

Numerical study on incomplete orthogonal factorization preconditioners¹

Zhong-Zhi Bai², Iain S. Duff³ and Jun-Feng Yin⁴

Technical Report TR/PA/08/24

April 1, 2008

CERFACS
42 Ave G. Coriolis
31057 Toulouse Cedex
France

ABSTRACT

We design, analyse and test a class of incomplete orthogonal factorization preconditioners constructed from Givens rotations, incorporating some dropping strategies and updating tricks, for the solution of large sparse systems of linear equations. Comprehensive accounts about how the preconditioners are coded, what storage is required and how the computation is executed for a given accuracy are presented. A number of numerical experiments show that these preconditioners are competitive with standard incomplete triangular factorization preconditioners when they are applied to accelerate Krylov subspace iteration methods such as GMRES and BiCGSTAB.

Keywords: preconditioning, IQR, ILU, Givens rotation, incomplete orthogonal factorization, nonsymmetric matrix, least-squares, normal equations.

AMS(MOS) subject classifications: 65F05, 65F10, 65F25, 65F50.

¹Current reports available at <http://www.cerfacs.fr/algor/reports/index.html>. Also appeared as Technical Report RAL-TR-2008-010 from Rutherford Appleton Laboratory, Oxfordshire.

² bzz@lsec.cc.ac.cn. State Key Laboratory of Scientific/Engineering Computing, Institute of Computational Mathematics and Scientific/Engineering Computing, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, P.O. Box 2719, Beijing 100080, P.R. China. Supported by The National Basic Research Program (No. 2005CB321702), The China Outstanding Young Scientist Foundation (No. 10525102) and The National Natural Science Foundation (No. 10471146), P.R. China.

³ duff@cerfacs.fr. Also at CSED, Atlas Centre, RAL, Oxon OX11 0QX, England.

⁴ yinjf@lsec.cc.ac.cn. Current address: Department of Mathematics, Tongji University, Shanghai 200092, P.R.China. junfengyin@gmail.com.

Contents

1	Introduction	1
2	The Practical IGO Method	3
3	The Dropping Strategies	7
3.1	Dropping Fill-ins Based on Numerical Thresholds	8
3.2	Dropping Fill-ins Based on Sparsity Patterns	9
4	The Implementation Strategies	10
4.1	Sparse Matrix Storage and Operations	10
4.2	Preconditioning of Iteration Methods	11
5	Description of Experiments	11
5.1	The Convection-Diffusion Problem	12
5.2	Matrices from Matrix Market	13
6	Numerical Results	15
6.1	The Convection-Diffusion Problems	15
6.2	Matrices from Matrix Market	19
6.3	The Approach of Normal Equations	21
7	Conclusions	22

1 Introduction

We consider solutions of large sparse systems of linear equations of the form

$$Ax = b, \quad A \in \mathcal{R}^{m \times n}, \quad x \in \mathcal{R}^n \quad \text{and} \quad b \in \mathcal{R}^m, \quad (1.1)$$

where $m \geq n$, A is a given real matrix, b is a given real right-hand-side vector, and x is the unknown vector. Linear systems of this form often arise in many areas of scientific computing and engineering applications; see [2, 4, 7, 11, 36, 41].

Throughout this paper, we assume that the matrix A is nonsingular when $m = n$, and is of full column rank when $m > n$. For the latter case, we actually consider the least-squares solution of the system of linear equations (1.1), which may be computed through solving the normal equations

$$A^T Ax = A^T b, \quad (1.2)$$

where A^T represents the transpose of the matrix A . See [9, 15, 19].

Iterative methods, such as Krylov subspace methods combined with high-quality preconditioners or preconditioning processes, are practical and effective solvers for the system of linear equations (1.1), with respect to computation costs and memory requirements [3, 34, 35]. The preconditioning process plays a crucial role for improving the convergence property of the Krylov subspace method and for guaranteeing the numerical accuracy of the computed solution when an appropriate stopping criterion is employed [1].

For preconditioning techniques, *incomplete triangular* (**ILU**) and *incomplete orthogonal* (**IQR**) factorizations are attractive and popular candidates in actual applications [2, 26, 27, 32]. The ILU factorization, normally used only for square matrices, computes a sparse unit lower-triangular matrix L and a sparse upper-triangular matrix U using Gaussian elimination coupled with some dropping rules so that the error matrix $E = LU - A$ satisfies certain constraints, such as having zero entries in some positions [2, 27, 33]. One important example is the *incomplete Cholesky* (**IC**) factorization for a symmetric positive definite matrix [10, 12, 20, 24]. For general nonsymmetric matrices, although a number of efficient ILU factorization techniques have been presented (see [2, 33, 34]), it is more difficult to give theoretical assurance about the robustness, feasibility and efficiency of these incomplete triangular factorization preconditioners. There can be breakdown in the factorization process due to zero pivots, inaccuracy of the incomplete triangular factors due to small pivots and inefficient dropping rules, as well as instability of the triangular solves due to the poorly conditioned incomplete triangular factors. Most IQR factorizations, which can be used for both square and rectangular matrices, compute a sparse and generally non-orthogonal matrix Q and a sparse upper-triangular matrix R by the modified Gram-Schmidt process incorporating some dropping rules. Recently, using a strategy that only drops entries of the upper-triangular matrix R , the authors of [37] proved the existence and stability of the associated IQR factorization preconditioner. For certain sparsity patterns, this strategy produces an R factor identical to that produced by the IC factorization applied

to the normal equations (1.2). In addition to the drawbacks of breakdown, inaccuracy and instability as in the ILU factorization, one major problem about the abovementioned IQR factorization is that the matrix Q is not in general orthogonal, and nothing guarantees that it is even nonsingular unless we adopt a strategy that does not drop many entries. However, this makes the resulting incomplete factors Q and R likely to be too dense to be useful in practice; see [32].

Considering the advantages and the power of the complete orthogonal factorization process, Bai, Duff and Wathen [5] presented a class of incomplete orthogonal factorization methods based upon Givens rotations that they call IGO methods. These IQR factorizations can be used for both square and rectangular matrices, and they can always produce a sparse nonsingular upper-triangular matrix R , and an orthogonal or a sparse nonsingular matrix Q such that the error matrices $E = QR - A$ and $E_0 = Q^T Q - I$ are “small”, where I is the identity matrix. Here, the word “small” means that either the values in the entries of E , E_0 are small or the incomplete factors satisfy certain prescribed nonzero patterns. That such incomplete Givens strategies can always compute an orthogonal factor Q (orthogonal to the limits of finite precision arithmetic) is a particular feature of this approach. One consequence is that the R factor is always an incomplete Cholesky factor of the matrix $A^T A$ in the normal equations (1.2). For this situation, Q is not generally required and therefore need not be stored. In [5], the IGO methods were described in detail and their theoretical properties were analysed comprehensively, however, no experimental results were given. We refer to [30, 40] for some implementation strategies and numerical results, which demonstrate good performance of the IGO methods used to precondition Krylov subspace iteration methods for certain problems, in particular least-squares problems.

In this paper, we will focus on an incomplete orthogonal factorization based on Givens rotations with some specially prescribed storage and sparsity-preserving techniques. We call this factorization method the *practical IGO method*. After describing the basic algorithm, we give a theoretical analysis and discuss the coding of the algorithm using a high level language and storing the preconditioner using sparse matrix techniques. A brief comparison of the new method with a few related existing ones is also given. Then, we apply the practical IGO preconditioner to a range of matrices arising from finite-difference discretizations of convection-diffusion equations and from the Harwell-Boeing collection in the *Matrix Market*; see [7, 25]. Numerical results show that the practical IGO preconditioner is accurate, robust and efficient for preconditioning the Krylov subspace methods for solving large sparse systems of linear equations, and is generally superior to the standard ILU preconditioner. We then show that our preconditioner is very competitive with RIF and CIMGS preconditioners when solving least-squares problems.

The rest of the paper is organized as follows: Section 2 describes the practical IGO method and analyses its theoretical properties. In Section 3 we discuss the dropping strategies for the practical IGO method. Some considerations about implementation details such as storage and dropping strategies are discussed in Section 4. Several classes of

experimental problems are described in Section 5, and numerical results are given and discussed in Section 6. Finally, in Section 7 we end this paper with some remarks and conclusions.

2 The Practical IGO Method

While general *incomplete Givens orthogonalization (IGO)* methods were constructed and analysed in [5], in actual applications we may need to change and omit some of the subordinate details so that a more practical and efficient version of the IGO method can be obtained.

To give a precise description of the new practical IGO method, we first review the well-known Givens rotation and some of its useful properties.

A Givens rotation (or plane rotation) $G(i, j, \theta) \in \mathcal{R}^{m \times m}$ is equal to the identity matrix except that

$$G([i, j], [i, j]) = \begin{pmatrix} c & s \\ -s & c \end{pmatrix},$$

where $c = \cos \theta$ and $s = \sin \theta$. The implementation $y = G(i, j, \theta)x$ rotates x through θ radians clockwise in the (i, j) -plane. Algebraically,

$$y_k = \begin{cases} x_k, & \text{for } k \neq i, j, \\ cx_i + sx_j, & \text{for } k = i, \\ -sx_i + cx_j, & \text{for } k = j, \end{cases} \quad 1 \leq k \leq m.$$

So, $y_j = 0$ if

$$s = \frac{x_j}{\sqrt{x_i^2 + x_j^2}} \quad \text{and} \quad c = \frac{x_i}{\sqrt{x_i^2 + x_j^2}}.$$

Givens rotations are therefore useful for introducing zeros into a vector one at a time. This property is especially useful when we handle sparse matrices. Note that there is no need to work out the angle θ , since s and c in the above are all that are needed to apply the rotation. Hence, to simplify our discussion, a Givens rotation $G(i, j, \theta)$ is abbreviated as $G(i, j)$ in the sequel.

Let the sets of integer pairs

$$\begin{aligned} P_{m,n} &= \{(i, j) \mid 1 \leq i \leq m, \quad 1 \leq j \leq n\}, \\ P_L &= \{(i, j) \mid i \geq j, \quad 1 \leq i \leq m, \quad 1 \leq j \leq n\} \quad \text{and} \\ P_U &= \{(i, j) \mid i \leq j, \quad 1 \leq i \leq m, \quad 1 \leq j \leq n\} \end{aligned}$$

represent the nonzero patterns of any general, lower-triangular and upper-triangular matrices in $\mathcal{R}^{m \times n}$, respectively. For a given matrix $A = (a_{ij}) \in \mathcal{R}^{m \times n}$, we use P_A to

denote its nonzero pattern, and $P_{A,L}$ and $P_{A,U}$ the nonzero patterns of its lower-triangular and upper-triangular parts, respectively. That is to say,

$$\begin{aligned} P_A &= \{(i, j) \mid a_{ij} \neq 0, \quad 1 \leq i \leq m, \quad 1 \leq j \leq n\}, \\ P_{A,L} &= \{(i, j) \mid a_{ij} \neq 0, \quad i \geq j, \quad 1 \leq i \leq m, \quad 1 \leq j \leq n\}, \quad \text{and} \\ P_{A,U} &= \{(i, j) \mid a_{ij} \neq 0, \quad i \leq j, \quad 1 \leq i \leq m, \quad 1 \leq j \leq n\}. \end{aligned}$$

To define an IQR factorization ($Q_{inc}R_{inc}$ of the matrix A based on Givens rotations, we let P_R be the chosen nonzero pattern for the matrix R_{inc} and P_Q the index set of the chosen Givens rotations used for rotating out the nonzero entries in the lower-triangular part of the matrix A . Without causing any confusion, we also call P_Q the nonzero pattern of the incomplete orthogonal factor Q_{inc} . See [5] for details.

The practical IGO method is a simplified and modified variant of the incomplete Givens orthogonalization method, that is Method 3.1 proposed by Bai, Duff and Wathen in [5]. It essentially consists of the following two elementary processes:

- (a) use Givens rotations to annihilate column by column the nonzero entries located at the strictly lower-triangular part of the matrix $A \in \mathcal{R}^{m \times n}$ from the bottom up to the first sub-diagonal, and update the matrix A by applying a defined dropping strategy P ;
- (b) form the corresponding row of the incomplete upper-triangular matrix $R_{inc} \in \mathcal{R}^{m \times n}$ by applying some dropping rule P_R .

In the process (a), when a Givens rotation, say $G(i, j)$, is applied to rows i and j ($i < j$) of the matrix $A = (a_{ij})$, it does not update all entries a_{ik} and a_{jk} for $k = i + 1, i + 2, \dots, n$ as was done in [5], but it updates according to the following rule.

Givens-Updating Rule:

For $k = i + 1, i + 2, \dots, n$

1. If $(i, k) \in P$ and $(j, k) \in P$ then
2. If $a_{ik} \neq 0$ and $a_{jk} \neq 0$ then
3. $a_{ik} := ca_{ik} + sa_{jk}$
4. $a_{jk} := -sa_{ik} + ca_{jk}$
5. EndIf
6. else
7. Set $a_{ik} := a_{ik}$
8. Set $a_{jk} := a_{jk}$
9. EndIf

EndFor

In the process (b), the rows of the matrix R_{inc} are formed by applying the dropping strategy P_R .

The above Givens-updating rule used in process (a) can be intuitively illustrated by the following example. Let the nonzero pattern of the matrix A be given as

$$A = \begin{pmatrix} \times & 0 & \times & \times \\ 0 & \times & 0 & \times \\ \times & \times & \times & 0 \\ \times & \times & 0 & \times \end{pmatrix},$$

where the symbol “ \times ” indicates that the entry in this position is nonzero. Define the nonzero patterns P_Q and P_R of the incomplete factors Q_{inc} and R_{inc} as

$$P_Q = \{(3, 1), (4, 2), (3, 2)\} \quad \text{and} \quad P_R = \begin{pmatrix} \times & 0 & \times & 0 \\ 0 & \times & 0 & \times \\ 0 & 0 & \times & 0 \\ 0 & 0 & 0 & \times \end{pmatrix},$$

respectively. We should zero out the nonzero entries in the strictly lower-triangular part of the matrix A , i.e., a_{41} , a_{31} , a_{42} and a_{32} , to obtain the incomplete upper-triangular factor R_{inc} . To this end, we first preprocess the matrix A by dropping “small” entries, say a_{41} , and then use three plane rotations, i.e., $G(3, 1)$, $G(4, 2)$ and $G(3, 2)$, to complete the incomplete orthogonal factorization process. Here, the word “small” means that either the entry is numerically small or is subordinate in position. The following figure presents a detailed illustration of this process:

$$A = \begin{pmatrix} \times & 0 & \times & 0 \\ 0 & \times & 0 & \times \\ \times & \times & \times & 0 \\ \times & \times & 0 & \times \end{pmatrix} \xrightarrow{\text{Preproc}} \begin{pmatrix} \times & 0 & \times & \times \\ 0 & \times & 0 & \times \\ \times & \times & \times & 0 \\ 0 & \times & 0 & \times \end{pmatrix} \xrightarrow{G(3,1)} \begin{pmatrix} * & 0 & * & \times \\ 0 & \times & 0 & \times \\ 0 & \times & * & 0 \\ 0 & \times & 0 & \times \end{pmatrix} \xrightarrow{G(4,2)} \begin{pmatrix} * & 0 & * & 0 \\ 0 & \odot & 0 & \odot \\ 0 & \times & * & 0 \\ 0 & 0 & 0 & \odot \end{pmatrix} \xrightarrow{G(3,2)} \begin{pmatrix} * & 0 & * & 0 \\ 0 & \otimes & 0 & \odot \\ 0 & 0 & * & 0 \\ 0 & 0 & 0 & \odot \end{pmatrix} = R_{inc}.$$

From the above investigation, we have the following observations:

- when a Givens rotation $G(i, j)$ is used to update the i -th and the j -th rows, we only update those entries at (i, k) and (j, k) positions which are such that both a_{ik} and

a_{jk} are nonzero and both (i, k) and (j, k) belong to P . As a result, after the Givens transform with $G(i, j)$, the updated matrix $\tilde{A} := G(i, j)A$ still inherits the sparsity pattern of the original matrix A and, thereby, the final incomplete upper-triangular factor R_{inc} possesses the same sparsity pattern as the upper-triangular part $P_{A,U}$ of the matrix A ;

- for given indices i and j such that $i < j$, the Givens rotation $G(i, j)$ may make a contribution to the incomplete orthogonal factor Q_{inc} only for (i, j) such that $a_{ij} \neq 0$;
- the sparsity patterns P_Q and P_R satisfy $P_Q \subseteq P_{A,L}$ and $P_R \subseteq P_{A,U}$.

Given sets of integer pairs P_l and P_u satisfying $P_{A,L} \subseteq P_l \subseteq P_L$ and $P_{A,U} \subseteq P_u \subseteq P_U$, we now give an algorithmic description of the new practical IGO method.

Method 2.1 (The Practical IGO Method)

For $j = 1, 2, \dots, n - 1$

1. For $i = r := \max\{k | k > j, a_{kj} \neq 0\}$ DownTo $j + 1$
 2. If $(i, j) \in P_l$ and $a_{ij} \neq 0$ then
 3. Compute $\rho := \sqrt{a_{jj}^2 + a_{ij}^2}$
 4. Compute $c := a_{jj}/\rho$
 5. Compute $s := a_{ij}/\rho$
 6. Set $a_{jj} := \rho$ and $a_{ij} := 0$
 7. Store c and s
 8. For $k = j + 1, j + 2, \dots, n$
 9. For $(i, k) \in P_u$ and $(j, k) \in P_u$
 10. If $a_{ik} \neq 0$ and $a_{jk} \neq 0$
 11. Compute $temp := -sa_{jk} + ca_{ik}$
 12. Compute $a_{jk} := ca_{jk} + sa_{ik}$
 13. Set $a_{ik} := temp$
 14. EndIf
 15. EndFor
 16. EndFor
 17. EndIf
 18. EndFor
- EndFor

Before continuing, we make the following remarks about the practical IGO method:

- (i) a dropping rule tells us which entries of the matrix are allowed to be kept or filled-in during the incomplete orthogonal factorization process. The sets P_l and P_u in the practical IGO method play a role of static dropping rules for Q_{inc} and R_{inc} . In fact, we can define the sets P_l and P_u by accepting more fill-ins based on the numerical magnitude and/or the sparsity pattern during the factorization process, so that more

stable but more costly dynamic dropping rules can possibly be obtained. Of course, different dropping rules may lead to different versions of the practical IGO method;

- (ii) the incomplete orthogonal factor Q_{inc} need not be explicitly computed; only the (c, s) -pairs produced by the Givens rotations need to be stored. The lower-triangular part of the matrix A can be used to save such (c, s) -pairs once an entry in this part is annihilated by the corresponding Givens rotation;
- (iii) the matrix A is updated immediately just after each Givens rotation and, finally, its upper-triangular part gives the matrix R_{inc} . Therefore, it is unnecessary to update and store R_{inc} separately, but it may be necessary to store the original matrix A separately if it needs to be used again later;
- (iv) the practical IGO method presented here is quite general. Some specific strategies about its implementation such as the choices of the sparsity patterns P_Q and P_R , as well as the dropping rules for the incomplete factors Q_{inc} and R_{inc} , will be discussed in Section 3.

Now, assume all diagonal entries of the matrix A are nonzero. Then from the construction of the practical IGO method we know that the first $n - 1$ diagonal entries of the incomplete upper-triangular factor R_{inc} are positive. Therefore, if A is nonsingular, then R_{inc} is also nonsingular. Q_{inc} is orthogonal. These properties are restated precisely in the following theorem, which can be proved in an analogous fashion to the proof in [5].

Theorem 2.1 *Let $m = n$. Assume $A \in \mathcal{R}^{n \times n}$ is nonsingular, and Q_{inc} and $R_{inc} \in \mathcal{R}^{n \times n}$ are, respectively, the incomplete orthogonal and the incomplete upper-triangular factors produced by the practical IGO method. Then*

- (i) R_{inc} is sparse and nonsingular, and its diagonal elements are positive, except possibly for the last one;
- (ii) Q_{inc} is orthogonal.

This theorem shows that the practical IGO method can produce an incomplete orthogonal factor Q_{inc} and an incomplete upper-triangular factor R_{inc} , which are both sparse and nonsingular. Here, the sparsity of Q_{inc} can be understood in the sense that it is based on the sparsity pattern P_Q . Numerical results in Section 6 will show that this practical IGO method can present a stable and efficient preconditioner for Krylov subspace iteration methods in actual applications.

3 The Dropping Strategies

We recall that the error matrix E in an IGO method with respect to the matrix A is defined as

$$E = A - Q_{inc}R_{inc} \quad \text{or} \quad A = Q_{inc}R_{inc} + E. \quad (3.1)$$

The factorization error E is important because we should make the entries in E as small as possible in the sense of magnitude or position in order to improve the computational efficiency of the incomplete factorization used as a preconditioner for a Krylov subspace iteration method. A number of typical dropping strategies will be described in detail in this section so that economical and effective preconditioners can be obtained in actual applications.

Generally speaking, the dropping strategies can be categorized as static or dynamic. A static dropping strategy may result from a fixed threshold or sparsity pattern without introducing or rejecting any fill-ins during the factorization process, while a dynamic dropping strategy may result from changing a threshold or a sparsity pattern in which the values of the stored entries or the locations of the permissible fill-ins are determined during the factorization process.

For most matrices, the numerical behaviour of the IGO preconditioners obtained from dropping nonzero entries according to the magnitude of value or the sparsity pattern of the matrix are considerably different. Usually, the former may lead to more robust and accurate preconditioners, although it may be more costly and more prone to give an unstable triangular solve than the latter.

3.1 Dropping Fill-ins Based on Numerical Thresholds

This class of dropping strategies are useful especially for indefinite and non-diagonally dominant matrices. For a dropping technique based on numerical magnitude, fill-in is permitted only if the value of an entry is larger than a prescribed tolerance and/or the number of nonzero entries in each row is not larger than a specified integer. The following example gives a brief illustration about the fundamental principle of this threshold-based dropping strategy.

Consider a basic IGO step

$$\begin{pmatrix} \hat{a} & \hat{b} \\ \hat{d} & \hat{e} \end{pmatrix} = \begin{pmatrix} \hat{c} & -\hat{s} \\ \hat{s} & \hat{c} \end{pmatrix} \begin{pmatrix} \hat{c}\hat{a} + \hat{s}\hat{d} & \hat{c}\hat{b} + \hat{s}\hat{e} \\ 0 & -\hat{s}\hat{b} + \hat{c}\hat{e} \end{pmatrix}, \quad (3.2)$$

with

$$\hat{c} = \frac{\hat{a}}{\sqrt{\hat{a}^2 + \hat{d}^2}} \quad \text{and} \quad \hat{s} = \frac{\hat{d}}{\sqrt{\hat{a}^2 + \hat{d}^2}},$$

that aims to annihilate the $(2,1)$ -entry \hat{d} in the original matrix. The dropping strategy either drops a small entry or fills in a larger entry once the entry $\hat{c}\hat{b} + \hat{s}\hat{e}$ is computed. Here, for a given threshold `droptol`, entries less than `droptol` (or $\|w\| \times \text{droptol}$) are set to zero and those larger than `droptol` (or $\|w\| \times \text{droptol}$) are kept, where w is often chosen to be the largest entry in absolute value or the corresponding row of the matrix, and $\|w\|$ denotes either the absolute value of w when it is a real or the norm of w when it is a vector.

Usually, the number of fill-ins generated during the factorization process is larger than the number dropped. So, a practical implementation of the IGO method may introduce

another parameter `lfil` that is used to limit the maximum number of fill-ins allowed in each row. The parameter `lfil` may make the storage of the IGO preconditioner acceptable and known beforehand.

Sometimes, it may be desirable to adopt dropping strategies to make the upper-triangular factor R_{inc} sparser or more approximate. To give an example of such a strategy, we consider the factorization in (3.2) again. If $|\hat{d}| \ll |\hat{a}|$, then $\hat{c} \approx 1$ and $\hat{s} \approx 0$. By setting $\hat{c} := 1$ and $\hat{s} := 0$ before the Givens rotation is used, we can see that (3.2) reduces to

$$\begin{pmatrix} \hat{a} & \hat{b} \\ \hat{d} & \hat{e} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \hat{a} & \hat{b} \\ 0 & \hat{e} \end{pmatrix} = \begin{pmatrix} \hat{a} & \hat{b} \\ 0 & \hat{e} \end{pmatrix}.$$

Hence, the factorization error is given by

$$\begin{pmatrix} \hat{a} & \hat{b} \\ \hat{d} & \hat{e} \end{pmatrix} - \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \hat{a} & \hat{b} \\ 0 & \hat{e} \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ \hat{d} & 0 \end{pmatrix},$$

which is small because \hat{d} is. However, if the dropping rule is applied after the factorization, then the error is given by

$$\begin{pmatrix} \hat{a} & \hat{b} \\ \hat{d} & \hat{e} \end{pmatrix} - \begin{pmatrix} \hat{c} & -\hat{s} \\ \hat{s} & \hat{c} \end{pmatrix} \begin{pmatrix} \hat{c}\hat{a} + \hat{s}\hat{d} & \hat{c}\hat{b} + \hat{s}\hat{e} \\ 0 & -\hat{s}\hat{b} + \hat{c}\hat{e} \end{pmatrix} = \begin{pmatrix} \hat{s}^2\hat{a} - \hat{c}\hat{s}\hat{d} & 0 \\ 0 & 0 \end{pmatrix}.$$

Because

$$|\hat{s}^2\hat{a} - \hat{c}\hat{s}\hat{d}| = \frac{|\hat{a}\hat{d}(\hat{a} - \hat{d})|}{\hat{a}^2 + \hat{d}^2} = \frac{|1 - \hat{d}/\hat{a}|}{1 + (\hat{d}/\hat{a})^2} \cdot \hat{d} \approx \hat{d},$$

it is clear that the error from the first dropping strategy is about the same as that from the second. This shows that it may be advantageous to immediately set $\hat{d} := 0$ and simply skip a Givens rotation if the absolute value of the corresponding entry \hat{d} is less than `droptol`. Of course, if a dropping strategy that skillfully combines the functions of both `droptol` and `lfil` is employed, then an effective IGO method can be obtained, which will lead to a more robust and accurate IGO preconditioner, although such an IGO process may have the drawback of increasing the condition number and the degree of non-normality of the matrices due to possible propagation of some large entries.

3.2 Dropping Fill-ins Based on Sparsity Patterns

Another class of dropping strategies is based on dropping small entries according to the sparsity pattern of the matrix. It was originally developed for preconditioning finite-difference matrices from elliptic partial differential equations using an ILU factorization, for which the structure of the incomplete triangular factors was determined by a discretization stencil for the partial differential operator; see [17, 20, 28, 29]. To make the incomplete factorization more accurate, we often adopt a larger stencil for the triangular factors, or

in other words, accept more nonzero entries in their sub- or super-diagonals; see [2, 33] for detailed descriptions.

These dropping strategies can be extended in a straightforward manner to the IQR factorizations. In actual applications of the practical IGO method, we may allocate the memory before the factorization process is started if the structures of the incomplete orthogonal factor Q_{inc} and the incomplete upper-triangular factor R_{inc} are already determined. The most computationally efficient forms are probably the row-wise and the column-wise forms, although they may be inappropriate for some classes of indefinite matrices.

4 The Implementation Strategies

In Section 4 we describe several implementation strategies for the practical IGO method when it is used to precondition Krylov subspace iteration methods for solving large sparse systems of linear equations under limited memory and execution time.

4.1 Sparse Matrix Storage and Operations

Consider the sparse matrix $A \in \mathcal{R}^{n \times n}$ with `nnz` nonzero entries. As is known, the easiest way for storing the matrix A may be the so-called *simple coordinate scheme*, which uses two integer arrays of length `nnz` to store the row and the column indices and a real array of length `nnz` to store the values of the nonzero entries. In fact, this is exactly the storage scheme used by the *Matrix Market* database for matrices and systems of linear equations; see [25].

Cheaper and more useful techniques for storing sparse matrices are the *compressed sparse column* (CSC) and the *compressed sparse row* (CSR) schemes [16], which usually require less storage than the simple coordinate scheme and in which elementary operations such as a matrix-vector product can be easily performed. In addition these two compressed storage schemes have their own merits. For an algorithm that proceeds by columns (such as column-wise annihilation of entries by Givens rotations) it is advantageous to use the CSC format of a matrix, whereas for an algorithm that proceeds by rows (such as back substitution), it is advantageous to use the CSR format. It turns out that in the sense of execution time a more effective implementation of the practical IGO method may come from appropriately using both storage formats in various parts of the algorithm. It is fortunate that constructing a CSR format from an existing CSC format is easy and cheap.

In actual computations, the diagonal entries of the matrix A and the corresponding entries of the incomplete factors are always assumed to be nonzero and stored, whether they are zero or not. Also, it is implicitly assumed that each column (for CSC) or row (for CSR) of the matrices contains at least one nonzero entry or, alternatively, a position allowed to have a nonzero value. For nonsingular matrices, these assumptions are obviously not restrictive.

4.2 Preconditioning of Iteration Methods

In this subsection, we estimate the computing cost for preconditioning a Krylov subspace iteration method by using the practical IGO method and give a comparison with that for using a standard ILU method.

Given a preconditioning matrix $M \in \mathcal{R}^{n \times n}$, we need to solve the generalized residual equation

$$Mz = r \tag{4.1}$$

at each step of a Krylov subspace iteration, where $r \in \mathcal{R}^n$ is the currently available residual. For the practical IGO preconditioner, we have $M = Q_{inc}R_{inc}$, with $Q_{inc} \in \mathcal{R}^{n \times n}$ orthogonal and $R_{inc} \in \mathcal{R}^{n \times n}$ nonsingular and upper triangular. So, to solve (4.1) we only need to compute the matrix-vector product $Q_{inc}^T r$ and solve the upper-triangular system of linear equations

$$R_{inc}z = Q_{inc}^T r.$$

For the standard ILU preconditioner, we have $M = L_{inc}U_{inc}$, with $L_{inc} \in \mathcal{R}^{n \times n}$ and $U_{inc} \in \mathcal{R}^{n \times n}$ lower- and upper-triangular. So, to solve (4.1) we need to solve the lower- and the upper-triangular systems of linear equations

$$L_{inc}\tilde{z} = r \quad \text{and} \quad U_{inc}z = \tilde{z}.$$

Recall that in the practical IGO factorization process, we use a Givens rotation once when there is a nonzero entry below the diagonal of the matrix A and belonging to the given sparsity pattern P_l ; see Method 2.1. This is in spirit analogous to the use of a Gauss transformation in a standard ILU factorization process. The difference is that by using a Givens rotation two rows need to be updated while by using a Gauss transform only one row needs to be updated. Therefore, to zero out an entry the practical IGO method usually costs much more than a standard ILU method. However, when the percentage of nonzero entries in the matrix A is small, these two methods may have about the same computing cost. See some numerical evidence given in Section 6.

In the practical IGO factorization process, it is not necessary to store and compute the incomplete orthogonal factor Q_{inc} . All we need is to store the quantities c and s in two arrays, compute the products of the Givens rotations with the currently available right-hand-side vectors, and then solve an upper-triangular linear system with the coefficient matrix R_{inc} by a BLAS routine. Note that for a standard ILU method, a lower- and an upper-triangular linear system need to be solved by the BLAS routines. We should point out that good properties of a preconditioner also depend on the good algebraic properties and sparse structure of the target matrix A itself.

5 Description of Experiments

A number of numerical experiments have been performed to assess the stability, accuracy and efficiency of the practical IGO method. This is achieved through comparisons between

the numerical behaviour of the practical IGO method and standard ILU preconditioners applied to Krylov subspace iteration methods such as GMRES[31] and BiCGSTAB[35] for solving systems of linear equations[32, 33, 5, 30]. The test matrices include finite difference matrices from a convection-diffusion equation with different choices of the problem parameters and matrices from the Harwell-Boeing collection in the *Matrix Market*. Besides, the normal equation approach (1.2) is also implemented by employing the practical IGO and the standard ILU preconditioners with the conjugate gradient method[19, 23].

In the experiments, the CSR format is used to store all the matrices and all codes are written in C++ and compiled on a Linux system. We remark that similar numerical results can be obtained for matrices stored in the CSC format. In addition, all initial guesses x_0 for the iterative solvers are randomly generated with a uniform distribution such that their entries belong to the interval $[-1, 1]$, and the iterations are terminated either when the number of iterations exceeds 1000 or when the current iterate satisfies $\|r_k\| \leq 10^{-6}\|r_0\|$, where $r_k = b - Ax_k$ is the residual at the k -th iteration.

5.1 The Convection-Diffusion Problem

Consider the following variable-coefficient convection-diffusion equation

$$\left\{ \begin{array}{l} -\nabla \cdot (\alpha(x, y)\nabla u) + q \left(\beta(x, y)\frac{\partial u}{\partial x} + \gamma(x, y)\frac{\partial u}{\partial y} \right) = f(x, y), \\ \text{for } (x, y) \in \Omega, \\ u = g(x, y), \\ \text{for } (x, y) \in \partial\Omega, \end{array} \right. \quad (5.1)$$

where $\alpha(x, y)$ is a uniformly positive/nonnegative function, $\beta(x, y)$ and $\gamma(x, y)$ are sufficiently regular functions, and q is a positive parameter used to control the magnitude of the convective term. The domain Ω is a square in \mathcal{R}^2 . The functions $f(x, y)$ and $g(x, y)$ are chosen such that $x_* = (1, 1, \dots, 1)^T$ is the exact solution of the corresponding system of linear equations whose coefficient matrix is the centered or the upwind finite-difference matrix of the convection-diffusion equation (5.1) on a uniform $N \times N$ discretization grid. We remark that this test problem was used in [13, 18] to perform and examine the effectiveness of incomplete triangular factorization and sparse approximate inverse preconditioners.

This class of linear systems becomes very difficult when the coefficient matrices are strongly nonsymmetric, or in other words, when the convective term of the convection-diffusion equation (5.1) is dominant. This can happen when q becomes very large.

For $N = 64$ and $N = 128$, we have implemented the abovementioned preconditioned Krylov subspace iteration methods for various choices of the functions $\alpha(x, y)$, $\beta(x, y)$ and $\gamma(x, y)$, which are listed in Table 5.1.

Table 5.1: Test Problems from the Convection-Diffusion Equation (5.1)

Prob. No.	$\alpha(x, y)$	$\beta(x, y)$	$\gamma(x, y)$
1	1	1	1
2	1	$x + y$	$x + y$
3	1	e^{x+y}	e^{x+y}
4	1	e^{x+y}	e^{-x-y}
5	1	e^{-x-y}	e^{x+y}
6	1	e^{-x-y}	e^{-x-y}
7	$x + y$	$x + y$	$x + y$
8	e^{x+y}	e^{x+y}	e^{x+y}

5.2 Matrices from Matrix Market

This group of test matrices includes twenty matrices selected from eleven sets (e.g., ASTROPH, BCSSTRUC1, BCSSTRUC2, BCSSTRUC3, CYLSHELL, etc.) of the Harwell-Boeing collection in the *Matrix Market*, a repository organized by National Institute of Standards and Technology; see [25]. The names and the characteristics including: the number m of rows, the number $\text{nnz}(\mathbf{A})$ of nonzeros, the density $\text{den}(\mathbf{A})$, the Frobenius norm (F -norm) and the estimated condition number $\text{cond}(\mathbf{A})$, of these matrices are given in Table 5.2. Note that the matrix A is square so that the number n of its columns is equal to m . Here, the density of a matrix $A \in \mathcal{R}^{m \times n}$ is defined as the percentage of its nonzero entries, i.e., $\text{den}(\mathbf{A}) := \frac{\text{nnz}(\mathbf{A})}{mn} \times 100$.

In Table 5.2, we classify the twenty matrices into five sub-groups according to their symmetry and definiteness. Clearly, most of the matrices are very ill-conditioned with very large F -norms, with only a few exceptions, for example NNC1374 and PDE2961. The sizes of the matrices vary significantly in terms of both dimension ($138 \leq m \leq 90449$) and number of nonzeros ($696 \leq m \leq 1921955$), the density ranges approximately from 0.02% to 8%, and the F -norm increases from 1.3e02 to 2.3e19.

The first sub-group includes eight matrices, which are symmetric positive definite; among them BCSSTK10, BCSSTK12 and BCSSTK14 are from the BCSSTRUC set from structural engineering applications, NOS1, NOS2 and NOS6 are from the LANPRO set from finite-element approximation to the biharmonic operator on a beam with one end free and one end fixed, and S3DKT3M2 and S3RMT3M1 are from the CYLSHELL set from the finite-element discretization of the octant of a cylindrical shell. The second sub-group includes three symmetric indefinite matrices, BCSSTK19, BCSSTK20 and BCSSTK26 that are from the BCSSTRUC set. The third sub-group includes two symmetric semidefinite matrices, BCSSTM10 and BCSSTM12, from the BCSSTRUC set. The fourth sub-group includes two pattern symmetric indefinite matrices, CAN_445 and CAN_838, from the LANPRO set of structural problems in aircraft design. And the fifth sub-group includes five nonsymmetric matrices from five different sets: CDDE5 is a model finite-difference matrix of a two-dimensional (2D) convection-diffusion operator, MCCA is

Table 5.2: Description of the Matrix Market Matrices

Matrix No.	Matrix Name	m	$\text{nnz}(A)$	$\text{den}(A)$	F -norm	$\text{cond}(A)$
Symmetric Positive Definite Matrices						
1	BCSSTK10	1086	22070	1.87	3.0e08	1.3e06
2	BCSSTK12	1473	34241	1.58	4.7e09	5.3e08
3	BCSSTK14	1086	63454	5.38	6.5e10	1.3e10
4	NOS1	237	1017	1.81	1.3e10	2.5e07
5	NOS2	957	4137	0.45	1.7e12	6.3e09
6	NOS6	675	3255	0.71	4.5e07	8.0e06
7	S3RMT3M1	5489	217669	0.72	1.7e05	1.3e10
8	S3DKT3M2	90449	1921955	0.02	0.6e06	3.6e11
Symmetric Indefinite Matrices						
9	BCSSTK19	817	6853	1.03	9.6e14	2.8e11
10	BCSSTK22	138	696	3.65	2.1e07	1.7e05
11	BCSSTK26	1922	30336	0.11	4.0e11	2.3e08
Symmetric Positive Semidefinite Matrices						
12	BCSSTM10	1086	22092	1.87	3.0e05	2.4e05
13	BCSSTM12	1473	19659	0.91	82	8.9e05
Pattern Symmetric Indefinite Matrices						
14	CAN_445	445	3809	1.92	–	–
15	CAN_838	838	10010	1.43	–	–
Nonsymmetric Matrices						
16	CDDE5	961	4681	0.51	1.3e02	5.3e04
17	MCCA	180	2659	8.21	2.3e19	3.6e17
18	NNC1374	1374	8588	1.67	9.6e03	1.0e02
19	PDE2961	2961	14585	0.17	2.2e02	9.5e02
20	SHERMAN2	1080	23094	1.98	7.0e09	1.4e12

from nonlinear radiative transfer and statistical equilibrium in astrophysics, NNC1374 is from models of nuclear reactor, PDE2961 is a five-point central difference matrix of 2D variable-coefficient linear elliptic equation, and SHERMAN2 is from the SHERMAN set of oil reservoir simulation challenge matrices.

For generating systems of linear equations for these coefficient matrices, we choose right-hand-side vectors such that their exact solutions are all equal to $x_* = (1, 1, \dots, 1)^T$. This set-up allows us to easily check the accuracy of the computed solution by using both the residual vector norm $\|Ax_k - b\|_2$ and the absolute solution error $\|x_k - x_*\|_2$.

6 Numerical Results

We examine the robustness, effectiveness and stability of the practical IGO preconditioner by numerically comparing it with the standard ILU preconditioner, which is used to accelerate GMRES [31] (without restarting) and BiCGSTAB [35]. We compare the methods with respect to the preconditioning time (CPU_p), the iteration time (CPU_{it}), the total number of iteration steps (IT) and the total CPU time (CPU); see detailed descriptions about these notations in Table 6.1.

Table 6.1: Notations

Notation	Description
N	the number of discretization points
CPU_p	the CPU time for constructing the preconditioner
CPU_{it}	the CPU time for executing the iterative scheme
IT	the total number of iterations for the iterative scheme
CPU	the total CPU time for the iteration

6.1 The Convection-Diffusion Problems

The numerical results in this subsection concern the IGO- and the ILU-preconditioned GMRES and BiCGSTAB methods, denoted as IGO-GMRES, ILU-GMRES and IGO-BiCGSTAB, ILU-BiCGSTAB, respectively, for solving systems of linear equations arising from the centered finite-difference discretization of the convection-diffusion equation (5.1); see Table 5.1.

In Table 6.2, we list the total number of iteration steps and the total CPU times for IGO-GMRES and ILU-GMRES, when $N = 64, 128$ and when $q = 500, 1000$, respectively. Clearly, all the IGO-GMRES iterations converge fast and accurately to the exact solution x_* . When q grows from 500 to 1000, the number of iteration steps increases by about a factor of 2 and the CPU time correspondingly increases by about two or three times. The ILU-GMRES iterations converge very slowly for $N = 64$, and they even diverge for the case $N = 128$ and $q = 1000$. Here, an iteration is termed divergent if it cannot achieve the stopping criterion within the prescribed number of iteration steps (1000). For the convergent iterations, IGO-GMRES is always less costly in computing time and requires fewer iteration steps than ILU-GMRES.

We plot the curves of IT and CPU for the eight test problems given in Table 5.1 for both IGO-GMRES and ILU-GMRES in Figures 6.1 and 6.2 when $q = 500$ and in Figures 6.3 and 6.4 when $q = 1000$, respectively. To identify the matrix sizes, we use IGO-GMRES(N) and ILU-GMRES(N) to denote IGO-GMRES and ILU-GMRES corresponding to the number $m = N \times N$ of discretization points. These figures show that both kinds of curves are very flat and smooth for the IGO-GMRES iteration, but are drastically steep and oscillating for ILU-GMRES iteration. It is evident that the practical IGO preconditioner is more

robust, stable and efficient than the standard ILU preconditioner when they are employed to precondition the GMRES method.

Table 6.2: IT and CPU for GMRES when $N = 64, 128$ and when $q = 500, 1000$

Prob. No.	N	$q = 500$				$q = 1000$			
		IGO-GMRES		ILU-GMRES		IGO-GMRES		ILU-GMRES	
		CPU	IT	CPU	IT	CPU	IT	CPU	IT
1	64	0.28	40	0.72	72	0.75	71	8.31	260
	128	1.12	39	2.20	58	2.80	67	–	–
2	64	0.39	49	0.78	75	1.29	96	1.48	106
	128	1.83	52	2.98	71	3.41	74	–	–
3	64	0.60	62	4.80	190	2.02	120	98.31	891
	128	2.10	55	441.59	933	5.39	92	–	–
4	64	0.36	46	0.75	70	0.69	66	4.77	192
	128	1.72	49	2.40	62	3.48	73	287.56	740
5	64	0.24	36	0.64	68	0.64	65	7.17	239
	128	1.50	46	2.13	59	2.18	57	308.31	780
6	64	0.20	32	0.49	58	0.56	60	2.12	127
	128	0.62	25	0.63	28	1.81	51	–	–
7	64	0.31	43	0.61	66	0.93	80	5.54	211
	128	1.33	41	2.20	60	3.26	70	–	–
8	64	0.31	40	0.83	74	0.88	73	5.84	205
	128	1.28	39	2.50	61	3.10	68	–	–

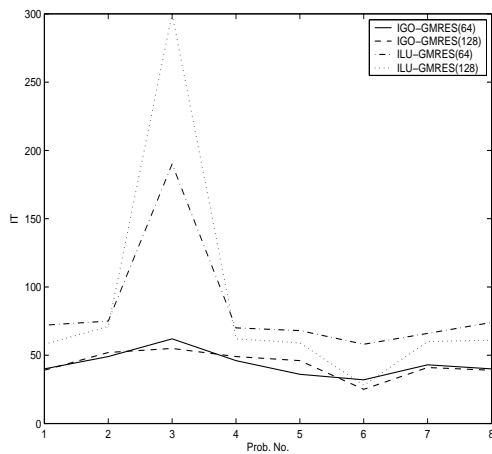


Figure 6.1: Curves of IT versus problem for GMRES when $q = 500$

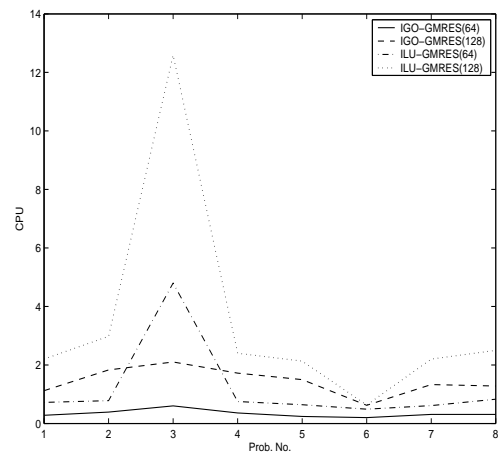


Figure 6.2: Curves of CPU versus problem for GMRES when $q = 500$

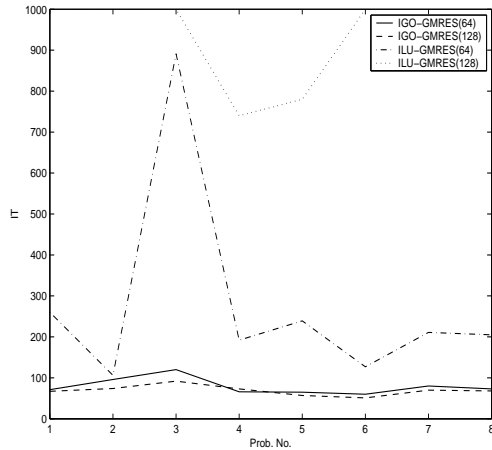


Figure 6.3: Curves of IT versus problem for GMRES when $q = 1000$

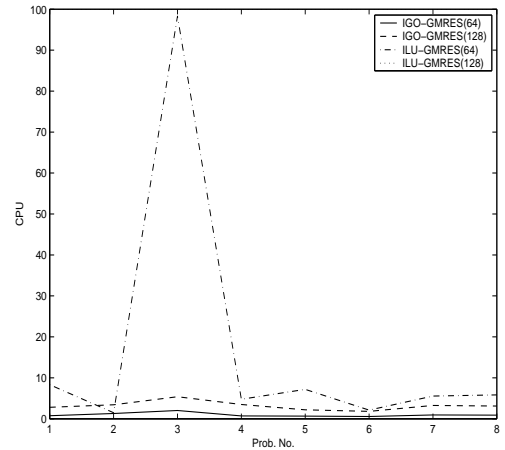


Figure 6.4: Curves of CPU versus problem for GMRES when $q = 1000$

The above observations are further confirmed by the curves of the relative residual norm (in the natural logarithm) versus the total iteration number plotted in Figures 6.5 and 6.6 for Problem 7, when $N = 128$ and $q = 500, 1000$, respectively.

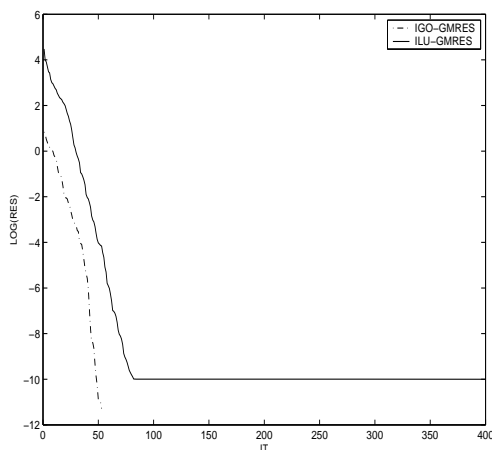


Figure 6.5: Curves of the relative residual norm (in the natural logarithm) versus the number of iterations for Problem 7 when $N = 128$ and $q = 500$

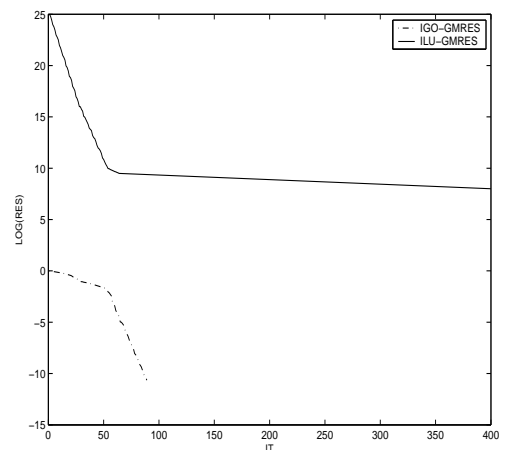


Figure 6.6: Curves of the relative residual norm (in the natural logarithm) versus the number of iterations for Problem 7 when $N = 128$ and $q = 1000$

Instead of GMRES, we can also use BiCGSTAB preconditioned by the IGO and the ILU methods to solve this class of linear systems.

In Table 6.3, we list the total iteration steps and the total CPU times for IGO-BiCGSTAB and ILU-BiCGSTAB, when $N = 64, 128$ and when $q = 500, 1000$, respectively. The data show similar numerical phenomenon to those in Table 6.2. From Table 6.3, we can

see that the IGO-BiCGSTAB iteration converges very quickly with high accuracy to the exact solution x_* for most cases, except for Problems 2 and 3 when $N = 64$ and $q = 1000$. However, the ILU-BiCGSTAB iteration either converges slowly in general, or even diverges for some cases of $q = 500$ and for most cases of $N = 128$ and $q = 1000$. For the convergent iterations, the IGO-BiCGSTAB iteration is often much faster than the ILU-BiCGSTAB iteration especially for Problems 1, 3, 7 and 8, and the number of iteration steps of the former is also less than that of the latter. In addition, the number of iterations decreases but the CPU time increases with an increasing number of discretization points, and these two quantities increase with increasing q .

Table 6.3: IT and CPU for BiCGSTAB when $N = 64, 128$ and when $q = 500, 1000$

Prob. No.	N	$q = 500$				$q = 1000$			
		IGO-BiCGSTAB		ILU-BiCGSTAB		IGO-BiCGSTAB		ILU-BiCGSTAB	
		CPU	IT	CPU	IT	CPU	IT	CPU	IT
1	64	0.14	27	0.29	83	0.53	102	–	–
	128	0.70	31	1.70	107	1.41	62	3.19	202
2	64	0.22	42	0.2	60	–	–	–	–
	128	0.82	36	0.84	53	1.44	63	–	–
3	64	0.30	57	–	–	–	–	–	–
	128	1.01	43	3.03	186	3.37	143	–	–
4	64	0.17	34	0.14	41	0.39	73	–	–
	128	0.91	40	0.77	49	1.37	60	2.41	153
5	64	0.14	25	0.14	40	0.53	102	–	–
	128	0.89	39	0.84	53	1.14	50	1.93	122
6	64	0.13	23	0.13	37	0.37	71	–	–
	128	0.43	19	0.46	29	0.94	41	2.35	149
7	64	0.14	28	0.25	73	0.26	49	–	–
	128	0.68	30	1.52	97	1.33	58	3.70	234
8	64	0.14	28	0.27	78	0.54	104	–	–
	128	0.72	32	1.41	90	1.43	63	3.75	237

We plot the curves of IT and CPU with respect to the eight test problems given in Table 5.1 for both IGO-BiCGSTAB and ILU-BiCGSTAB in Figures 6.7 and 6.8 when $q = 500$ and in Figures 6.9 and 6.10 when $q = 1000$, respectively. In an analogous fashion, to identify the matrix sizes we use IGO-BiCGSTAB(N) and ILU-BiCGSTAB(N) to denote IGO-BiCGSTAB and ILU-BiCGSTAB corresponding to the number $m = N \times N$ of discretization points. These figures show that both curves are very flat and smooth for the IGO-BiCGSTAB iteration but are very steep and oscillating for the ILU-BiCGSTAB iteration. It is evident that the practical IGO preconditioner is much more robust, stable

and efficient than the standard ILU preconditioner when it is used to precondition the BiCGSTAB method.

The above observations are further confirmed by the curves of the relative residual norm (in the natural logarithm) versus the total number of iterations plotted in Figures 6.11 and 6.12 for Problem 7, when $N = 128$ and $q = 500, 1000$, respectively.

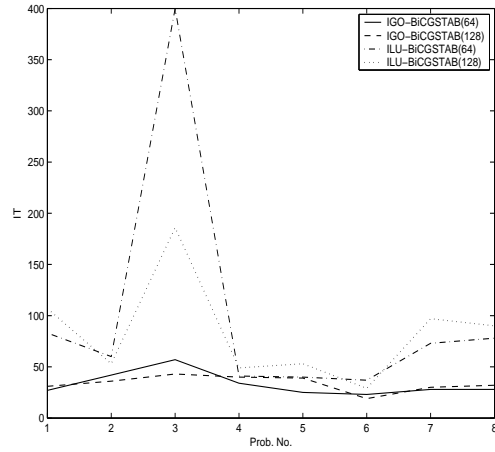


Figure 6.7: Curves of IT versus problem for BiCGSTAB when $q = 500$

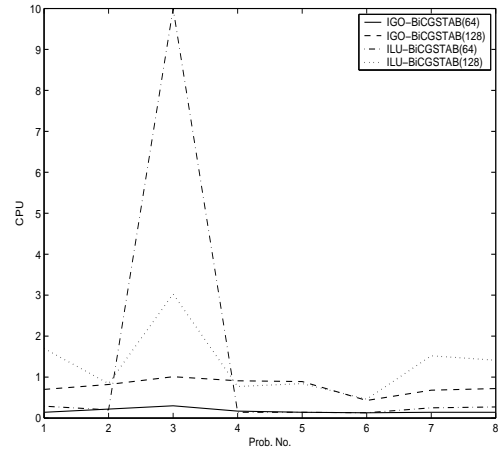


Figure 6.8: Curves of CPU versus problem for BiCGSTAB when $q = 500$

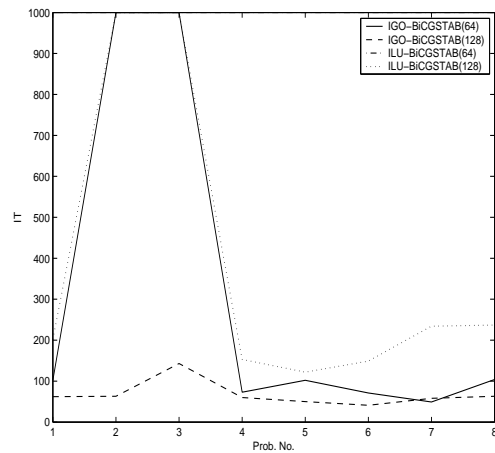


Figure 6.9: Curves of IT versus problem for BiCGSTAB when $q = 1000$

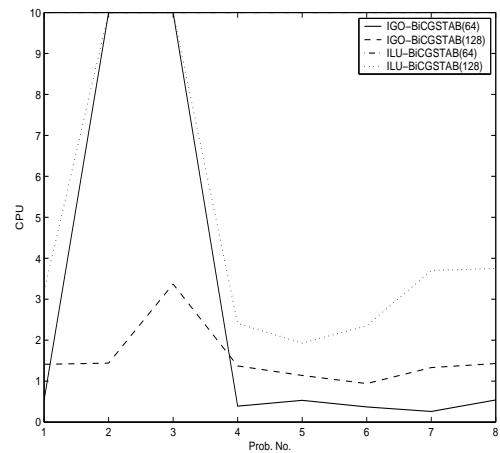


Figure 6.10: Curves of CPU versus problem for BiCGSTAB when $q = 1000$

6.2 Matrices from Matrix Market

The numerical results in this subsection concern IGO-GMRES and ILU-GMRES for solving systems of linear equations with coefficient matrices from the *Matrix Market*; see Table 5.2.

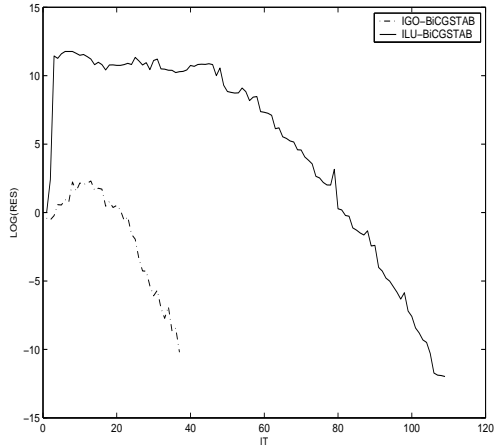


Figure 6.11: Curves of the relative residual norm (in the natural logarithm) versus the number of iterations for Problem 7 when $N = 128$ and $q = 500$

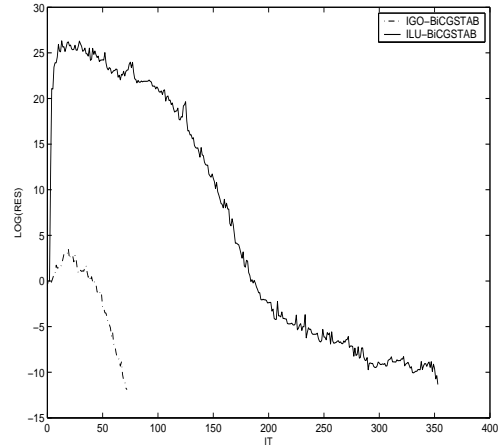


Figure 6.12: Curves of the relative residual norm (in the natural logarithm) versus the number of iterations for Problem 7 when $N = 128$ and $q = 1000$

In Table 6.4, we list the preconditioning time, the iteration time, the total number of iteration steps and the total CPU time for IGO-GMRES and ILU-GMRES. From this table, we can see that the construction times for the practical IGO and the standard ILU preconditioners are quite comparable, but the execution times of the IGO-preconditioned GMRES iterations are much less than those of the ILU-preconditioned GMRES iterations. In general, IGO-GMRES outperforms ILU-GMRES in total computing time and total number of iterations. Note that ILU-GMRES fails to compute acceptable approximate solutions for the three systems of linear equations with the coefficient matrices CAN_445, CAN_838 and NNC1374, as these matrices are more difficult than the others. This evidently shows that the practical IGO method can lead to more robust, accurate and efficient preconditioners than the standard ILU method for the GMRES method.

We plot the curves of IT and CPU with respect to the twenty test matrices given in Table 5.2 for both IGO-GMRES and ILU-GMRES in Figures 6.13 and 6.14, respectively. These figures show that both kinds of curves may be steep and oscillating for the IGO-GMRES and the ILU-GMRES iterations, but those for the latter behave worse than those for the former. Evidently, the practical IGO preconditioner is more robust, stable and efficient than the standard ILU preconditioner when they are employed to precondition the GMRES method.

The above observations are further illustrated by the curves of the relative residual norm (in the natural logarithm) versus the total number of iterations plotted in Figures 6.15 and 6.16 for matrices NOS1 and NOS2, respectively. It is clear that IGO-GMRES converges much more rapidly and accurately than ILU-GMRES.

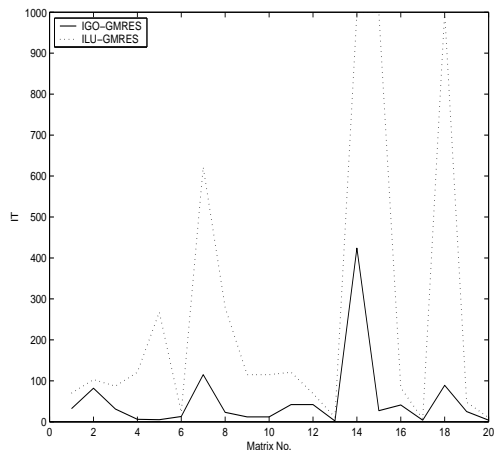


Figure 6.13: Curves of IT versus matrix for the matrices in Table 6.4 for GMRES

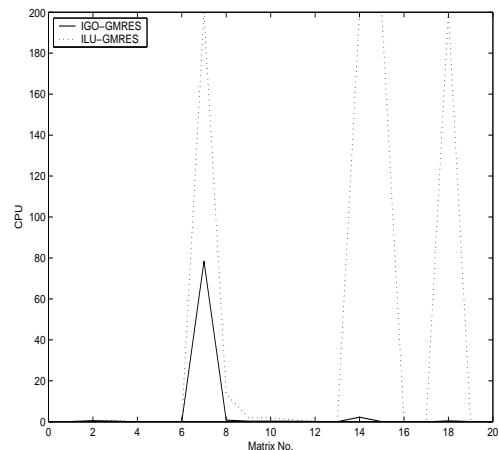


Figure 6.14: Curves of CPU versus matrix for the matrices in Table 6.4 for GMRES

6.3 The Approach of Normal Equations

The preconditioned *conjugate gradient* (CG) method is usually the best solver for the normal equations (1.2), giving rise to the CGNR (conjugate gradient on normal residual) method for computing the least-squares solution of the rectangular system of linear equations (1.1). We remark that it can also be used for square systems of linear equations. As we have mentioned before, the practical IGO method can be used to produce an effective incomplete Cholesky factorization and, thereby, lead to a high-quality preconditioner for the symmetric positive definite matrix $A^T A$. In fact, if $A = Q_{inc} R_{inc}$ is the practical IGO factorization of the matrix $A \in \mathcal{R}^{m \times n}$, with $Q_{inc} \in \mathcal{R}^{m \times n}$ and $R_{inc} \in \mathcal{R}^{n \times n}$, then we can precondition the matrix $A^T A$ by

$$M := R_{inc}^T R_{inc} = (Q_{inc} R_{inc})^T (Q_{inc} R_{inc}) \approx A^T A.$$

Of course, the standard ILU factorization $M = L_{inc} U_{inc}$ can also be used to precondition the CGNR method when the matrix $A \in \mathcal{R}^{m \times n}$ is square, i.e., $m = n$. This is often achieved by applying the CG method to the preconditioned linear system

$$(A(L_{inc} U_{inc})^{-1})^T (A(L_{inc} U_{inc})^{-1}) y = (A(L_{inc} U_{inc})^{-1})^T b,$$

with $y = (L_{inc} U_{inc}) x$. This kind of preconditioned CGNR method is also termed as the preconditioned CGLS method in [15, page 288]; see also [21].

The numerical results in this subsection concern IGO-CGNR and ILU-CGNR for solving systems of linear equations arising from the centred or the upwind finite-difference discretization of the convection-diffusion equation (5.1); see Table 5.1.

In Table 6.5, for $N = 32$ and 64 and for q ranging from 100 to 800, we list the number of iterations and the CPU times for IGO-CGNR and ILU-CGNR, when they are used to solve systems of linear equations whose coefficient matrices are obtained by

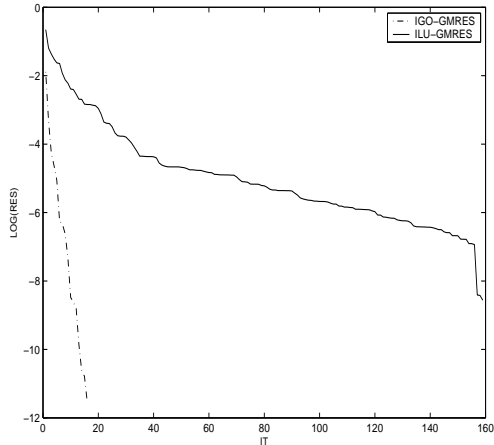


Figure 6.15: Curves of the relative residual norm (in the natural logarithm) versus the number of iterations for NOS1

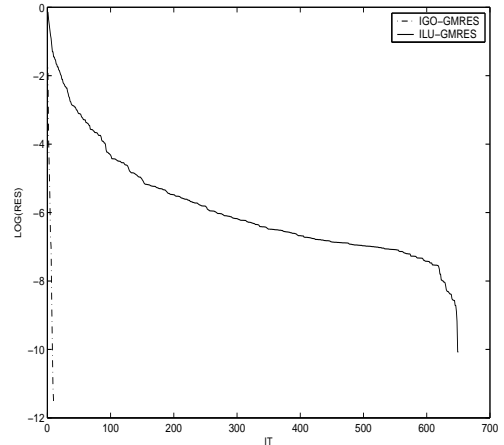


Figure 6.16: Curves of the relative residual norm (in the natural logarithm) versus the number of iterations for NOS2

the centred finite-difference discretization scheme. Clearly, when q is growing, the IGO-CGMR iteration converges very fast and accurately to the exact solution x_* for all problems except for Problem 7, and both the number of iterations and the CPU time decrease very quickly. However, the ILU-GMRES iteration converges very slowly for most of the problems when q is small, and it even diverges for all test problems when q is large. Hence, the practical IGO preconditioner is much more robust, stable and efficient than the standard ILU preconditioner when they are used to precondition the CGMR method.

There are other efficient incomplete factorization preconditioning methods for solving linear least-squares problems. For example, the *robust incomplete factorization* (**RIF**) in [14] and the *compressed incomplete modified Gram-Schmidt* (**CIMGS**) procedure in [37]. For numerical implementations and comparisons of the CGMR method preconditioned by IGO, RIF and CIMGS, we refer to [38, 14, 22]. However, we include some numerical results on systems of linear equations arising from the upwind finite-difference discretization of the convection-diffusion equation (5.1) in Table 6.6. These test problems are defined in Table 5.1. Clearly, as a preconditioner to CGMR, IGO outperforms RIF and CIMGS for both number of iterations and CPU time.

7 Conclusions

We have modified and tested numerically a class of incomplete orthogonal factorization preconditioners based on Givens rotations. Our numerical experiments have shown that these practical preconditioners outperform the standard incomplete triangular factorization preconditioners on aspects of solution accuracy, computation time and number of iterations when they are used to accelerate the Krylov subspace iteration methods

such as GMRES and BiCGSTAB for several problems. In particular, the incomplete orthogonal factorization preconditioners can produce incomplete Cholesky factorization preconditioners of high-quality for the CGNR methods.

The Givens-updating rule introduced in Section 2 can be analogously applied to modify other IGO-type methods such as GIGO, TIGO(τ) and GTIGO(τ, p) established in [5], and the theoretical analyses about the correspondingly obtained practical IGO-type methods can be done in a similar fashion to Theorem 2.1.

Further research may be carried out on dropping strategies and ordering techniques for the incomplete orthogonal factorization methods, which can take into account both numerical values and sparse structures of the matrices, as well as numerical comparisons with other competitive splitting preconditioners such as HSS [7], NSS [8] and BTSS [6]. Also, efficient parallel implementation of this class of incomplete orthogonal factorization preconditioners is an important topic that needs in-depth study on algorithmic design, theoretical analysis, and practical applications; see [38, 39].

Acknowledgements: The authors are very thankful to Andrew Wathen from Oxford for many valuable suggestions and thorough modifications which greatly improved the original manuscript of this paper. They are very much indebted to the referees and the editor, Lothar Reichel, for their useful comments.

References

- [1] M. Arioli, D. Loghin and A.J. Wathen, Stopping criteria for iterations in finite element methods, *Numer. Math.* **99** (2005) 381–410.
- [2] O. Axelsson, Iterative Solution Methods, *Cambridge University Press*, Cambridge, 1997.
- [3] Z.-Z. Bai, Sharp error bounds of some Krylov subspace methods for non-Hermitian linear systems, *Appl. Math. Comput.* **109** (2000) 273–285.
- [4] Z.-Z. Bai, Splitting iteration methods for non-Hermitian positive definite systems of linear equations, *Hokkaido Math. J.* **36** (2007) 801–814.
- [5] Z.-Z. Bai, I.S. Duff and A.J. Wathen, A class of incomplete orthogonal factorization methods. I: methods and theories, *BIT Numer. Math.* **41** (2001) 53–70.
- [6] Z.-Z. Bai, G.H. Golub, L.-Z. Lu and J.-F. Yin, Block triangular and skew-Hermitian splitting methods for positive-definite linear systems, *SIAM J. Sci. Comput.* **26** (2005) 844–863.
- [7] Z.-Z. Bai, G.H. Golub and M.K. Ng, Hermitian and skew-Hermitian splitting methods for non-Hermitian positive definite linear systems, *SIAM J. Matrix Anal. Appl.* **24** (2003) 603–626.

- [8] Z.-Z. Bai, G.H. Golub and M.K. Ng, On successive-overrelaxation acceleration of the Hermitian and skew-Hermitian splitting iterations, *Numer. Linear Algebra Appl.* **14** (2007) 319–335.
- [9] Z.-Z. Bai and C.-H. Jin, Column-decomposed relaxation methods for the overdetermined systems of linear equations, *Intern. J. Appl. Math.* **13** (2003) 71–82.
- [10] Z.-Z. Bai, G.-Q. Li and L.-Z. Lu, Combinative preconditioners of modified incomplete Cholesky factorization and Sherman-Morrison-Woodbury update for self-adjoint elliptic Dirichlet-periodic boundary value problems, *J. Comput. Math.* **22** (2004) 833–856.
- [11] Z.-Z. Bai, J.-F. Yin and Y.-F. Su, A shift-splitting preconditioner for non-Hermitian positive definite matrices, *J. Comput. Math.* **24** (2006) 539–552.
- [12] Z.-Z. Bai and S.-L. Zhang, A regularized conjugate gradient method for symmetric positive definite system of linear equations, *J. Comput. Math.* **20** (2002) 437–448.
- [13] M. Benzi and M. Tuma, Ordering for factorized sparse approximate inverse preconditioners, *SIAM J. Sci. Comput.* **21** (2000) 1851–1868.
- [14] M. Benzi and M. Tuma, A robust preconditioner with low memory requirements for large sparse least squares problems, *SIAM J. Sci. Comput.* **25** (2003) 499–512.
- [15] A. Björck, Numerical Methods for Least Squares Problems, *SIAM*, Philadelphia, 1996.
- [16] I.S. Duff, A.M. Erisman and J.K. Reid, Direct Methods for Sparse Matrices, *Oxford University Press*, London, 1986.
- [17] T. Dupont, R. Kendall and H.H. Rachford, Jr., An approximate factorization procedure for solving selfadjoint elliptic difference equations, *SIAM J. Numer. Anal.* **5** (1968) 559–573.
- [18] H.C. Elman, Relaxed and stabilized incomplete factorizations for non-self-adjoint linear systems, *BIT Numer. Math.* **29** (1989) 890–915.
- [19] G.H. Golub and C.F. Van Loan, Matrix Computations, The 3rd Edition, *The Johns Hopkins University Press*, Baltimore and London, 1996.
- [20] I. Gustafsson, A class of first order factorization methods, *BIT Numer. Math.* **18** (1978) 142–156.
- [21] K. Hayami and T. Ito, Solutions of least squares problems using generalized minimal residual methods, *Proc. Inst. Statist. Math.* **53** (2005) 331–348. (In Japanese)

- [22] K. Hayami, J.-F. Yin and T. Ito, GMRES Methods for Least Squares Problems, *NII Technical Report*, **NII-2007-009E**, National Institute of Informatics, Tokyo, Japan, 2007.
- [23] M.R. Hestenes and E.L. Stiefel, Methods of conjugate gradients for solving linear systems, *J. Research National Bureau Standards, Section B* **49** (1952) 409–436.
- [24] T.A. Manteuffel, An incomplete factorization technique for positive definite linear systems, *Math. Comput.* **34** (1980) 473–497.
- [25] Matrix Market, <http://math.nist.gov/MatrixMarket/data/>, *Test Matrices Database Maintained by NIST*, Gaithersburg, MD.
- [26] J.A. Meijerink and H.A. Van der Vorst, An iterative solution method for linear systems of which the coefficient matrix is a symmetric M -matrix, *Math. Comput.* **31** (1977) 148–162.
- [27] J.A. Meijerink and H.A. Van der Vorst, Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems, *J. Comp. Phys.* **44** (1981) 134–155.
- [28] P. Saylor, Second order strongly implicit symmetric factorization methods for the solution of elliptic difference equations, *SIAM J. Numer. Anal.* **11** (1974) 894–908.
- [29] H.L. Stone, Iterative solution of implicit approximations of multidimensional partial differential equations, *SIAM J. Numer. Anal.* **5** (1968) 530–558.
- [30] A.T. Papadopoulos, I.S. Duff and A.J. Wathen, A class of incomplete orthogonal factorization methods. II: implementation and results, *BIT Numer. Math.* **45** (2005) 159–179.
- [31] Y. Saad and M.H. Schultz, GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, *SIAM J. Sci. Stat. Comput.* **7** (1986) 856–839.
- [32] Y. Saad, Preconditioning techniques for nonsymmetric and indefinite linear systems, *J. Comput. Appl. Math.* **24** (1988) 89–105.
- [33] Y. Saad, ILUT: A dual threshold incomplete ILU factorization, *Numer. Linear Algebra Appl.* **1** (1994) 387–402.
- [34] Y. Saad, Iterative Methods for Sparse Linear Systems, *PWS Publishing Company*, Boston, 1996.
- [35] H.A. Van der Vorst, Iterative Krylov Methods for Large Linear Systems, *Cambridge University Press*, Cambridge, 2003.
- [36] R.S. Varga, Matrix Iterative Analysis, *Prentice-Hall*, Englewoods Cliffs, N.J., 1962.

- [37] X.-G. Wang, K.A. Gallivan and R. Bramley, CIMGS: An incomplete orthogonal factorization preconditioner, *SIAM J. Sci. Comput.* **18** (1997) 516–536.
- [38] T. Washio and K. Hayami, Parallel block preconditioning based on SSOR and MILU, *Numer. Linear Algebra Appl.* **1** (1994) 533–553.
- [39] T. Washio and K. Hayami, Overlapped multicolor MILU preconditioning, *SIAM J. Sci. Comput.* **16** (1995) 636–650.
- [40] J.-F. Yin and Z.-Z. Bai, The restrictively preconditioned conjugate gradient methods on normal residual for block two-by-two linear systems, *J. Comput. Math.* **26** (2008) 240–249.
- [41] D.M. Young, Iterative Solution of Large Linear Systems, *Academic Press*, New York, 1971.

Table 6.4: IT and CPU for GMRES for Matrices in Table 5.2

Matrix No.	IGO-GMRES				ILU-GMRES			
	CPU _p	CPU _{it}	IT	CPU	CPU _p	CPU _{it}	IT	CPU
Symmetric Positive Definite Matrices								
BCSSTK10	0.02	0.10	32	0.12	0.02	0.19	70	0.21
BCSSTK12	0.03	0.53	82	0.56	0.03	0.54	103	0.57
BCSSTK14	0.10	0.21	31	0.31	0.08	0.54	87	0.62
NOS1	0.00	0.00	6	0.00	0.00	0.11	121	0.11
NOS2	0.00	0.00	5	0.00	0.00	1.93	268	0.00
NOS6	0.00	0.01	13	0.01	0.00	0.02	23	0.01
S3DKT3M2	6.05	72.36	115	78.41	4.74	1349.69	622	1354.43
S3RMT3M1	0.35	0.50	23	0.85	0.28	13.24	279	13.52
Symmetric Indefinite Matrices								
BCSSTK19	0.16	0.14	12	0.30	0.14	1.85	115	1.99
BCSSTK22	0.16	0.14	12	0.30	0.14	1.85	115	1.99
BCSSTK26	0.02	0.22	42	0.24	0.02	0.91	121	0.93
Symmetric Positive Semidefinite Matrices								
BCSSTM10	0.02	0.13	42	0.15	0.02	0.19	69	0.21
BCSSTM12	0.02	0.01	2	0.03	0.02	0.02	11	0.04
Pattern Symmetric Indefinite Matrices								
CAN_445	0.00	2.52	424	2.52	–	–	–	–
CAN_838	0.01	0.04	27	0.05	–	–	–	–
Nonsymmetric Matrices								
CDDE5	0.00	0.07	41	0.07	0.00	0.22	83	0.22
MCCA	0.00	0.00	4	0.00	0.00	0.01	7	0.01
NNC1374	0.00	0.43	89	0.43	–	–	–	–
PDE2961	0.01	0.09	25	0.10	0.01	0.23	48	0.24
SHERMAN2	0.02	0.01	4	0.03	0.02	0.02	10	0.04

Table 6.5: IT and CPU for CGNR for various q when $N = 32$ and 64

Prob. No.	N		$q = 100$		$q = 200$		$q = 400$		$q = 600$		$q = 800$	
			IT	CPU	IT	CPU	IT	CPU	IT	CPU	IT	CPU
1	32	IGO-CGNR	50	0.03	16	0.01	9	0.01	7	0.01	6	0.01
		ILU-CGNR	49	0.01	48	0.03	–	–	–	–	–	–
	64	IGO-CGNR	162	0.37	89	0.21	17	0.05	10	0.03	9	0.03
		ILU-CGNR	211	0.65	88	0.24	102	0.28	–	–	–	–
2	32	IGO-CGNR	61	0.04	32	0.02	19	0.01	12	0.01	11	0.01
		ILU-CGNR	–	–	177	0.13	197	0.17	–	–	–	–
	64	IGO-CGNR	162	0.37	106	0.25	58	0.14	36	0.10	30	0.07
		ILU-CGNR	–	–	–	–	–	–	–	–	–	–
3	32	IGO-CGNR	35	0.02	12	0.01	8	0.01	6	0.01	6	0.01
		ILU-CGNR	43	0.03	–	–	–	–	–	–	–	–
	64	IGO-CGNR	131	0.31	56	0.14	13	0.04	9	0.03	8	0.02
		ILU-CGNR	148	0.40	75	0.21	–	–	–	–	–	–
4	32	IGO-CGNR	43	0.02	15	0.01	8	0.01	7	0.01	6	0.01
		ILU-CGNR	43	0.03	28	0.02	–	–	–	–	–	–
	64	IGO-CGNR	149	0.35	71	0.17	16	0.04	10	0.04	8	0.02
		ILU-CGNR	171	0.46	76	0.21	49	0.14	191	0.63	–	–
5	32	IGO-CGNR	43	0.03	14	0.01	8	0.01	7	0.01	6	0.01
		ILU-CGNR	43	0.03	28	0.02	–	–	–	–	–	–
	64	IGO-CGNR	150	0.35	71	0.17	15	0.05	10	0.04	8	0.02
		ILU-CGNR	171	0.46	76	0.21	49	0.14	191	0.63	–	–
6	32	IGO-CGNR	61	0.04	30	0.02	11	0.01	8	0.01	7	0.01
		ILU-CGNR	81	0.05	39	0.02	–	–	–	–	–	–
	64	IGO-CGNR	179	0.42	119	0.28	45	0.11	16	0.04	12	0.04
		ILU-CGNR	–	–	191	0.54	72	0.20	109	0.30	–	–
7	32	IGO-CGNR	–	–	–	–	–	–	–	–	–	–
		ILU-CGNR	–	–	–	–	–	–	–	–	–	–
	64	IGO-CGNR	–	–	–	–	–	–	–	–	–	–
		ILU-CGNR	–	–	–	–	–	–	–	–	–	–
8	32	IGO-CGNR	35	0.02	12	0.01	8	0.01	6	0.01	6	0.01
		ILU-CGNR	40	0.03	34	0.02	–	–	–	–	–	–
	64	IGO-CGNR	116	0.27	59	0.14	14	0.04	9	0.02	8	0.02
		ILU-CGNR	131	0.36	70	0.19	65	0.18	–	–	–	–

Table 6.6: IT and CPU for CGNR preconditioned by IGO, RIF and CIMGS for various q when $N = 32$ and 64

Prob. NO.	m		$q = 100$		$q = 200$		$q = 400$		$q = 600$		$q = 800$	
			IT	CPU	IT	CPU	IT	CPU	IT	CPU	IT	CPU
1	32	IGO	36	0.02	35	0.02	34	0.02	34	0.02	34	0.02
		RIF	41	0.02	40	0.02	40	0.02	39	0.02	32	0.02
		CIMGS	64	0.03	64	0.03	64	0.03	64	0.03	64	0.03
	64	IGO	62	0.53	61	0.53	60	0.52	60	0.52	60	0.52
		RIF	77	0.59	77	0.59	76	0.59	76	0.59	74	0.58
		CIMGS	128	0.63	128	0.63	128	0.63	128	0.63	128	0.63
3	32	IGO	37	0.02	37	0.02	37	0.02	37	0.02	37	0.02
		RIF	44	0.02	43	0.02	42	0.02	41	0.02	35	0.02
		CIMGS	65	0.03	65	0.03	64	0.03	63	0.03	63	0.03
	64	IGO	77	0.53	77	0.53	76	0.52	76	0.52	75	0.52
		RIF	79	0.59	78	0.59	77	0.59	77	0.59	77	0.59
		CIMGS	130	0.63	130	0.63	129	0.63	127	0.63	127	0.63
4	32	IGO	34	0.02	33	0.02	33	0.02	33	0.02	33	0.02
		RIF	44	0.02	43	0.02	42	0.02	41	0.02	35	0.02
		CIMGS	65	0.03	65	0.03	65	0.03	63	0.03	63	0.03
	64	IGO	54	0.52	54	0.52	54	0.52	54	0.52	53	0.52
		RIF	79	0.59	78	0.59	77	0.59	77	0.59	77	0.59
		CIMGS	130	0.63	130	0.63	130	0.63	127	0.63	127	0.63
8	32	IGO	34	0.02	33	0.02	33	0.02	33	0.02	33	0.02
		RIF	44	0.02	43	0.02	42	0.02	42	0.02	41	0.02
		CIMGS	64	0.03	64	0.03	63	0.03	59	0.03	57	0.03
	64	IGO	54	0.52	54	0.52	54	0.52	53	0.52	53	0.52
		RIF	79	0.59	78	0.59	78	0.59	77	0.59	77	0.59
		CIMGS	128	0.63	128	0.63	120	0.63	113	0.63	108	0.62