Preconditioned GMRES and CGNR Methods for the Convection-Diffusion Equation Using a Finite Difference Scheme

Otmane Souhar

Laboratory of Computer Science, Mathematics and Applications
Faculty of Sciences, University of Chouaib Doukkali, EL Jadida, Morocco

Said El Hajji

Laboratory of Mathematics, Computing and Applications
Faculty of Sciences, University of Mohammed V-Agdal, Rabat, Morocco

Copyright © 2014 Otmane Souhar and Said El Hajji. This article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Abstract

Combinations of four preconditioners with two Krylov subspace methods for the solution of nonsymmetric sparse linear systems are investigated in this paper. First, we have considered implicit methods applied to the original matrix and to the matrix associated to normal equations which lead to incomplete factorizations followed by an approximate inverse. Second, we have used an approximate inverse method based on Frobenius norm minimization. We finally have considered factorized sparse approximate inverses of the original matrix and of the matrix associated to normal equations. These preconditioners were used for the GMRES and CGNR iterative solution methods. Comparative results of numerical tests on linear systems, provided from finite difference scheme of a convection-diffusion problem, were presented.

Mathematics Subject Classification: 65F10, 65F35, 65F50, 65N06

Keywords: Sparse linear systems, Preconditioned Krylov subspace methods, Approximately inverse techniques
1 Introduction

Mathematical modeling in science and engineering mostly leads to the solution of system of linear equations

\[ Au = b, \tag{1} \]

where \( A \) is a real, nonsymmetric, sparse and large matrix of order \( n \). In this work, we are concerned with the application of different preconditioners for the two Krylov subspace GMRES and CGNR methods. Preconditioned conjugate gradient (CG) methods applied to symmetric positive definite (SPD) matrices have good theoretical and numerical results [1], these lead to various techniques which consist to convert a nonsymmetric linear system into a symmetric one. The well known technique is to solve the equivalent normal SPD system

\[ A^T A u = A^T b, \tag{2} \]

Convergence rate of Krylov subspace methods for solving (1) is strongly influenced by the spectral properties of \( A \). Therefore, we transform the original system into another that have the same solution but with favorable spectral properties. The following left and right preconditioned systems could be solved:

\[ MAu = Mb \quad \text{and/or} \quad AMx = b, \quad u = Mx. \tag{3} \]

Note that these two systems are nonsymmetric when the system (1) is symmetric. To preserve the symmetry a simple strategy is possible; it consists in using both forms of preconditioning at once. e.g., when \( M \) is available in the form of an Incomplete Cholesky factorization (IC), \( M = LL^T \), and the following symmetric transformation leads to the SPD system.

\[ L^{-1} A L^{-T} x = L^{-1} b; u = L^{-T} x. \tag{4} \]

Construction of preconditioners for sparse linear systems is a serious task in numerical analysis. One of the simplest ways is to perform an incomplete factorization of the original matrix.

\[ A = LU + E, \tag{5} \]

where \( L \) and \( U \) have, respectively, the same nonzero structure as the lower and upper parts of \( A \), and \( E \) is the error matrix. The preconditioner provided from this incomplete factorization, known as ILU(0), is rather easy and inexpensive to compute; however, it is not robust and fails for many real-life problems, furthermore the degree of parallelism is very limited. Some extensions of this factorization which increase its accuracy and robustness (degree of parallelism), have been considered [8], [9], [12], [14], [15], [17], [20], as well as other works.
Another way of finding approximate inverse is based on minimizing the Frobenius norm of the residual matrix \((I - AM)\). Consider the minimization of

\[
F(M) = \|I - AM\|_F^2 = \sum_{j=1}^{n} \|e_j - Am_j\|_2^2,
\]

where \(e_j\) and \(m_j\) are, respectively, the \(jth\) columns of the identity and \(M\) matrices. Thus, minimizing is equivalent to minimizing the individual functions

\[
f_j(m) = \|e_j - Am\|_2 \quad j = 1, \ldots, n.
\]

This is clearly useful for parallel implementations [6], [7], [10], [11], [13].

A last way of finding preconditioners is based on incomplete inverse factorizations, i.e., on an incomplete factorization of \(A^{-1}\). If \(A\) admits the factorization \(A = LDU\) where \(L\) is unit lower triangular, \(D\) is diagonal, and \(U\) is unit upper triangular, then \(A^{-1}\) can be factorized as \(A^{-1} = U^{-1}D^{-1}L^{-1} = ZD^{-1}W^T\) where \(Z = U^{-1}\) and \(W = L^{-T}\) are unit upper triangular matrices. Factorized sparse approximate inverse preconditioners can be constructed by computing sparse approximations \(\bar{Z} \approx Z\) and \(\bar{W} \approx W\). The factorized approximate inverse is then

\[
M = \bar{Z} \bar{D}^{-1} \bar{W}^T \approx A^{-1},
\]

where \(\bar{D}^{-1}\) is a nonsingular diagonal matrix, \(\bar{D} \approx D\). Interested reader is invited to [3].

Finally, To develop preconditioners for the normal equations by using an incomplete factorization of the inverse. Specifically, we have used the stabilized approximate inverse (SAINV) developed by Benzi et al. in the case of symmetric systems [2].

This work is organized as follows. In Section 2, an overview and background on approximate inverse techniques, used herein, to construct preconditioners are provided. In Section 3, we detail our numerical experiments and compare results obtained by applying different combinations of preconditioners and iterative solution methods. Conclusions are drawn up in the last section.

## 2 Approximate inverse methods

We give, in this section, an overview of some approximate inverse techniques which were used in our numerical experiments. The interested reader is referred, for a general overview on the given classes of preconditioner, to [4] or to the original papers.
2.1 Inverse ILU technique: ILUT

Computing via Gaussian elimination an approximate of the exact LU factorization of the matrix $A$ is the subject of several papers and books. In this context, Saad has developed an ILUT($\tau; p$) dual threshold incomplete factorization using two dropping parameters, $\tau$ and $p$, to control the storage cost [17]. Its implementation is based on the IKJ variant of Gaussian elimination [18]. In Algorithm 1, $a_{i,\beta}$ and $u_{k,\beta}$ denote the $i$th and $k$th rows of $A$ and $U$, respectively.

**Algorithm 1. STANDARD ILUT($\tau, p$) factorization.**

1. For $i = 2, \ldots, n$
2. $w = a_{i,\beta}$
3. For $k = 1, \ldots, i - 1$ and when $w \neq 0$
4. $w_k = w_k/a_{k,k}$
5. If $|w_k| < \tau \text{nzavg}(a_{i,\beta})$, set $w_k = 0$
6. If $w \neq 0$ then
7. $w = w - w_k u_{k,\beta}$
8. Endif
9. End($k$-loop)
10. Apply a dropping strategy to row $w$
11. Set $l_{i,j} = w_j$, for $j = 1, \ldots, i - 1$ whenever $w \neq 0$
12. Set $u_{i,j} = w_j$, for $j = i, \ldots, n$ whenever $w \neq 0$
13. Set $w = 0$
14. End($i$-loop).

In line 5, the function $\text{nzavg}(a_{i,\beta})$ returns the average magnitude of the nonzero elements of a given sparse row. Elements with relatively small magnitude are dropped. In line 10, a different dropping strategy is applied. First, small elements are dropped according to the relative magnitude similar to the criterion used in line 5. Then a sorting operation is performed and only the largest $p$ elements in absolute value of the $L$ and $U$ factors are kept.

In the SPD case, the equivalent drop tolerance based on the incomplete factorization IC(tol) is used for our numerical experiments to solve equation (2).

2.2 Method based on Frobenius norm minimization: SPAI

The SPAI algorithm [11] computes a sparse approximate inverse $M$ by minimizing $\|I - AM\|_F$ in the Frobenius norm. The resulting method is easy to parallelize and the sparsity pattern of $M$ is not imposed, a priori, but captured automatically.

We follow [12] in the sequel of this subsection. The nonzero pattern is a subset $G = \{(i, j) / 1 \leq i, j \leq n \}$. Denote by $m_j$ the $j$th column of $M$ ($1 \leq j \leq n$).
For a fixed $j$, consider the set $\mathcal{J} = \{ i \mid (i,j) \in \mathcal{G} \}$, which specifies the nonzero pattern of $m_j$. The only columns of $A$ that enter the definition of $m_j$ are those whose index is in $\mathcal{J}$. Let $A(:,\mathcal{J})$ be the submatrix of $A$ formed from such columns, and let $\mathcal{I}$ be the set of indices of nonzero rows of $A(:,\mathcal{J})$. Then we can restrict our attention to the matrix $\hat{A} = A(\mathcal{I},\mathcal{J})$, to the unknown vector $\hat{m}_j = m_j(\mathcal{J})$ and to the right hand side $\hat{e}_j = e_j(\mathcal{J})$. The nonzero entries in $m_j$ can be computed by solving the (small) unconstrained least squares problem $\| \hat{e}_j - \hat{A}\hat{m}_j \|_2$. This least squares problem can be solved, for instance, by means of the QR factorization of $\hat{A}$. Each column $m_j$ can be computed independently of the other columns of $M$. Note that due to the sparsity of $A$, the submatrix $\hat{A}$ will contain only a few nonzero rows and columns, so each least squares problem has small size and can be solved efficiently by dense matrix techniques. But for matrices with a general sparsity pattern it is difficult to prescribe a good nonzero pattern for $M$. Several authors have developed adaptive strategies which start with a simple initial guess and successively augment this pattern until a criterion of type $\| e_j - Am_j \|_2 < \varepsilon$ is satisfied for a given $\varepsilon > 0$ or a maximum number of nonzeros in $m_j$ has been reached. Based on this technique, the following algorithm was developed by Grote and Huckle [11].

Algorithm 2. SPAI ALGORITHM.

For every column $m_j$ of $M$.
1. Choose an initial sparsity $\mathcal{J}$.
2. Determine the row indices $\mathcal{I}$ of the corresponding nonzero entries and the QR decomposition of $\hat{A} = A(\mathcal{I},\mathcal{J})$. Then compute the solution $\hat{m}_j$ of the least squares problem $\| \hat{e}_j - \hat{A}\hat{m}_j \|_2$ and its residual given by $r = \hat{e}_j - \hat{A}\hat{m}_j$. While $\| r \|_2 > \varepsilon$:
3. Set $\mathcal{L}$ equal to the set of indices $l$ for which $r(l) \neq 0$.
4. Set $\tilde{\mathcal{J}}$ equal to the set of all new column indices of $A$ that appear in all $\mathcal{L}$ rows but not in $\mathcal{J}$.
5. For each $k \in \tilde{\mathcal{J}}$ compute the norm $\rho_k$ of the new residual via the formula $\rho_k^2 = \| r \|_2^2 - \frac{(r^T A e_k)^2}{\| A e_k \|_2^2}$ and delete from $\tilde{\mathcal{J}}$ all but the most profitable indices.
6. Determine the new indices $\hat{\mathcal{I}}$ and update the QR factorization of the submatrix $A(\mathcal{I} \cup \hat{\mathcal{I}}, \mathcal{J} \cup \tilde{\mathcal{J}})$. Then solve the new least squares problem, compute the new residual $r = e_j - Am_j$, and set $\mathcal{I} = \mathcal{I} \cup \hat{\mathcal{I}}$ and $\mathcal{J} = \mathcal{J} \cup \tilde{\mathcal{J}}$.

This algorithm requires several parameters, for example, the initial sparsity structure of $M$ and the bound on the number of nonzeros allowed in each column of $M$. 
2.3 Factorized sparse approximate inverse: AINV

The AINV algorithm is based on an algorithm which computes two sets of vectors \((z_i)_{1 \leq i \leq n}\) and \((w_i)_{1 \leq i \leq n}\) directly from \(A\) by means of \(A\)-biconjugate process applied to the unit basis vectors, i.e., \(w_j^T A z_j = 0\) if and only if \(i \neq j\). In order to describe the procedure, let \(a_i^T\) and \(c_i^T\) denote, respectively, the \(i\)th row of \(A\) and \(A^T\).

**Algorithm 3. Biconjugation algorithm.**

1. Let \(w_i^{(0)} = z_i^{(0)} = e_i\) \((1 \leq i \leq n)\).
2. For \(i = 1, 2, ..., n\) do
3. For \(j = i, i + 1, ..., n\) do
4. \(p_j^{(i-1)} = a_i^T z_j^{(i-1)}; \quad q_j^{(i-1)} = c_i^T w_j^{(i-1)}\)
5. End(j-loop)
6. If \(i = n\) go to 11.
7. For \(j = i + 1, ..., n\) do
8. \(z_j^{(i)} = z_j^{(i-1)} - (p_j^{(i-1)}/p_i^{(i-1)})z_i^{(i-1)}; \quad w_j^{(i)} = w_j^{(i-1)} - (q_j^{(i-1)}/q_i^{(i-1)})w_i^{(i-1)}\)
9. End(j-loop)
10. End(i-loop)
11. Let \(z_i = z_i^{(i-1)}, w_i = w_i^{(i-1)}\) and \(p_i = p_i^{(i-1)}, \text{ for } 1 \leq i \leq n\).

Return \(Z = [z_1, ..., z_n], W = [w_1, ..., w_n], D = \text{diag}(p_1, p_2, ..., p_n)\).

The incomplete biconjugate algorithm computes sparse unit upper triangular matrices \(\bar{Z} \approx Z, \bar{W} \approx W\), and a nonsingular diagonal matrix \(\bar{D} \approx D\) such that \(M = \bar{Z} \bar{D}^{-1} \bar{W}^T \approx A^{-1}\) is a factorized sparse approximate inverse of \(A\). Sparsity in \(\bar{Z}\) and \(\bar{W}\) factors is preserved by removing small fill-in in the \(z\) and \(w\) vectors.

For a SPD matrix \(B = AA^T\), the AINV does not breakdown, and the procedure can be carried out by computing only the matrix \(Z\). For this class of matrices Benzi et al. [2] have been developed a variant of the AINV algorithm, referred to SAINV, which is based on a reformulation of Algorithm 3.

3 Numerical results

Our primary goal of these numerical experiments is to compare and evaluate the advantages and disadvantages of combinations of iterative methods and preconditioners described above. We consider then the convection-diffusion equation who plays a very important role in computational fluid dynamics to simulate flow problems,

\[-\Delta u + Re(p(x,y)\frac{\partial u}{\partial x} + q(x,y)\frac{\partial u}{\partial y}) = f \in \Omega,\]  \hspace{1cm} (9)
with a homogeneous Neumann boundary condition
\[ \frac{\partial u}{\partial \nu} = 0 \text{ on } \partial \Omega, \]  
(10)

where \( \partial \Omega \) is the boundary of a smooth convex domain \( \Omega = [0, 1] \times [0, 1] \text{ in } \mathbb{R}^2 \); \( \nu \) is the outer normal to \( \Omega \); and \( Re \) is a dimensional Reynolds number which gives the ratio of inertial to viscous forces. When \( Re \) is small, viscous forces predominate so that the flow tends to remain laminar but when the ratio increases beyond a critical value \( Re_c \), inertial forces take charge and the flow becomes turbulent. Furthermore, for a large \( Re \), e.g., \( Re > 10^3 \), the equation (9) is said to be convection-dominated, otherwise it is diffusion-dominated. In the following, we study the two frequently used problems:

\[ (P1) \]
\[ p(x, y) = x(x - 1)(1 - 2y), \]
\[ q(x, y) = -y(y - 1)(1 - 2x). \]

\[ (P2) \]
\[ p(x, y) = (1 - x^2)(2y - 1), \]
\[ q(x, y) = 2xy(y - 1). \]

These two problems were proposed by Ruge and Stuben to test their algebraic multigrid method [16]. In these examples, we point out the existence of a stagnation point at \((0.5, 0.5)\) for the first problem, and a semi-recirculation effect for the second problem because one of its convection coefficients \( p(x, y) \) vanishes at \( y = 0.5 \). Directions and magnitudes of the convective flow are illustrated on Figure 1.

Equation (9) may be discretized using various ways. In the context of finite differences, the most familiar schemes are the central differences and the upwind differences. The discretization of equation (9) by use of such schemes lead to sparse linear systems of the form (1). Furthermore, standard central finite differences approximation of the convection-diffusion equation produces a stable discretization only if
\[ h \leq \frac{1}{Re \times \sup_{(x, y)} \|(p, q)\|_\infty}, \]
(11)

where \( h \) is an uniform meshsize; in which case \( A \) is an M-matrix. In the case when condition (11) is not met, spurious oscillations appear in the discrete solution. To overcome this problem the forward and backward upwind finite difference operators may be used. The truncation error of the standard five-point central difference scheme for equation (9) is obviously of \( O(h^2) \) but this truncation is only of \( O(h) \) for the five point upwind finite difference scheme.

Due to the size of \( A \), direct solvers are prohibitively expensive because of the amount of work and storage required. As an alternative we consider
Table 1: Number of iterations using gmres(20) algorithm with $M_T$, $M_S$ and $M_F$ preconditioning.
Preconditioned GMRES and CGNR methods for the convection-diffusion

<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>$M_T$ Preconditioner</th>
<th></th>
<th></th>
<th>$M_S$ Preconditioner</th>
<th></th>
<th></th>
<th>$M_F$ Preconditioner</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(P1) 64</td>
<td>(P2) 64</td>
<td>(P1) 128</td>
<td>(P2) 128</td>
<td>(P1) 64</td>
<td>(P2) 64</td>
<td>(P1) 128</td>
<td>(P2) 128</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.54 0.11 2.66</td>
<td>0.51 0.12 2.75</td>
<td>4.81 0.34 2.66</td>
<td>6.98 0.34 2.66</td>
<td>4.61 0.36 2.77</td>
<td>1.27 0.11 2.63</td>
<td>0.65 0.12 2.68</td>
<td>3.35 0.33 2.66</td>
<td>7.10 0.36 2.75</td>
</tr>
<tr>
<td>$10^1$</td>
<td>0.76 0.11 2.65</td>
<td>0.76 0.12 2.68</td>
<td>6.98 0.34 2.66</td>
<td>6.24 0.36 2.77</td>
<td>4.61 0.36 2.77</td>
<td>0.47 0.10 2.53</td>
<td>0.26 0.11 2.57</td>
<td>1.84 0.32 2.58</td>
<td>1.34 0.33 2.60</td>
</tr>
<tr>
<td>$10^2$</td>
<td>1.27 0.11 2.63</td>
<td>0.65 0.12 2.68</td>
<td>3.35 0.33 2.66</td>
<td>7.10 0.36 2.75</td>
<td>4.81 0.34 2.66</td>
<td>1.65 0.09 2.24</td>
<td>1.09 0.09 2.26</td>
<td>6.70 0.27 2.19</td>
<td>6.18 0.30 2.28</td>
</tr>
<tr>
<td>$10^3$</td>
<td>0.47 0.10 2.53</td>
<td>0.26 0.11 2.57</td>
<td>1.84 0.32 2.58</td>
<td>1.34 0.33 2.60</td>
<td>2.66 0.09 2.13</td>
<td>1.76 0.09 2.16</td>
<td>5.78 0.24 1.85</td>
<td>4.11 0.26 1.91</td>
<td></td>
</tr>
<tr>
<td>$10^4$</td>
<td>1.27 0.11 2.63</td>
<td>0.65 0.12 2.68</td>
<td>3.35 0.33 2.66</td>
<td>7.10 0.36 2.75</td>
<td>4.81 0.34 2.66</td>
<td>1.65 0.09 2.24</td>
<td>1.09 0.09 2.26</td>
<td>6.70 0.27 2.19</td>
<td>6.18 0.30 2.28</td>
</tr>
<tr>
<td>$10^5$</td>
<td>0.47 0.10 2.53</td>
<td>0.26 0.11 2.57</td>
<td>1.84 0.32 2.58</td>
<td>1.34 0.33 2.60</td>
<td>2.66 0.09 2.13</td>
<td>1.76 0.09 2.16</td>
<td>5.78 0.24 1.85</td>
<td>4.11 0.26 1.91</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: CPU-time performance and memory cost comparisons of $M_T$, $M_S$ and $M_F$ preconditioners using gmres(20) algorithm.
<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>$16$</th>
<th>$32$</th>
<th>$64$</th>
<th>$128$</th>
<th>$16$</th>
<th>$32$</th>
<th>$64$</th>
<th>$128$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_N$ Preconditioner</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$(P1)$</td>
<td>$(P2)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Re$</td>
<td>$(P1)$</td>
<td>$(P2)$</td>
<td>$(P1)$</td>
<td>$(P2)$</td>
<td>$(P1)$</td>
<td>$(P2)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$1$</td>
<td>$6$</td>
<td>$7$</td>
<td>$16$</td>
<td>$18$</td>
<td>$56$</td>
<td>$48$</td>
<td>$137$</td>
<td>$124$</td>
</tr>
<tr>
<td>$10^1$</td>
<td>$6$</td>
<td>$7$</td>
<td>$17$</td>
<td>$19$</td>
<td>$66$</td>
<td>$56$</td>
<td>$142$</td>
<td>$112$</td>
</tr>
<tr>
<td>$10^2$</td>
<td>$6$</td>
<td>$9$</td>
<td>$15$</td>
<td>$17$</td>
<td>$143$</td>
<td>$161$</td>
<td>$121$</td>
<td>$167$</td>
</tr>
<tr>
<td>$10^3$</td>
<td>$8$</td>
<td>$9$</td>
<td>$12$</td>
<td>$15$</td>
<td>$80$</td>
<td>$93$</td>
<td>$96$</td>
<td>$117$</td>
</tr>
<tr>
<td>$10^4$</td>
<td>$10$</td>
<td>$6$</td>
<td>$12$</td>
<td>$16$</td>
<td>$116$</td>
<td>$169$</td>
<td>$215$</td>
<td>$210$</td>
</tr>
<tr>
<td>$10^5$</td>
<td>$10$</td>
<td>$7$</td>
<td>$13$</td>
<td>$12$</td>
<td>$101$</td>
<td>$127$</td>
<td>$201$</td>
<td>$142$</td>
</tr>
<tr>
<td>$M_C$ Preconditioner</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$(P1)$</td>
<td>$(P2)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Re$</td>
<td>$(P1)$</td>
<td>$(P2)$</td>
<td>$(P1)$</td>
<td>$(P2)$</td>
<td>$(P1)$</td>
<td>$(P2)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$1$</td>
<td>$5$</td>
<td>$5$</td>
<td>$10$</td>
<td>$10$</td>
<td>$36$</td>
<td>$32$</td>
<td>$111$</td>
<td>$95$</td>
</tr>
<tr>
<td>$10^1$</td>
<td>$5$</td>
<td>$5$</td>
<td>$9$</td>
<td>$11$</td>
<td>$52$</td>
<td>$40$</td>
<td>$120$</td>
<td>$103$</td>
</tr>
<tr>
<td>$10^2$</td>
<td>$6$</td>
<td>$6$</td>
<td>$9$</td>
<td>$11$</td>
<td>$80$</td>
<td>$37$</td>
<td>$80$</td>
<td>$117$</td>
</tr>
<tr>
<td>$10^3$</td>
<td>$6$</td>
<td>$5$</td>
<td>$11$</td>
<td>$8$</td>
<td>$36$</td>
<td>$26$</td>
<td>$87$</td>
<td>$61$</td>
</tr>
<tr>
<td>$10^4$</td>
<td>$7$</td>
<td>$6$</td>
<td>$10$</td>
<td>$11$</td>
<td>$50$</td>
<td>$50$</td>
<td>$107$</td>
<td>$98$</td>
</tr>
<tr>
<td>$10^5$</td>
<td>$6$</td>
<td>$6$</td>
<td>$12$</td>
<td>$12$</td>
<td>$40$</td>
<td>$70$</td>
<td>$117$</td>
<td>$80$</td>
</tr>
</tbody>
</table>

Table 3: Number of iterations using CGNR algorithm with $M_N$ and $M_C$ preconditioning.

<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>$64$</th>
<th>$128$</th>
<th>$64$</th>
<th>$128$</th>
<th>$64$</th>
<th>$128$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_N$ Preconditioner</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$(P1)$</td>
<td>$(P2)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Re$</td>
<td>$(P1)$</td>
<td>$(P2)$</td>
<td>$(P1)$</td>
<td>$(P2)$</td>
<td>$(P1)$</td>
<td>$(P2)$</td>
</tr>
<tr>
<td>$1$</td>
<td>$0.50$</td>
<td>$1.89$</td>
<td>$4.12$</td>
<td>$1.76$</td>
<td>$2.77$</td>
<td>$6.92$</td>
</tr>
<tr>
<td>$10$</td>
<td>$0.59$</td>
<td>$1.88$</td>
<td>$4.16$</td>
<td>$1.80$</td>
<td>$2.71$</td>
<td>$6.77$</td>
</tr>
<tr>
<td>$10^2$</td>
<td>$1.29$</td>
<td>$1.80$</td>
<td>$4.01$</td>
<td>$1.52$</td>
<td>$2.61$</td>
<td>$6.68$</td>
</tr>
<tr>
<td>$10^3$</td>
<td>$0.72$</td>
<td>$1.79$</td>
<td>$3.89$</td>
<td>$1.17$</td>
<td>$2.57$</td>
<td>$6.45$</td>
</tr>
<tr>
<td>$10^4$</td>
<td>$1.04$</td>
<td>$1.68$</td>
<td>$3.72$</td>
<td>$2.52$</td>
<td>$2.51$</td>
<td>$6.22$</td>
</tr>
<tr>
<td>$10^5$</td>
<td>$0.91$</td>
<td>$1.65$</td>
<td>$3.61$</td>
<td>$2.46$</td>
<td>$2.42$</td>
<td>$5.85$</td>
</tr>
<tr>
<td>$M_C$ Preconditioner</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$(P1)$</td>
<td>$(P2)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Re$</td>
<td>$(P1)$</td>
<td>$(P2)$</td>
<td>$(P1)$</td>
<td>$(P2)$</td>
<td>$(P1)$</td>
<td>$(P2)$</td>
</tr>
<tr>
<td>$1$</td>
<td>$2.52$</td>
<td>$0.76$</td>
<td>$3.22$</td>
<td>$2.31$</td>
<td>$0.79$</td>
<td>$3.46$</td>
</tr>
<tr>
<td>$10$</td>
<td>$3.64$</td>
<td>$0.76$</td>
<td>$3.16$</td>
<td>$2.89$</td>
<td>$0.76$</td>
<td>$3.33$</td>
</tr>
<tr>
<td>$10^2$</td>
<td>$5.60$</td>
<td>$0.72$</td>
<td>$3.07$</td>
<td>$2.67$</td>
<td>$0.74$</td>
<td>$3.21$</td>
</tr>
<tr>
<td>$10^3$</td>
<td>$2.50$</td>
<td>$0.70$</td>
<td>$2.87$</td>
<td>$1.88$</td>
<td>$0.69$</td>
<td>$3.07$</td>
</tr>
<tr>
<td>$10^4$</td>
<td>$3.50$</td>
<td>$0.62$</td>
<td>$2.74$</td>
<td>$3.61$</td>
<td>$0.65$</td>
<td>$2.95$</td>
</tr>
<tr>
<td>$10^5$</td>
<td>$2.80$</td>
<td>$0.56$</td>
<td>$2.55$</td>
<td>$5.05$</td>
<td>$0.60$</td>
<td>$2.73$</td>
</tr>
</tbody>
</table>

Table 4: CPU-time performance and memory cost comparisons of $M_N$ and $M_C$ preconditioners using CGNR algorithm.
the most popular Krylov subspace iterative methods: the generalized minimal residual method GMRES [19], and the conjugate gradient method CGNR applied to the normal equations [18].

The GMRES algorithm is used with a right preconditioning. This method minimizes the residual and it is therefore mathematically equivalent to several other generalized conjugate gradient methods; such method requires only matrix-vector products per iteration and the convergence is accelerated using a preconditioner. Therefore, the preconditioners considered here were constructed from the standard ILUT(τ;lfil) factorization of A, noted MT; from the sparse approximate inverse technique SPAI [11], noted MS; and from the factorized sparse approximate inverse AINV [3], noted MF.

The fill-ins of MT is controlled by two parameters, a threshold drop tolerance τ = 10^{-4} and a fill-in number l_{fil} = 10; the accuracy of MS is controlled by an accuracy, ε = 0.1, used to solve the n independent least squares problem, and by a parameter s = 5 used in step 5 of Algorithm 2 to keep at most s indices with smallest ρ_j; and the fill-ins of MF is controlled by a value of the drop tolerance, β = 0.1 [2].

The CGNR algorithm is used with a split preconditioning. It is used to solve
the normal equations (2) in the goal to compare it with the first method. This method minimizes the $A^T A$-norm of the error, which is the $2$-norm of the residual $(b - Au_k)$, over the affine subspace

$$u_k \in u_0 + \text{span}[A^T r_0, (A^T A)r_0, (A^T A)A^T r_0, \ldots, (A^T A)^{(k-1)}A^T r_0].$$

This technique is a popular combination to solve nonsymmetric linear systems with a conjugate gradient method. The preconditioner, noted $M_N$, is computed from the normal matrix using the stabilized sparse approximate inverse sainv algorithm described in Section 2.3; for more details see [2]. The accuracy of $M_N$ is controlled in the same way as $M_F$, i.e., $\beta = 0.1$ and $s = 10$. The latest preconditioner used in this paper was provided from the drop-tolerance incomplete Cholesky factorization $\text{ic}(10^{-4})$ applied to $A^T A$, referred to $M_C$.

For all computations, the right hand side is generated by assuming that the exact solution is a vector of all ones; the initial guess is a vector of some random numbers; the iterations are stopped when the 2-norm of the residual is reduced by a factor of $10^8$, i.e., $\|r_m\|_2 < 10^{-8}\|r_0\|_2$; and the numerical experiments are conducted on a Pentium CPU at 1.6 GHZ with 1024 Mb RAM. In Tables 2 and 4, "solu." is the CPU time in seconds for the solution phase (iteration), "prec." shows the CPU time in seconds spent in constructing the preconditioners (preprocessing phase), and "spar." shows the sparsity ratio which is the ratio between the number of nonzero elements of the preconditioner to that of the original matrix.

Results with different $h$ and $Re$ are presented in Table 1. It is clear to see that the performance of GMRES is related to some algebraic properties of the coefficient matrix $A$. By comparing results, we can see that the GMRES number of iterations to achieve convergence with $M_T$ is less than that with $M_S$ and $M_F$, and it is strongly affected by the magnitude of the Reynolds number. More precisely, for diffusion-dominated cases and small meshsizes, we guarantee the diagonal dominance of the matrix $A$ that ensure the well behavior of the iterative method. For convection-dominated cases, we lose the diagonally dominant property and the GMRES algorithm has an irregular behavior for all preconditioners; GMRES with the SPAI preconditioner gives bad results. Test results given in Table 2 shows that the "solu." solution phase time for $M_T$ is larger than those for $M_S$ and $M_F$. That is because of at each GMRES-iteration two triangular linear systems have to be solved using $M_T$ preconditioner, where as only matrix-vector products are required for $M_S$ and $M_F$. A major advantage of iterative methods over direct methods is that they may require far less memory. The "spar." sparsity results in Table 2 show that $M_T$ yield better performance than $M_F$ and much more than $M_S$, the sparsities are highly dependent on the dropping threshold using in the reduction. A last trivial remark, we see that the "prec." CPU time increases with the sparsity ratio.
The preconditioned CGNR algorithm is used for solving problems in the second type of matrices (spd), related results are exhibited in Table 3. It is shown that CGNR with the SAINV preconditioner \((M_N)\) gives an irregular behavior, and the number of iterations is larger than that with the IC\((10^{-4})\) preconditioner \((M_C)\). Furthermore, IC\((10^{-4}) +\) CGNR algorithm and ILUT\((10^{-4}, 10)\) + GMRES algorithm give almost the same results, with a small advantage of the second combination. Clearly, the "solu." solution phase time for \(M_C\) is larger than that for \(M_N\) since, as noted before, we have to solve two triangular systems using \(M_C\) and only matrix-vector products are required for \(M_N\) (Table 4). Note that the performance of all methods considered here deteriorates with increasing problem size. One last remark concerns the ordering issue. All factorized preconditioners (like ILUT, AINV or SAINV) are sensitive to the ordering of the grid points. In the literature, there are several papers showing that the performance of both ILUT and AINV on convection-diffusion problems can be greatly improved by reordering. For ILUT it is best to use reverse Cuthill-McKee (RCM), whereas for AINV it is best to use some variant of the minimum degree algorithm [5].

4 Conclusion

We have presented a number of known techniques for solving large, sparse and nonsymmetric linear systems arising from the central finite difference discretization of the convection-diffusion equation. These are combinations of Krylov subspace methods with different classes of preconditioners in the goal to improve the convergence, CPU time performance and cost memory for two types of convection-diffusion equation. Knowing that CG methods with incomplete Cholesky factorization lead always to a fast convergence and a good convergence rate, we have applied the same strategies to the normal equations. The associated results are not quite satisfactory, this is because \(A^T A\) is more ill-conditioned than the original matrix \(A\). We have numerically shown that the GMRES algorithm with the standard incomplete preconditioner ILUT leads to best results, followed by the CGNR algorithm with standard incomplete Cholesky preconditioner IC; however these two preconditioners remain completely implicit, then the degree of parallelism is very limited. The explicit preconditioners arising from SPAI, AINV or SAINV techniques lead to bad results and irregular behaviors. Numerical results showed that, GMRES algorithm with SPAI preconditioner combination gives bad results.
References


Received: March 15, 2014; Published: March 23, 2016