2ND INTERNATIONAL WORKSHOP ON THE NUMERICAL SOLUTION OF MARKOV CHAINS

1

PRECONDITIONED KRYLOV SUBSPACE METHODS FOR THE NUMERICAL SOLUTION OF MARKOV CHAINS

Yousef Saad

University of Minnesota

Department of Computer Science

Minneapolis, Minnesota

ABSTRACT

In a general projection technique the original matrix problem of size N is approximated by one of dimension m, typically much smaller than N. A particularly successful class of techniques based on this principle is that of Krylov subspace methods which utilize subspaces of the form $span\{v, Av, ..., A^{m-1}v\}$. This general principle can be used to solve linear systems and eigenvalue problems which arise when computing stationary probability distributions of Markov chains. It can also be used to approximate the product of the exponential of a matrix by a vector as occurs when following the solutions of transient models. In this paper we give an overview of these ideas and discuss preconditioning techniques which constitute an essential ingredient in the success of Krylov subspace methods.

1 INTRODUCTION

Direct methods have often been preferred to iterative methods when solving sparse linear systems of equations such as those that arise when computing the stationary probability distribution of Markov chains. The main reason for this is that direct methods have the advantage of being 'predictably reliable'. However, iterative methods are currently gaining ground because of recent improvements in their robustness on the one hand, and of the increasing sizes of the matrices that must be tackled, on the other. Relaxation type techniques, such as Gauss-Seidel or SSOR, can be quite successful and have been a popular alternative to direct solvers. The performance of these techniques is difficult to predict and is dependent on an optimal acceleration parameter ω, which is not easy to determine. A class of methods that can be quite efficient is that of Krylov subspace techniques. This class includes the conjugate gradient method and the Lanczos algorithm as well as the Bi-CG and related algorithms, many of which have been developed in the last few years.

An essential ingredient in the success of projection type methods is the *preconditioner*. Preconditioning amounts to transforming the original linear system into one which would require fewer steps to converge from the Krylov projection method. Among the many possible options available to precondition a system, are to use a standard iterations, e.g., SOR, or one of several known 'incomplete LU' factorizations of the matrix [15]. In this context, the standard and inexpensive approaches, such as ILU(0), or SSOR, have a great rate of failure for the harder, nearly decomposable cases. However, alternatives based on the more accurate factorizations, such as ILUT [22] yield more robust techniques. In terms of overall cost, the more accurate ILU factorizations are usually less expensive. However, they do require more storage, since denser LU factors must be saved.

Krylov subspace methods can also be used to compute transient solutions of Markov chain models. The problem here is to solve the simple system of Ordinary Differential Equations $y' = Qy$, whose exact solution $y(t) = exp(Qt)y_0$ can be approximated using a Krylov subspace approach. Thus, the Krylov subspaces are used to approximate the exponential propagator $exp(Q\Delta t)y$ directly [21, 10]. This approach has been shown to be quite successful in a recent paper [16].

In this paper we will give an overview of preconditioned Krylov subspace methods. We will emphasize the preconditioning techniques since these are currently the most critical component of a preconditioned Krylov approach. The ideas presented herein constitute only a brief introduction in the most part. For details, the reader is referred to the recent literature, in particular, see e.g., [31], [25, 23].

2 PRECONDITIONED KRYLOV SUBSPACE METHODS

Iterative techniques based on Krylov subspace projection coupled with suitable preconditioners are currently considered to be the best compromise between efficiency and robustness in solving general sparse linear systems. In addition to their advantage over direct methods, in terms of memory and computational cost, these methods are also attractive because of the simplicity with which they can be adapted to high performance computers. There are two ingredients in the use of a preconditioned Krylov subspace approach namely the *accelerator* and the *preconditioner.*

2.1 Accelerators

What is referred to as an *accelerator* is typically a projection – type method on a subspace of dimension > 1. Given two subspaces K_m and L_m of the same dimension m, a projection method onto K_m and orthogonally to L_m seeks an approximation to the linear system

$$Ax = b \tag{1.1}$$

in the form $x = x_0 + \delta$ where δ belongs to K_m and the following Petrov-Galerkin condition is satisfied:

$$(b - Ax) \perp L_m \ . \tag{1.2}$$

Since the two subspaces are of the same dimension there are as many degrees of freedom as there are constraints and there is, in general, one and only one approximate solution. In Krylov subspace techniques, the subspace K_m is a Krylov subspace, i.e., a subspace of the form,

$$K_m(A, v) = \text{span}\{v, Av, \ldots, A^{m-1}v\} \tag{1.3}$$

Often the vector v is obtained from scaling the initial residual $r_0 = b - Ax_0$. In addition, L_m is either another Krylov subspace or a related subspace. Specifically, the most popular choices of K_m and L_m are the following.

Orthogonal Projection Methods: $L_m = K_m = K_m(A, r_0)$. This is the orthogonal projection or Galerkin case. A method in this class is the Full Orthogonalization Method (FOM) [18] which is closely related to Arnoldi's method for solving eigenvalue problems [1]. Also in this class is ORTHORES [12], a method that is mathematically equivalent to FOM. Axelsson [2] also derived an algorithm of this class for general nonsymmetric matrices. When A is symmetric positive definite, it can be shown that the approximate solution x_m minimizes the A-norm of the error vector $A^{-1}b-x$ over all candidate vectors x in $x_0 + K_m$. In this case, FOM is mathematically to the conjugate gradient method.

Minimum Residual methods: $L_m = AK_m; K_m = K_m(A, r_0)$**.** With this choice of L_m, it can be shown, see e.g., [26] that the approximate solution x_m minimizes the residual norm $||b - Ax||_2$ over all candidate vectors in $x_0 + K_m$. In contrast, there is no similar optimality property known for methods of the first class when A in nonsymmetric. Because of this, many methods of this type have been derived for the nonsymmetric case [3, 12, 7, 27]. The Conjugate Residual method [4] is the analogue of conjugate gradient method that is in this class. The GMRES algorithm [27] and the Generalized Conjugate Residual algorithms [7] is an extension of the Conjugate Residual method to nonsymmetric problems.

Bi-conjugate gradient: $L_m = K_m(A^T, r_0); K_m = K_m(A, r_0)$**.** Clearly, in the symmetric case this class of methods reduces to the first one. In the nonsymmetric case, the biconjugate gradient method (BCG) due to Lanczos [13] and Fletcher [8] is a good representative of this class. There are various mathematically equivalent formulations of the biconjugate gradient method [19], some of which are more numerically viable than others. An efficient variation on this method, called CGS (Conjugate gradient squared) was proposed by Sonneveld [29, 17]. More recently, a number of methods based on the CGS principle have been developed, see for example the TFQMR algorithm [9] and the BiCGSTAB algorithm [32].

CGNR: $L_m = K_m = K_m(A^TA, A^Tr_0)$**.** This is nothing but the conjugate gradient method applied to the normal equations $A^TAx = A^Tb$, often referred to as CGNR. The condition number of the normal equations is likely to be too large for most problems to make this approach competitive with the approaches 1 to 3, except possibly for indefinite problems, i.e., problems for which the symmetric part is not positive definite. LSQR [14] is an implementation that is somewhat less sensitive to large condition numbers. Moreover, for least squares problems with non-square matrices, one must either explicitly or implicitly use an approach based on the normal equations. We put in this category also conjugate gradient method applied to $AA^Ty = b$, whose solution y is trivially related to x by $x = A^Ty$. This is often referred to as CGNE, or Craig's method. If we express the Galerkin conditions in terms of the y variable, then, clearly, $K_m = K_m(AA^T, r_0)$ and $L_m = K_m$. Using the relationship $x = A^Ty$

between the x and y variables, we can translate the Galerkin condition that y satisfies in terms of the x variable to find that for the variable x Craig's method corresponds to taking $K_m = K_m(A^T A, A^T r_0)$ and $L_m = A^{-T} K_m$. Moreover, the main difference between CGNR and CGNE is that the first minimizes the residual norm over K_m while the second minimizes the error norm over K_m.

2.2 Preconditioners

The second ingredient in a preconditioned Krylov subspace method is the "preconditioner". Typically, the original linear system (1.1) is preconditioned by, for example, transforming it into the "right-preconditioned" equivalent system

$$AM^{-1}(Mx) = b. \tag{1.4}$$

where the preconditioned matrix M has the property that it is not too expensive to compute $M^{-1}v$ for an arbitrary v. Thus, the system $AM^{-1}y = b$ is solved for the unknown $y \equiv Mx$, and the final x result is obtained through the post-transformation $x = M^{-1}y$. One can also use left-preconditioning,

$$M^{-1}Ax = M^{-1}b$$

which requires a pre-transformation of the right-hand-side, or the initial residual. The main practical difference between these two approaches is that in the right-preconditioned case, the actual residual norm is available at each step of the iterative process whereas in the second case only the preconditioned residual is explicitly available.

Here we will describe a method presented in [24] and known as FGMRES which can be used when the preconditioning operations varies from step to step, a property which can be very useful. This variant is derived by observing that in the last step of the standard GMRES algorithm [27], the approximate solution x_m is formed as

$$x_m = x_0 + \sum_{i=1}^{m} \alpha_i M^{-1} v_i$$

Here, the v_i's are the Arnoldi vectors, M the preconditioner, x_0 the initial guess and m the dimension of the Krylov subspace. This is a linear combination of the preconditioned vectors $z_i = M^{-1}v_i, i = 1, \ldots, m$. Since these vectors are all obtained by applying the same preconditioning matrix M^{-1} to the v's, we need not save them. We only need to apply M^{-1} to the linear combination of the $v's$. If the preconditioner varies at every step, then we need to save the 'preconditioned' vectors $z_i = M_i^{-1}v_i$ and use them instead of $M^{-1}v_i$ when computing the above linear linear combination. The resulting 'flexible' variant of GMRES is described below.

ALGORITHM **2.1 Flexible GMRES (FGMRES)**

```
1. Start:
   Choose $x_0$ and a dimension $m$ of the Krylov subspaces.
   Define $\bar{H}_m = \{h_{i,j} := 0\}_{i=1,\ldots,m+1;\ j=1,\ldots,m}\}$.
2. Arnoldi process:
   Compute $r_0 = b - Ax_0$, $\beta = ||r_0||_2$ and $v_1 = r_0/\beta$.
   For $j = 1, ..., m$ do
      Compute $z_j := M_j^{-1} v_j$
      Compute $w := Az_j$
      For $i = 1, \ldots, j$, do: Compute $h_{i,j} := (w, v_i)$ and $w := w - h_{i,j} v_i$
      Compute $h_{j+1,j} = ||w||_2$ and $v_{j+1} = w/h_{j+1,j}$.
   Enddo
   Define $Z_m := [z_1, ...., z_m]$.
3. Form the approximate solution:
   Compute $x_m = x_0 + Z_m y_m$ where
   $y_m = \text{argmin}_y ||\beta e_1 - \bar{H}_m y||_2$ and $e_1 = [1, 0, \ldots, 0]^T$.
4. Restart:
   If satisfied stop, else set $x_0 \leftarrow x_m$ and goto 2.
```

The Arnoldi loop simply constructs an orthogonal basis of the preconditioned subspace $\text{Span}\{v_1, AM_1^{-1}v_1, \ldots, AM_{m-1}^{-1}v_{m-1}\}$ by a modified Gram-Schmidt process, in which the new vector to be orthogonalized is defined from the previous vector in the process.

Note that if $M_j = M$ for $j = 1, \ldots, m$ then the method is equivalent to the standard GMRES algorithm, right-preconditioned with M. The approximate solution x_m obtained from this modified algorithm minimizes the residual norm $||b - Ax_m||_2$ over $x_0 + \text{Span}\{Z_m\}$, [24]. In addition, if at a given step k, we have $Az_k = v_k$ (i.e., if the preconditioning is 'exact' at step k) and if the $k \times k$ Hessenberg matrix $H_k = \{h_{ij}\}_{i,j=1,\ldots,k}$ is nonsingular then the approximation x_k is exact.

There are many possible applications of the added flexibility provided by FGMRES. In our context, we would like to be able to use any *subsidiary* iterative procedure such as SOR as a preconditioner. In fact the inner iteration can be any iteration, including a subsidiary GMRES or FOM iteration, for example.

It can also be a multilevel type approach or an iteration on a Gauss-Seidel iteration on a small part of the adjacency graph. One idea which we will test in the numerical experiments section is to use an Arnoldi process for preconditioning.

3 PRECONDITIONING TECHNIQUES

As was stated before a critical component in the success of iterative methods is the preconditioner. For a linear system that is poorly preconditioned the iterative process may require too many steps to converge and a direct solver may then be a better approach. Preconditioning a linear system is typically a difficult task, except in the traditional elliptic-dominated, one variable-per mesh point cases. Unfortunately, the best known preconditioning techniques have been developed mainly with these problems in mind. In this section we give an overview of standard preconditioners and present some alternatives based on Gaussian Elimination with drop-tolerance.

3.1 Relaxation based preconditioners

The simplest way of preconditioning a system is to take M to be simply the diagonal or block diagonal of A. This is referred to as Jacobi (or diagonal), or block Jacobi (block diagonal) preconditioning and is effective only in special cases, e.g., for transient solutions.

Consider the standard splitting of A into,

$$A = D - E - F$$

in which D is the diagonal of A, $-E$ its strict lower part, and $-F$ its strict upper part. The SOR preconditioning matrix is defined by

$$M_\omega = (D - \omega E)^{-1} .$$

This corresponds to performing a forward SOR sweep starting with the initial guess $x_0 = 0$. One can also use several steps of SOR as a preconditioning operation. Thus, for a general iteration of the form

$$x_{k+1} = Gx_k + f$$

the preconditioning matrix will take the form,

$$M_k = G^{k-1}f,$$

which again corresponds to performing k steps of the iteration starting with $x_0 = 0$. For example, the k-step SOR preconditioner is defined by

$$M_{k,\omega} = [(D - \omega E)^{-1} F]^{k-1} (D - \omega E)^{-1}$$

which corresponds to taking k steps of SOR with the initial vector $x_0 = 0$.

In many cases, the SSOR iteration, which consists of a forward SOR sweep followed by a backward SOR sweep, is actually more popular than SOR as a preconditioner. The main reason is that SSOR tends to preserve symmetry. The SSOR preconditioning matrix is given by,

$$M_\omega = (D - \omega E) D^{-1} (D - \omega F) \ .$$

In the late 60's and early 70's the idea came about to use an M which has the same form as above with $\omega = 1$ but with a D that is defined recursively to ensure that the diagonal elements of M and A are the same. This lead to the ILU(0) preconditioner in the special case of 5-point matrices.

3.2 ILU(0)

The idea of ILU(0) which was initially derived for special finite difference matrices, was later extended by taking the viewpoint of a Gaussian elimination in which *fill-ins* during the process are dropped. A fill-in element refers to a nonzero element introduced in the matrix which holds the LU factors, in a location where there was initially a zero element in the matrix A. Thus, if we denote by $NZ(A)$ the nonzero structure of A, i.e., the set of all pairs (i, j) such that $a_{ij} \neq 0$ then ILU(0) can be described as follows.

ALGORITHM **3.1** *ILU(0*

```
For $i = 1, \ldots, N$ Do:
    For $k = 1, \ldots, i-1$ and if $(i,k) \in NZ(A)$ Do:
        Compute $a_{ik} := a_{ik}/a_{kj}$
        For $j = k+1, \ldots$ and if $(i,j) \in NZ(A)$, Do:
            compute $a_{ij} := a_{ij} - a_{ik} a_{k,j}$.
        EndDo
    EndDo
EndDo
```

From an implementation viewpoint, this is an (i, k, j) version of Gaussian elimination [11] which is restricted to the $NZ(A)$ part of the matrix. This algorithm can be generalized to any preset nonzero pattern. In particular, one can classify the fill-ins by assigning them a level which is defined from the parents which generated the element in the elimination [33]. For diagonally dominant matrices, the higher the level-of-fill the smaller the element. Once the level of fill of each element is defined we can execute an algorithm similar to the one above, in which $NZ(A)$ is replaced by $NZ_p(A)$ which is the set of all elements whose level-of-fill does not exceed p. This defines the ILU(p) factorization.

3.3 ILUT

The elements that are dropped in the ILU(p) factorization technique outlined above depend only on the pattern of A and not on the values. The property exploited here is that the larger the level of fill, the smaller the elements, which results in the dropping of smaller elements in the ILU(p) factorization. However, this property is no longer true for non-diagonally dominant matrices. Another strategy altogether is to use the same general structure of the ILU factorization, namely the i, k, j variant of Gaussian Elimination, and to drop elements according to their magnitude. One such strategy defined in [22], was referred to as ILUT (ILU with threshold).

ALGORITHM **3.2** *ILUT(p, ϵ)*

```
For $i = 1, \ldots, N$ Do:
    Compute $\epsilon_i := \epsilon ||a_{i,:}||_2$
    For $k = 1, \ldots, i-1$ and if $a_{i,k} \neq 0$ Do:
        Compute $a_{ik} := a_{ik}/a_{kj}$
        If $|a_{ik}| \geq \epsilon_i$ Then
            For $j = k+1, \ldots$ Do:
                compute $a_{ij} := a_{ij} - a_{ik}a_{k,j}$.
                If $|a_{ij}| \leq \epsilon_i$ then $a_{ij} := 0$
            EndDo
        EndIf
        Keep $p$ largest elements in $L$-part of $a_{i,:}$
            and $p$ largest elements in $U$-part of $a_{i,:}$;
    EndDo
EndDo
```

An advantage of this algorithm is that the amount of fill-in that it gebnerates is controlled. When $\epsilon = 0$, then the higher the parameter p, the more accurate the factorization. Reordering for reducing fill-in can help improve the quality of the factorization, and we refer to [5] and [6] for similar experiments performed with the ILU(0) factorization.

4 EXPERIMENTS

We compared a few preconditioned Krylov subspace techniques on four test matrices generated by the software package MARCA [30]. These examples are variants of the ones used in the numerical experiments of [15]. All experiments have been performed on a Sparcstation 10 in double precision.

4.1 The test problems

The first example models the behavior of a multiprogrammed, time-shared, virtual memory computer system consisting of N terminals, a CPU, a secondary memory, and a filing device. This example was also used in [15]. However, one difference with the example in [15] is that we increased the number of users to $n = 30$ from $n = 20$. This gives rise to a matrix of size $5,456$. Our second matrix is also taken from [15] and was generated using Marca. This is a telecommunicatoin model which has been used to to determine the effect of impatient telephone customers. When a request is made by a customer for service the customer is prepared to wait for a certain period of time for a reply. If at the end of this period, the reply has not arrived, the customer may either give up and leave the network or else wait for some period of time before trying again. Finally, our third and fourth test matrices arise from yet another telecommunication example which models a single service center at which two identical servers provide service to two different classes of customer. Class-one customers are assumed to have a high priority. Once a server starts providing service to a class-two customer, it will continue to serve that customer even if a class-one customers arrives. The service rates may differ for each class but both are exponentially distributed. The arrival processes of the two classes is not exponential. In our first matrix we used a buffer size of 20 which yields a matrix of size $N = 3,060$ and then we used a buffer size of 30 which yields a matrix of size $n = 6,980$. The sizes and number of nonzero elements of all four matrices are shown in Table 1. For details on these problems and their origins, see [15] and the references therein.

Matrix	N	Nz
marc1	5456	35,216
marc2	2431	11,681
marc3	3060	20,320
marc4	6980	46,620

Table 1 Sizes and number of nonzero elements of the test matrices

4.2 Experiments with accelerators

We start by showing a comparison of a number of ILU(0) preconditioned accelerators. The results are shown in Table 2 for the following accelerators.

- CGNR; Equivalent to Conjugate Gradient applied to $A^T Ax = A^b$;
- Bi-Conjugate gradient (bcg in the table).
- DBCG. This is a Lanczos implementation of the biCG algorithm in which partial pivoting is applied when solving the tridiagonal systems, [20].
- BiCGSTAB. This a variant of CGS [29] developed by Van-der Vorst [32]. This is denoted by BCGST in the table.
- TFQMR. This is yet another variant of CGS which use quasi-minimization. The algorithm was developed by Freund [9].
- Full Orthogonalization Method (FOM), using a Krylov subspace of dimension 15.
- Generalized Minimum Residual method (GMRES) using a Krylov subspace of dimension 15.
- Direct Quasi-Generalized Minimum Residual method (DQGMRES). This consists of truncating the Arnoldi sequence during the orthogonalization. A quasi-minimization is then performed in the spirit of QMR [28]. In our example, we keep the 15 most recent Arnoldi directions. This is referred to as dqg in the table.

In Table 2 we show for each matrix the number of matrix–vector products and the total CPU time required to converge. The convergence criterion used was to reduce the residual norm by a factor of 10^{-5}. A maximum of 500 steps is allowed. Thus any data showing 500 steps means that the method did not achieve convergence.

As a first observation, we can note that the CGNR and BiCG-type methods (BCG, and DBCG) performed rather poorly compared with others. However, the more stable variants, namely BICGSTAB and TFQMR, performed much better. Interestingly, in other application areas (e.g., semi-conductor simulation) the converse is sometimes observed. DQGMRES is competive in the number of matrix-vector products required but, due to a larger number of inner products required, requires more CPU time. The other four methods, namely, TFQMR, BiCGSTAB, FOM, and GMRES are rather close to one another overall.

	cgn	**bcg**	**dbcg**	**bcst**	**tfqmr**	**fom**	**gmres**	**dqg**
marc1	500	357	500	66	70	70	61	66
	51.0	39.0	73.1	7.6	8.9	12.2	10.7	23.0
marc2	500	501	500	92	62	123	57	70
	17.8	19.2	26.8	3.7	2.9	8.2	3.8	10.1
marc3	500	29	30	28	20	15	15	15
	28.4	1.9	2.5	1.9	1.5	1.5	1.5	2.1
marc4	500	41	42	32	20	17	16	16
	67.7	6.1	8.2	5.0	3.5	4.0	3.9	5.5

Table 2 Iterations and execution times for ILU(0) preconditioned accelerators on four Markov chain matrices

4.3 Experiments with preconditioners

We tested the following preconditioning techniques: SOR, SSOR, ILU(0), and ILUT(p, ϵ), with $p = 2, 5, 8$, on all four test matrices. In all cases we used a drop-tolerance of $\epsilon = 0.0001$ in ILUT(p,ϵ). The non-preconditioned restarted GMRES algorithm was also tested (**nopre** in the table) as well as FGMRES using FOM as a preconditioner. In the latter case, we used a maximum of two steps for the inner FOM iteration. We used GMRES (or FGMRES) acceleration with Krylov subspace of dimension $m = 10$ in all cases. The convergence criterion used was to reduce the residual norm by a factor of 10^{-6} and as before, a maximum of 500 steps is allowed. Thus 501 steps indicates convergence has not taken place. The SSOR and SOR preconditioners are always used with the relaxation parameter $\omega = 1$.

	nopre	**ssor**	**ilu(0)**	**ilut**	**ilut**	**ilut**	**fom**	**sor**
		$k = 1$		$p = 2$	$p = 5$	$p = 8$		$k = 1$
marc1	501	391	111	54	5	4	1451	501
	49.7	56.3	15.9	7.3	1.4	1.4	109.8	75.8
marc2	501	501	256	501	22	5	1451	501
	18.9	27.0	13.8	26.0	1.7	0.8	40.9	28.7
marc3	176	31	20	23	8	6	283	80
	9.4	2.5	1.6	1.8	1.8	3.1	11.5	6.7
marc4	294	39	23	31	8	7	610	99
	38.1	7.3	4.4	5.5	4.2	7.5	60.2	19.5

Table 3 Iterations and execution times for preconditioned GMRES(10) for four Markov chain matrices

The results are shown in Table 3. All the times include the preprocessing to obtain the ILUT factorization. As can be observed, the best performance was obtained with ILUT(p,ϵ) with moderate values of the fill-in. As p increases, the processing time to obtain the L and U factors starts to become high and dominate the total time.

We also tested a k-step SOR and SSOR preconditioning techniques which we denote by SOR(k) and SSOR(k) respectively. As can be expected SSOR(k) is likely to be more reliable than SSOR(1). Indeed, using SSOR(k) is similar to using more fill-in in an ILUT preconditioning. The advantage of SSOR(k) over ILUT(p, ϵ), is that there is no additional storage required. On the other hand, as can be observed, SSOR(k) does not perform as well as ILUT on the harder examples. In general, however, we note that SSOR(1) rarely yields the best performance. It is very common that the best performance in terms of exectution time is obtained for SSOR(k) with a small k, in general less than 6 but larger than 1 or 2, for the examples we have tried. In one exception, namely marc2, SSOR(k) only converged for the larger values of k, and the overall time kept improving even all the way to ythe largest value of k used, namely $k = 16$. We should mension that the results with SOR(k), which are not reported here, are very similar. In fact each step of SSOR(k) behaves much like two steps of SOR(k).

	SSOR(k) Preconditioner							
	$k=2$	$k=4$	$k=6$	$k=8$	$k=10$	$k=12$	$k=14$	$k=16$
marc1	296	241	191	193	182	153	179	164
	85.2	113.0	123.9	160.1	183.9	183.5	245.8	254.6
marc2	501	501	501	501	223	229	141	116
	53.2	85.8	118.1	150.7	81.5	98.3	69.6	64.5
marc3	18	9	7	6	5	5	5	4
	2.8	2.2	2.3	2.4	2.4	2.8	3.2	2.7
marc4	19	11	8	7	6	6	5	5
	7.1	6.4	6.2	6.8	6.9	8.1	7.5	8.5

Table 4 Iterations and execution times for k-step SOR preconditioning applied to GMRES(10) for four Markov chain matrices

As can be observed, for the Marc1 and Marc2 matrices, the GMRES iteration preconditioned with SOR(1) and SOR(2) did not converge in the maximum of 500 steps allowed. However, SOR(k) converges for $k \geq 4$ and the convergence keeps improving. As k increases the cost of each preconditioning step increases and in the end the overall cost increases again. The optimum number of steps seems to be about $k = 10$ for the first three matrices.

5 CONCLUSION

The techniques described in this paper, as well as other ones developed in the recent literature, suggest that in many of the problems in Markov chain models, one can work with a Krylov subspace of small dimension and get good approximations to the original problem. This principle can be used to solve matrix eigenvalue problems, linear systems, and systems of Ordinary Differential Equations. Preconditioning linear systems or the eigenvalue problems arising from Markov chain problems is not too difficult because of the nature of the matrices. Some of the best, and most robust, preconditioning techniques are based on a form of Gaussian elimination with numerical dropping, in which the small values generated during Gaussian elimination are neglected to maintain sparsity. Other preconditioners such as those based on multiple-step SOR or SSOR can also be effective, and require less memory.

REFERENCES

[1] W. E. Arnoldi. The principle of minimized iteration in the solution of the matrix eigenvalue problem. *Quart. Appl. Math.*, 9:17–29, 1951.

[2] O. Axelsson. Conjugate gradient type-methods for unsymmetric and inconsistent systems of linear equations. *Linear Algebra Appl.*, 29:1–16, 1980.

[3] O. Axelsson. A generalized conjugate gradient, least squares method. *Num. Math.*, 51:209–227, 1987.

[4] R. Chandra. *Conjugate Gradient Methods for Partial Differential Equations.* PhD thesis, Yale University, Computer Science Dept., New Haven, CT. 06520, 1978.

[5] I. S. Duff and G. A. Meurant. The effect of reordering on preconditioned conjugate gradients. *BIT*, 29:635–657, 1989.

[6] L. C. Dutto. The effect of reordering on the preconditioned GMRES algorithm for solving the compressible Navier-Stokes equations. *International Journal for Numerical Methods in Engineering*, 36:457–497, 1993.

[7] S. C. Eisenstat, H. C. Elman, and M. H. Schultz. Variational iterative methods for nonsymmetric systems of linear equations. *SIAM J Num. Anal.*, 20:345–357, 1983.

[8] R. Fletcher. Conjugate gradient methods for indefinite systems. In G. A. Watson, editor, *Proceedings of the Dundee Biennal Conference on Numerical Analysis 1974*, pages 73–89, New York, 1975. University of Dundee,Scotland, Springer Verlag.

[9] Roland W. Freund. A Transpose-Free Quasi-Minimal Residual algorithm for non-Hermitian linear systems. *SIAM J. Sci. Comp.*, 14(2):470–482, 1993.

[10] E. Gallopoulos and Y. Saad. Efficient solution of parabolic equations by polynomial approximation methods. *SIAM J. Sci. Stat. Comput.*, 13:1236–1264, 1992.

[11] G. H. Golub and C. Van Loan. *Matrix Computations.* Academic Press, New York, 1981.

[12] K. C. Jea and D. M. Young. Generalized conjugate gradient acceleration of nonsymmetrizable iterative methods. *Linear Algebra Appl.*, 34:159–194, 1980.

[13] C. Lanczos. Solution of systems of linear equations by minimized iterations. *J. Res. Nat. Bur. Standards*, 49:33–53, 1952.

[14] C. C. Paige and M. A. Saunders. An algorithm for sparse linear equations and sparse least squares. *ACM Trans. Math. Software*, 8:43–71, 1982.

[15] B. Philippe, Y. Saad, and W. J. Stewart. Numerical methods in Markov chain modeling. *Journal of Operations Research*, 40(6):1156–1179, 1992.

[16] B. Philippe and R. B. Sidje. Transient solutions of Markov processes by Krylov subspaces. Technical Report 736, IRISA, University of Rennes, Campus de Beaulieu, Rennes, France, 1993.

[17] S. J. Polak, C. Den Heijer, W. H. A. Schilders, and P. Markowich. Semiconductor device modelling from the numerical point of view. *Internat. J. Numer. Meth. Eng.*, 24:763–838, 1987.

[18] Y. Saad. Krylov subspace methods for solving large unsymmetric linear systems. *Mathematics of Computation*, 37:105–126, 1981.

[19] Y. Saad. The Lanczos biorthogonalization algorithm and other oblique projection methods for solving large unsymmetric systems. *SIAM J. Numer. Anal.*, 19:470–484, 1982.

[20] Y. Saad. Practical use of some Krylov subspace methods for solving indefinite and unsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 5:203–228, 1984.

[21] Y. Saad. Analysis of some Krylov subspace approximations to the matrix exponential operator. *SIAM J. Numer. Anal.*, 29:209–228, 1992.

[22] Y. Saad. ILUT: a dual threshold incomplete ILU factorization. Technical Report 92-38, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, 1992. to appear.

[23] Y. Saad. *Numerical Methods for Large Eigenvalue Problems.* Halstead Press, New York, 1992.

[24] Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Sci. Stat. Comput.*, 14:461–469, 1993.

[25] Y. Saad. *Iterative methods for sparse linear systems.* PWS publishing, New York, 1995. to appear.

[26] Y. Saad and M. H. Schultz. Conjugate gradient-like algorithms for solving nonsymmetric linear systems. *Mathematics of Computation*, 44(170):417–424, 1985.

[27] Y. Saad and M. H. Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 7:856–869, 1986.

[28] Y. Saad and K. Wu. DQGMRES: a quasi-minimal residual algorithm based on incomplete orthogonalization. Technical Report UMSI-93/131, Minnesota Supercomputing Institute, Minneapolis,MN, 1993. submitted.

[29] P. Sonneveld. CGS, a fast Lanczos-type solver for nonsymmetric linear systems. *SIAM J. Scient. Statist. Comput.*, 10(1):36–52, 1989.

[30] W. J. Stewart. MARCA: markov chain analyzer, a software package for markov modeling. In W. J. Stewart, editor, *Numerical solution of Markov chains*, pages 37–62, New-York, 1991. Marcel Dekker Inc.

[31] W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains.* Princeton University Press, New York, 1994.

[32] H. A. van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of non-symmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 12:631–644, 1992.

[33] J. W. Watts-III. A conjugate gradient truncated direct method for the iterative solution of the reservoir simulation pressure equation. *Society of Petroleum Engineer Journal*, 21:345–353, 1981.