. Again the CPU time of LCD solution was not competitive with the others.

Table VI. Oblique shock - Computational costs - Structured mesh.

Method	Nsteps	Niter	Time(sec)
LCD(5)	618	4159	332
GMRES(5)	612	4977	276
Bi-CGSTAB	610	1905	279
TFQMR	589	2603	298

Figures 12 shows the density contours for the unstructured mesh, for solutions obtained with all iterative methods. We observe that contours are virtually identical.

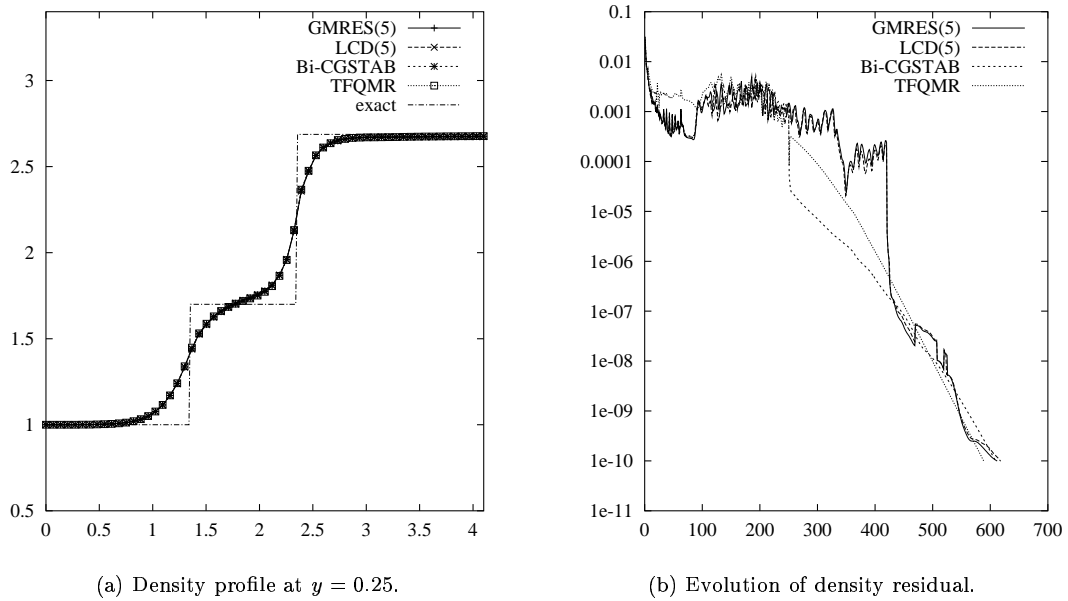


Figure 11. Reflected shock - Structured mesh - Solution with all methods.

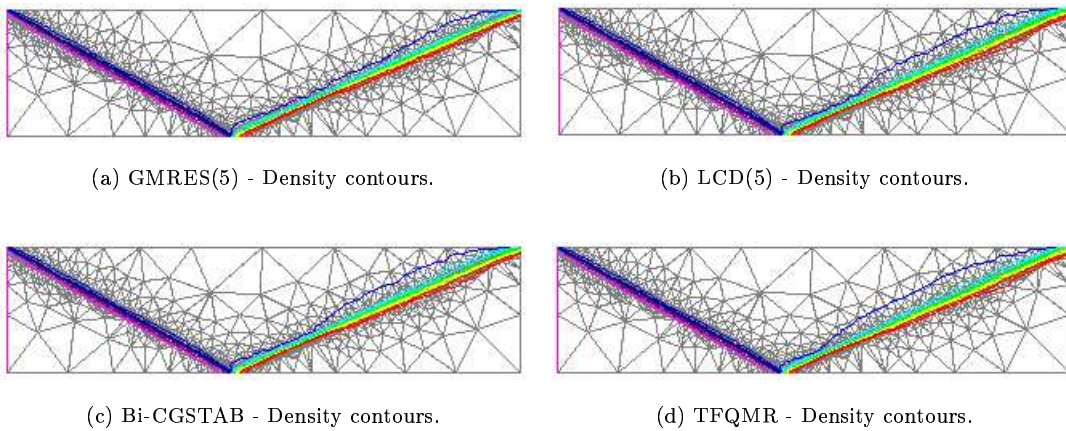


Figure 12. Reflected shock - Unstructured mesh.

Figure 13 shows the evolution of density residual when the convergence stagnation detection procedure is used and Table 7 shows the number of steps, iterations and CPU times to reach a relative density residual of 10^{-10} . Here also GMRES converges in less time than all other methods. The most time consuming solution was obtained by LCD.

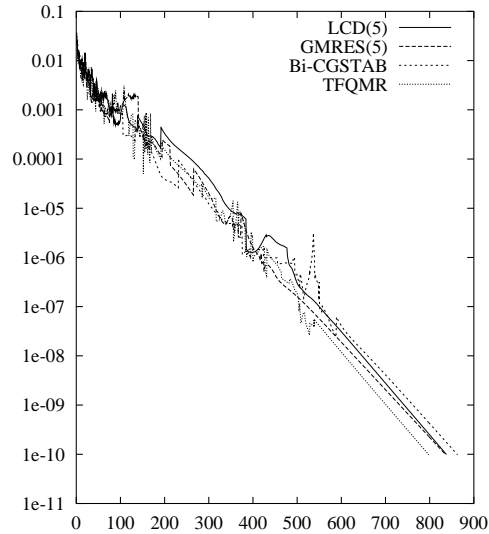


Figure 13. Reflected shock - Evolution of density residual in the unstructured mesh.

Table VII. Oblique shock - Computational costs - Unstructured mesh.

Method	Nsteps	Niter	Time(sec)
LCD(5)	838	13498	1222
GMRES(5)	836	9573	730
Bi-CGSTAB	863	7930	1025
TFQMR	798	10286	1000

5. Concluding remarks

In this work we studied the performance of LCD algorithm, a new iterative method where a special Krylov subspace is built, obtaining the exact solution by solving a triangular system rather than a general least-squares problem. Its nice mathematical properties and a prototype MATLAB implementation have shown the effectiveness of LCD when compared to QMR, Bi-CGSTAB and GMRES.

That motivated us to investigate the performance of LCD within a finite element computer program to solve inviscid flow problems. We have extended the original algorithm to incorporate restarts and typical finite element preconditioners. We have shown by numerical

experimentation that starting the iterations taking the as the first left conjugate vector the right hand side vector is the best choice. However, to restart the iterations we need to take as starting vector the last left conjugate vector from the previous cycle.

Comparisons with other Krylov space methods with or without preconditioning unfortunately do not favour LCD. Although requiring usually less iterations, CPU times and memory are larger than GMRES, Bi-CGSTAB and TFQMR. The main reason is the need to compute two matrix-vector product per iteration, one with the coefficient matrix and the other with its transposed matrix. Those are the dominant costs, although thanks to the element-by-element data structure both are performed with the same efficiency.

Those results should be viewed as a first attempt to incorporate LCD into the methods available for solving finite element linear systems of equations and certainly much more experiments are needed before reaching a final conclusion on its effectiveness.

6. Acknowledgments

This work is partly supported by grant CNPq/PROTEM-CC 68.0066/01-2 and CNPq/MCT 522692/95-8. During the course of this work the third author was visiting LNCC/Brazil and would like to thank their hospitality and support. This research has been partially supported by the University of Colorado at Denver and CNPq.

REFERENCES

1. Y. Saad, *Iterative Methods for Sparse Linear Systems*, PWS Publishing, Boston, 1996.
2. F. Shakib, T. J. R. Hughes, Z. Johan, A multi-element group preconditioned gmres algorithm for nonsymmetric systems arising in finite element analysis, *Comput. Methods Appl. Mech. and Engrg.* 65 (1989) 415–456.
3. T. E. Tezduyar, J. Liou, Grouped element-by-element iteration schemes for incompressible flow simulations, *Computer Physics Communication* 53 (1989) 441–453.
4. J. Yuan, G. Golub, R. Plemmons, W. A. G. Cecilio, Semi-conjugate direction methods for nonsymmetric systems, *Scientific Computing/Computational Mathematics Program SCCM-02-02*, Stanford University (2002).
5. G. J. Le Beau, T. E. Tezduyar, Finite element computation of compressible flows with the SUPG formulation, in: M. Dhaubhadel, M. Engelman, J. Reddy (Eds.), *Advances in Finite Element Analysis in Fluid Dynamics*, Vol. 123, ASME, New York, 1991, pp. 21–27.
6. L. Catabriga, A. L. G. A. Coutinho, Implicit supg solution of euler equations using edge-based data structures, *Comput. Methods Appl. Mech. and Engrg.* 191 (2002) 3477–3490.
7. A. Brooks, T. Hughes, Streamline Upwind/Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations, *Comput. Methods Appl. Mech. and Engrg.* 32 (1982) 199–259.
8. L. Franca, S. Frey, T. Hughes, Stabilized finite element methods: Application to the advective-diffusive model, *Comput. Methods Appl. Mech. and Engrg.* 95 (1992) 253–276.
9. T. Hughes, *The Finite Element Method*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1987.
10. C. Hirsh, *Numerical Computation of Internal and External Flows - Computational Methods for Inviscid and Viscous Flows*, Vol. 2, John Wiley and Sons Ltd, Chichester, 1992.
11. T. E. Tezduyar, T. J. R. Hughes, Development of time-accurate finite element techniques for first-order hyperbolic systems with particular emphasis on the compressible Euler equations, *NASA Technical Report NASA-CR-204772*, NASA (1982).
12. T. E. Tezduyar, T. J. R. Hughes, Finite element formulations for convection dominated flows with particular emphasis on the compressible Euler equations, in: *21st Aerospace Sciences Meeting*, Vol. 83-0125, AIAA, Reno, Nevada, 1983.
13. S. K. Aliabadi, T. E. Tezduyar, Parallel fluid dynamics computations in aerospace applications, *International Journal for Numerical Methods in Fluids* 21 (1995) 783–805.

14. G. J. Le Beau, T. E. Tezduyar, Finite element computation of compressible flows with the SUPG formulation, in: M. Dhaubhadel, M. Engelman, J. Reddy (Eds.), *Advances in Finite Element Analysis in Fluid Dynamics*, Vol. 123, ASME, New York, 1991, pp. 21–27.
15. T. J. R. Hughes, T. E. Tezduyar, Finite element methods for first-order hyperbolic systems with particular emphasis on the compressible Euler equations, *Comput. Methods Appl. Mech. and Engrg.* 45 (1984) 217–284.
16. Y. T. Feng, G. J. Huang, D. R. J. Owen, D. Peric, An evaluation of iterative methods in the solution of a convection-diffusion problems, *Numerical Methods in Laminar and Turbulent Flow 8* (1993) 1567–1579.
17. R. W. Fleund, A transpose-free quasi-minimal residual methods for non-hermetian linear systems, *Numerical Analysis Manuscripts 92-07* (1992).
18. L. Catabriga, A. L. G. A. Coutinho, Improving convergence to steady-state of implicit SUPG solution of Euler equations, *Communications in Numerical Methods in Engineering* 18 (2002) 345–353.