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

Lucia Catabriga¹, Alvaro L.G.A. Coutinho²,* 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 Cecílio 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@macad.ufrj.br

Received 10 March 2003

Copyright © 2003 John Wiley & Sons, Ltd.

Accepted 3 September 2003
element preconditioners, using the same data structures of the matrix-vector product, have been proposed earlier [2, 3].

Recently Yuan, Golub, Plemmons and Cecílio [4] introduced a new algorithm for solving non-symmetric, non-singular linear systems, the Left Conjugate Direction (LCD for short) method. This method is based on the concept of left and right conjugate vectors for non-symmetric and non-singular 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. Silva, Raupp and Almeida [5] have applied the LCD method to solve linear systems arising from the stabilized finite element solution of thermal pollution problems. That motivated us to investigate the performance of LCD. Therefore, in this work we evaluate the performance of LCD in the solution of non-symmetric systems of linear equations arising from the implicit semi-discrete SUPG finite element formulation for advection-diffusion and inviscid compressible flows described in [6, 7]. 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:

\[ u_t + \sigma^a_{i,i} + \sigma^d_{i,i} = f, \]

where \( u \) represents the quantity being transported (e.g. temperature, concentration), \( \sigma^a_i = \beta_i \) is the advective flux, \( \beta_i \) is the flow with \( \nabla \beta = 0 \), \( \sigma^d_i = -\kappa_{ij} u_{,i} \) is the diffusive flux and \( \kappa_{ij} \) is the volumetric diffusivity given as,

\[ \kappa = \begin{bmatrix} \kappa_x & 0 \\ 0 & \kappa_y \end{bmatrix}. \]  

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

\[ \frac{\partial u}{\partial t} + \beta \cdot \nabla u - \nabla \cdot (\kappa \nabla u) = f. \]
The essential and natural boundary conditions appended to equation (3) are:

$$u = g \quad \text{on } \Gamma_g,$$
$$n \cdot \kappa \nabla u = h \quad \text{on } \Gamma_h,$$

(4)

where $g$ and $h$ are given functions of $x = (x, y)$ and $t$, $n$ is the unit outward normal vector at the boundary, $\Gamma_g$ and $\Gamma_h$ are the complementary subsets of $\Gamma$ where boundary conditions are prescribed. Initial conditions are given by,

$$u(x, 0) = u_0(x) \quad \text{on } \Omega.$$  

(5)

Consider a finite element discretization of $\Omega$ into elements $\Omega_e, e = 1, \ldots, n_{el}$, 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

$$\int_{\Omega} \left( w^h \frac{\partial u^h}{\partial t} + w^h \beta^h \nabla u^h - \nabla w^h \cdot \kappa \nabla u^h \right) d\Omega +$$
$$\sum_{e=1}^{n_{el}} \int_{\Omega_e} \tau_{SUPG} \beta^h \nabla w^h \left( \frac{\partial u^h}{\partial t} + \beta^h \nabla u^h \right) d\Omega =$$
$$\int_{\Omega} w^h f d\Omega + \sum_{e=1}^{n_{el}} \int_{\Omega_e} \tau_{SUPG} \beta^h \nabla w^h f d\Omega,$$

(6)

where $\tau_{SUPG}$ is the SUPG stabilization parameter which may computed as suggested in [8] and [9]. Let the standard finite element approximation be given as follows:

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

(7)

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 (7) into (6) we arrive at a system of ordinary differential equations,

$$Ma + Kv = F,$$  

(8)

where $v = \{ u_1, u_2, \ldots, u_{n_{nodes}} \}^t$ is the vector of nodal values of $u$, $a$ is its time derivative, $M$ is called the “mass” matrix, $K$ is called the “stiffness” matrix and $F$ is called the “load” vector. In this work we restrict ourselves to linear triangles only. Thus, the interpolation within an element is simply,

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

(9)

where $N_1, N_2$ and $N_3$ are the conventional shape functions [10]. Proceeding in the standard manner, matrix $M$ and $K$ are built from element contributions and it is convenient to identify their terms.
where $A$ is the assembling operator, $N = \{N_1, N_2, N_3\}^t$ is a vector containing the shape functions and $B$ is a matrix containing the derivatives of $N$ with respect to the spatial coordinates. The subscripts $g$ and $pg$ and in equations (10) and (11) 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 flows are the Euler equations. In two dimensions these equations can be written in terms of the conservation variables, $U = (\rho, \rho u, \rho v, \rho e)$, as

$$ U_t + F_{x,x} + F_{y,y} = 0 \text{ on } \Omega \times [0,T]. $$

Here $\rho$ is the fluid density, $u = (u, v)$ is the velocity vector, $e$ is the total energy per unit mass, $F_x$ and $F_y$ are the Euler fluxes, $\Omega$ is a domain in $\mathbb{R}^2$, and $T$ is a positive real number. We denote the spatial and temporal coordinates respectively by $x = (x, y) \in \overline{\Omega}$ and $t \in [0,T]$, where the superimposed bar indicates set closure, and $\Gamma$ is the boundary of domain $\Omega$. We consider ideal gases. Alternatively, Equation (12) can be written as

$$ U_t + A_x U_x + A_y U_y = 0 \text{ on } \Omega \times [0,T], $$

where $A_x = \frac{\partial F}{\partial x}$ and $A_y = \frac{\partial F}{\partial y}$. Associated to Equation (13) we have proper set of boundary and initial conditions.

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

$$ \int_\Omega W^h : \left( \frac{\partial U^h}{\partial t} + A_x^h \frac{\partial U^h}{\partial x} + A_y^h \frac{\partial U^h}{\partial y} \right) \, d\Omega + $$

$$ \sum_{e=1}^{nel} \int_{\Omega^e} \left( \frac{\partial W^h}{\partial x} A_x^h + \frac{\partial W^h}{\partial y} A_y^h \right) : \left( \frac{\partial U^h}{\partial t} + A_x^h \frac{\partial U^h}{\partial x} + A_y^h \frac{\partial U^h}{\partial y} \right) \, d\Omega + $$

$$ \sum_{e=1}^{nel} \int_{\Omega^e} \delta \left( \frac{\partial W^h}{\partial x} \cdot \frac{\partial U^h}{\partial x} + \frac{\partial W^h}{\partial y} \cdot \frac{\partial U^h}{\partial y} \right) \, d\Omega = 0. \quad (14) $$


Prepared using nmeauth.cls
where $W^h$ and $U^h$ are the finite-dimensional test and trial functions that are defined on standard finite element spaces. In Equation (14) 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 scalar stabilization parameter $\tau$ is the same as the one used in [13], which is a slightly modified version of the one introduced in [11, 12]. The shock-capturing parameter, $\delta$, is evaluated here using the approach proposed by [14].

Considering the standard finite element approximation we have:

$$
\begin{align*}
U^h &= Nv, \\
W^h &= Nc, \\
U^h_t &= Na,
\end{align*}
$$

(15)

where $v$ is the vector of nodal values of $U$ (depending on time only), $c$ is a vector of arbitrary constants, $a$ is the time derivate of $v$ ($a = \frac{dv}{dt}$) and $N$ 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,

$$
N^e = \begin{bmatrix} N_1 I & N_2 I & N_3 I \end{bmatrix},
$$

(16)

where $N_1$, $N_2$ and $N_3$ are usual element shape functions [10] and $I$ is the identity matrix of order 4. Using these approximations the spatial discretization of equation (14) leads to a set of coupled non-linear ordinary differential equations,

$$
Ma + C(v) = 0,
$$

(17)

where $M$ is the generalized "mass" matrix and $C$ is a non-linear vector function of $v$. To solve the system of non-linear ordinary differential equations (17) 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,

$$
M^* \Delta a = R,
$$

(18)

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

$$
\begin{align*}
M &= \sum_{e=1}^{nel} A^e(m^e) \\
m^e &= m^e_g + m^e_{pg} = \int_{\Omega_e} N^T N d\Omega^e + \int_{\Omega_e} \tau (B^T A^h_x N + B^T y A^h_y N) d\Omega^e,
\end{align*}
$$

(19)
\[
K = \sum_{e=1}^{nel} (k_e)
\]

\[
k_e = k_{g}^{e} + k_{pg}^{e} + k_{dc}^{e}
\]

\[
= \int_{\Omega^e} \left( N^T A^h_x B_x + N^T A^h_y B_y \right) d\Omega^e + \int_{\Omega^e} \tau B^T \left[ \begin{array}{cc} A^h_x A^h_x & A^h_x A^h_y \\ A^h_y A^h_x & A^h_y A^h_y \end{array} \right] B d\Omega^e + \int_{\Omega^e} \delta B^T B d\Omega^e,
\]

where \( A \) is the assembly operator, \( N \) is a matrix containing the shape functions and \( B_x, B_y \) denotes respectively the derivatives of \( N \) with respect to the spatial coordinates. The subscripts \( g, pg \) and \( dc \) in equations (19) and (20) 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 (8) for the advection-diffusive equation or in (17) 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,

\[
Ax = b,
\]

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:

\[
A = \sum_{e=1}^{nel} A^e,
\]

\[
b = \sum_{e=1}^{nel} b^e,
\]

where \( A \) 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 \( p_1, p_2, \ldots, p_N \in \mathbb{R}^N \) are called left conjugate gradient vectors of an \( N \times N \) real nonsingular matrix \( A \) if
\[ p_i^T A p_j = 0 \quad \text{for} \quad i < j, \]
\[ p_i^T A p_j \neq 0 \quad \text{for} \quad i = j. \]  
(24)

Suppose that the solution of the system (21) is \( x^* \), and \( \{p_1, p_2, \ldots, p_N\} \) are left conjugate gradient vectors of \( A \). Then it follows that
\[ x^* = x_0 + \sum_{i=1}^{N} \alpha_i p_i, \]  
(25)

for every fixed vector \( x_0 \). If \( r \) denotes the residual vector then
\[ r = r_0 - \sum_{i=1}^{N} \alpha_i A p_i, \]  
(26)

where \( r_0 \) is the initial residual vector. To determine \( \alpha_i \), since \( p_1, p_2, \ldots, p_N \) are linearly independent, then take \( r \) orthogonal to all \( p_i \), that is
\[ p_i^T r = 0 \quad \forall i = 1, \ldots, N. \]  
(27)

From (27) we obtain
\[ \alpha_i = \frac{p_i^T r_{i-1}}{p_i^T A p_i}. \]  
(28)

We also can write
\[ r_i = b - A x_i = r_{i-1} - \alpha_i A p_i, \]  
(29)
\[ x_i = x_0 + \sum_{k=1}^{i} \alpha_k p_k = x_{i-1} + \alpha_i p_i. \]  
(30)

From (28), (29) and (30) we can implement the left conjugate direction method if we know the set of linearly independent vectors \( p_1, p_2, \ldots, p_N \) such that they are left conjugate gradient vectors of \( A \). There is still a recurrence relation among \( p_1, p_2, \ldots, p_k \) and \( r_k \) to compute the left conjugate gradient vector \( p_{k+1} \), given in [4]:
\[ q_0 = r_k, \]
\[ \beta_i = \frac{p_i^T A q_{i-1}}{p_i^T A p_i}, \]
\[ q_i = q_{i-1} + \beta_i p_i \quad \text{for} \quad i = 1, \ldots, k, \]
\[ p_{k+1} = q_k. \]  
(31)

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 $p_1^TAp_1 \neq 0$ and $b$;
2. Calculate $r_0 = b - Ax_0$;
3. For $k = 1, \ldots, N$ do
   3.1 $q_k = A^T p_k$,
      $\alpha_k = \frac{p_k^Tr_{k-1}}{q_k^Tp_k}$,
      $x_k = x_{k-1} + \alpha_k p_k$,
      $r_k = b - Ax_k = r_{k-1} - \alpha_k Ap_k$;
   3.2 $p_{k+1} = r_k$,
      $\beta_i = \frac{q_i^Tp_k}{q_i^TAp_k}$,
      $p_{k+1} = p_{k+1} + \beta_i p_i$ for $i = 1, \ldots, k$.

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 $x_0$, $A$, $b$, $l_{\text{max}}$, $k$ and $\epsilon_{\text{tol}}$;
2. $r_0 = b - Ax_0$;
3. $\epsilon = \epsilon_{\text{tol}} ||r||$;
4. Choose $p_1$ such that $p_1^TAp_1 \neq 0$;
5. For $l = 1, \ldots, l_{\text{max}}$ do
   5.1. For $i = 1, \ldots, k$ do
      5.1.1. $q_i = A^T p_i$,
      $\alpha_i = \frac{p_i^Tr_{i-1}}{q_i^Tp_i}$,
      $x_i = x_{i-1} + \alpha_i p_i$,
      $r_i = r_{i-1} - \alpha_i Ap_i$;
   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 $j = 1, \ldots, i$ do
      $\beta_j = \frac{q_j^Tp_{i+1}}{q_j^Tp_i}$,
      $p_{i+1} = p_{i+1} + \beta_j p_j$,
   5.2. Choose the new $p_1$ such that $p_1^TAp_1 \neq 0$;

where $l_{\text{max}}$ is the maximum number of iterations, $\epsilon_{\text{tol}}$ 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 ($\{p_1, \ldots, p_k\}$ and $\{q_1, \ldots, q_k\}$).
2. For each iteration $l$ we need two matrix-vector products ($q_k = A^T p_k$ and $r_k = r_{k-1} - \alpha_k Ap_k$ - 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 $p_1 = \text{diag}(A)$. 

Copyright © 2003 John Wiley & Sons, Ltd.
Prepared using nmeauth.cls
4. In every restart $l$ there is the need to choose a new $p^l_1$ (step 5.2 in Algorithm 3.2). The options here are: $p^l_1 = p^{l-1}_k$, $p^l_1 = r^{l-1}$ or $p^l_1 = x^{l-1}$.

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 (21) in scaled form,

$$\tilde{A}\tilde{x} = \tilde{b},$$

(32)

$$\tilde{A} = W^{-1/2}AW^{-1/2}, \quad \tilde{x} = W^{1/2}x, \quad \tilde{b} = W^{1/2}b \quad \text{and} \quad W = diag(A).$$

Thus we define the preconditioned system of equations to be solved as,

$$\tilde{A}\tilde{x} = \tilde{b},$$

(33)

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

$$L = \bigoplus_{e=1}^{n_{el}} L_e,$$

(34)

$$U = \bigoplus_{e=1}^{n_{el}} U_e,$$

(35)

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

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

(36)

For the Euler equation a nodal block-diagonal preconditioner is used. In this case, $\tilde{A} = BD^{-1}A$, $\tilde{b} = BD^{-1}b$, $BD = \bigoplus_{ino=1}^{n_{nodes}} BD_{ino}$ and $BD_{ino}$ is the $4 \times 4$ 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
\[ u = 0.0 \quad \text{along} \quad y = 0.0, \]
\[ u = 0.0 \quad \text{along} \quad x = 0.0 \quad \text{and} \quad 0.0 < y < 0.25, \]
\[ u = 1.0 \quad \text{along} \quad x = 0.0 \quad \text{and} \quad 0.25 < y < 1.0. \]  \hfill (37)

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

The diffusivity is \( \kappa_x = \kappa_y = 1 \times 10^{-7} \), the flow direction is \( 45^\circ \) from the \( x \)-axis, \( \| \beta \| = 1 \) and the stabilization parameter is computed as in [8]. The domain is discretized by grids of triangular elements with \( 64 \times 64, 128 \times 128, 256 \times 256 \) and \( 512 \times 512 \) 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 \( 64 \times 64 \) cells. We observed that both solutions are virtually identical. The solution for other numbers of left conjugate gradient vectors is the same.

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 \( p_l^1 = p_{k+1}^{l-1} \) is the best 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 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 6 shows the performance of GMRES and LCD methods for \( k=1, 5, 10 \) and 20 considering the mesh of \( 128 \times 128 \) 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.
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

<table>
<thead>
<tr>
<th>Cells</th>
<th>Neq</th>
<th>GMRES(1)</th>
<th>LCD(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 x 64</td>
<td>8192</td>
<td>714</td>
<td>14</td>
</tr>
<tr>
<td>128 x 128</td>
<td>32768</td>
<td>1123</td>
<td>114</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mesh</th>
<th>GMRES(5)</th>
<th>LCD(5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cells</td>
<td>Neq</td>
<td>Niter</td>
</tr>
<tr>
<td>64 x 64</td>
<td>8192</td>
<td>471</td>
</tr>
<tr>
<td>128 x 128</td>
<td>32768</td>
<td>888</td>
</tr>
<tr>
<td>256 x 256</td>
<td>131072</td>
<td>1661</td>
</tr>
<tr>
<td>512 x 512</td>
<td>524288</td>
<td>3104</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mesh</th>
<th>GMRES(10)</th>
<th>LCD(10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cells</td>
<td>Neq</td>
<td>Niter</td>
</tr>
<tr>
<td>64 x 64</td>
<td>8192</td>
<td>399</td>
</tr>
<tr>
<td>128 x 128</td>
<td>32768</td>
<td>751</td>
</tr>
<tr>
<td>256 x 256</td>
<td>131072</td>
<td>1479</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mesh</th>
<th>GMRES(20)</th>
<th>LCD(20)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cells</td>
<td>Neq</td>
<td>Niter</td>
</tr>
<tr>
<td>64 x 64</td>
<td>8192</td>
<td>448</td>
</tr>
<tr>
<td>128 x 128</td>
<td>32768</td>
<td>756</td>
</tr>
<tr>
<td>256 x 256</td>
<td>131072</td>
<td>1383</td>
</tr>
</tbody>
</table>

without restart

<table>
<thead>
<tr>
<th>Mesh</th>
<th>GMRES</th>
<th>LCD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cells</td>
<td>Neq</td>
<td>Niter</td>
</tr>
<tr>
<td>64 x 64</td>
<td>8192</td>
<td>261</td>
</tr>
<tr>
<td>128 x 128</td>
<td>32768</td>
<td>533</td>
</tr>
</tbody>
</table>

### Table II. Memory requirements for the mesh of 128 x 128 cells

<table>
<thead>
<tr>
<th>Number of vectors</th>
<th>GMRES (Mwords)</th>
<th>LCD (Mwords)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.57</td>
<td>1.77</td>
</tr>
<tr>
<td>5</td>
<td>1.84</td>
<td>2.36</td>
</tr>
<tr>
<td>10</td>
<td>2.16</td>
<td>2.95</td>
</tr>
<tr>
<td>20</td>
<td>2.82</td>
<td>4.26</td>
</tr>
</tbody>
</table>

without restart

<table>
<thead>
<tr>
<th>Number of vectors</th>
<th>GMRES (Mwords)</th>
<th>LCD (Mwords)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>41.56</td>
<td>80.29</td>
</tr>
</tbody>
</table>
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.

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

<table>
<thead>
<tr>
<th>operation</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Ap) in LCD</td>
<td>0.052</td>
</tr>
<tr>
<td>(Ap) in LCD or GMRES</td>
<td>0.050</td>
</tr>
</tbody>
</table>

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.

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Niter</th>
<th>CPU Time(sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCD(5)</td>
<td>247</td>
<td>48</td>
</tr>
<tr>
<td>GMRES(5)</td>
<td>288</td>
<td>34</td>
</tr>
<tr>
<td>Bi-CGSTAB</td>
<td>149</td>
<td>41</td>
</tr>
<tr>
<td>TFQMR</td>
<td>180</td>
<td>34</td>
</tr>
</tbody>
</table>
Figure 5. Pure convection of a scalar on a square domain - Relative residual evolution for LCD(5) and GMRES(5).
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.

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.
4.2. Compressible Flow - Oblique Shock

This problem consists of a two-dimensional steady problem of an inviscid, Mach 2, uniform flow, over a wedge at an angle of $-10^\circ$ with respect to a horizontal wall, resulting in the occurrence of an oblique shock with an angle of $29.3^\circ$ emanating from the leading edge of the wedge, as shown in Figure 8. Multi-corrections are fixed to 3 and we considered a fixed time step of 0.01.

![Figure 8. Oblique shock - problem description.](image)

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:

\[
\begin{align*}
\text{Inflow} & : & \\
M & = 2.0 & M & = 1.64052 \\
\rho & = 1.0 & \rho & = 1.45843 \\
u & = \cos10^\circ & u & = 0.88731 \\
v & = -\sin10^\circ & v & = 0.0 \\
p & = 0.17857 & p & = 0.30475 \\
\end{align*}
\]

where $M$ is the Mach number, $\rho$ is the flow density, $u$ and $v$ 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 $v = 0$ is set at the bottom boundary; and no boundary conditions are imposed on the outflow (right) boundary. A $20 \times 20$ 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 ($N_{\text{steps}}$), the total number of iterations ($N_{\text{iter}}$) 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.

![Density profile at $x = 0.9$.](image)

![Evolution of density residual.](image)

Figure 9. Oblique shock - Solution with both methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>$N_{\text{steps}}$</th>
<th>$N_{\text{iter}}$</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCD(5)</td>
<td>870</td>
<td>5285</td>
<td>134</td>
</tr>
<tr>
<td>GMRES(5)</td>
<td>869</td>
<td>5226</td>
<td>117</td>
</tr>
<tr>
<td>Bi-CGSTAB</td>
<td>746</td>
<td>3520</td>
<td>128</td>
</tr>
<tr>
<td>TFQMR</td>
<td>843</td>
<td>4296</td>
<td>142</td>
</tr>
</tbody>
</table>

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):

\[
\begin{align*}
R1 & \quad M = 2.9 & & R2 & \quad u = 2.61934 & & R3 & \quad u = 2.40140 \\
\rho &= 1.0 & & \rho &= 1.7 & & \rho &= 2.68728 \\
u &= 2.9 & & v &= -0.50632 & & v &= 0.0 \\
p &= 0.714286 & & p &= 1.52819 & & p &= 2.93407 \\
\end{align*}
\]

(39)

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 x 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 (\(N_{\text{steps}}\)) 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. Reflected shock - Computational costs - Structured mesh.

<table>
<thead>
<tr>
<th>Method</th>
<th>(N_{\text{steps}})</th>
<th>(N_{\text{iter}})</th>
<th>Time(sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCD(5)</td>
<td>618</td>
<td>4159</td>
<td>332</td>
</tr>
<tr>
<td>GMRES(5)</td>
<td>612</td>
<td>4977</td>
<td>276</td>
</tr>
<tr>
<td>Bi-CGSTAB</td>
<td>610</td>
<td>1905</td>
<td>279</td>
</tr>
<tr>
<td>TFQMR</td>
<td>589</td>
<td>2603</td>
<td>298</td>
</tr>
</tbody>
</table>

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. Reflected shock - Computational costs - Unstructured mesh.

<table>
<thead>
<tr>
<th>Method</th>
<th>Nsteps</th>
<th>Niter</th>
<th>Time(sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCD(5)</td>
<td>838</td>
<td>13498</td>
<td>1222</td>
</tr>
<tr>
<td>GMRES(5)</td>
<td>836</td>
<td>9573</td>
<td>730</td>
</tr>
<tr>
<td>Bi-CGSTAB</td>
<td>863</td>
<td>7930</td>
<td>1025</td>
</tr>
<tr>
<td>TFQMR</td>
<td>798</td>
<td>10286</td>
<td>1000</td>
</tr>
</tbody>
</table>

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 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.

Recently, Dai and Yuan [19] proposed a new technique to overcome the breakdown problem
appearing in the semi-conjugate direction method and a memory limitation scheme similar to
the limited-memory BFGS method to minimize memory requirements of the original algorithm
[4]. Those results should be viewed as a first attempt to incorporate LCD into the methods
available for solving finite element 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

2. F. Shakib, T. J. R. Hughes, Z. Johan, A multi-element group preconditioned gmres algorithm for
3. T. E. Tezduyar, J. Liou, Grouped element-by-element iteration schemes for incompressible flow simulations,
4. J. Yuan, G. Golub, R. Flemons, W. A. G. Cecilio, Semiconjugate direction methods for real positive
definite systems, BIT (http://www-sccm.stanford.edu/urap/pub-tech.html/SxCCM-02-02) (accepted).
5. R. S. Silva, F. M. Raupp, R. C. Almeida, A numerical methodology to solve thermal pollution problems,
in: 9th Brazilian Congress of Thermal Engineering and Sciences CD-ROM, Caxambu, Minas Gerais, Brazil,
2002, pp. 1–12.
6. 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
8. A. Brooks, T. Hughes, Streamline Upwind/Petrov-Galerkin formulations for convection dominated flows
9. L. Franca, S. Frey, T. Hughes, Stabilized finite element methods: Application to the advective-diffusive
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


