Numerical methods in Markov chain modeling

Bernard Philippe, Yousef Saad, William J. Stewart

INRIA

UNITE DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P 105
78153 Le Chesnay Cedex
France
Tél. (1) 39 63 55 11

Rapports de Recherche

N° 1115

Programme 1
Programmation, Calcul Symbolique
et Intelligence Artificielle

NUMERICAL METHODS IN MARKOV CHAIN MODELING

Bernard PHILIPPE
Youcef SAAD
William J. STEWART

Novembre 1989

Numerical Methods

in Markov Chain Modeling

Bernard PHILIPPE, Youcef SAAD

William J. STEWART

Publication Interne n° 495

Septembre 1989

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTÈMES ALÉATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CÉDEX
FRANCE
Téléphone : 99 36 20 00
Télex : UNIRISA 950 473 F
Télécopie : 99 38 38 32

Numerical Methods in Markov Chain Modeling

Résolution numérique de la modélisation par chaînes de Markov

Bernard Philippe† Youcef Saad* William J. Stewart+

September 29, 1989

Publication Interne n° 495 - Septembre 1989 - 46 Pages

† IRISA, Rennes, France. Research supported by CNRS (87:N 920070).
* Research Institute for Advanced Computer Science, NASA Ames Research Center. Moffett Field CA 94035. Research supported by Cooperative Agreement NCC 2-387 between the National Aeronautics and Space Administration (NASA) and the Universities Space Research Association (USRA).
+ North Carolina State University, Computer Science. Research supported in part by NSF (INT-8613332), by IRISA, France and by by Cooperative Agreement NCC 2-387 between the National Aeronautics and Space Administration (NASA) and the Universities Space Research Association (USRA).

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE
(L. A. 227)
UNIVERSITÉ DE RENNES 1 I. N. S. A. DE RENNES

INSTITUT NATIONAL DE RECHERCHE
EN INFORMATIQUE ET EN AUTOMATIQUE
(LABORATOIRE DE RENNES)

Abstract

This paper describes and compares several methods for computing stationary probability distributions of Markov chains. The main linear algebra problem consists of computing an eigenvector of a sparse, usually non-symmetric, matrix associated with a known eigenvalue. It can be also be cast as a problem of solving a homogeneous. singular linear system. We present several methods based on combinations of Krylov subspace techniques, single vector power iteration and relaxation procedures, and acceleration techniques. We compare the performance of these methods on some realistic problems.

Key words: Markov chain models; Homogeneous linear systems; Direct methods; Successive Overrelaxation; Preconditioned power iterations; Arnoldi's method; GMRES.

Résumé

Diverses méthodes, calculant la distribution des probabilités de l'etat stationnaire d'une chaîne de Markov, sont décrites et comparées dans cet article. Le problème consiste à calculer le vecteur propre associé à une valeur propre connue d'une matrice creuse et non symétrique. Il peut aussi être vu comme la résolution d'un système homogène singulier. Nous présentons plusieurs méthodes fondées sur des combinaisons de techniques d'espaces de Krylov, d'itérations sur un seul vecteur, de relaxations et de préconditionnements divers. Nous comparons les performances de ces méthodes sur quelques problèmes réels.

Mots clés : Modélisation par chaînes de Markov; systèmes linéaires homogènes; méthodes directes; méthode de sur-relaxation; méthode de la puissance avec preconditionnement; méthode d'Arnoldi; GMRES.

Contents

1 Introduction **5**

2 Iterative and Direct Solution Methods **7**

3 Direct Methods and Markov Chains **8**
3.1 Inverse Iteration 10
3.2 Compact Storage Schemes for Direct Methods 11

4 Single vector iterations **13**
4.1 The power method 13
4.2 Gauss-Seidel iteration and Successive Overrelaxation 13
4.3 SSOR Iteration 14
4.4 Preconditioned power iterations 15
4.5 Gauss-Seidel, SOR and SSOR preconditionings 15
4.6 ILU preconditioning 16

5 Projection Techniques **17**
5.1 General projection processes 17
5.2 Subspace Iteration 18
5.3 Arnoldi's method 19
5.4 Preconditioned GMRES for singular systems 21
5.5 Preconditioned Arnoldi and GMRES Algorithms 23

6 Numerical tests **23**
6.1 Example 1: An Interactive Computer System. 24
6.1.1 Results for Example 1 25
6.2 Example 2: A Telecommunications Model. 26
6.2.1 Results for Example 2 27
6.3 Example 3: A Multi-Class, Finite Buffer, Priority System. 28
6.3.1 Results for Example 3 29

7 Conclusion **30**

List of Tables

1 Performance Results for Example 1. N=1,771; NZ=11,011. NCD Case. . . 35
2 Performance Results for Example 1. N=23,426; NZ=156,026. NCD Case. . 36
3 Performance Results for Example 1. N=1,771; NZ=11,011 Non-NCD Case 37
4 Performance Results for Example 1. N=23,426; NZ=156,026. Non-NCD Case 38
5 Performance Results for Example 2. N=2,431; NZ=11,681 39
6 Performance Results for Example 2. N=17,081; NZ = 84,211 40
7 Performance Results for Example 3. N=1,940; NZ=12,824. NCD Case. . . 41
8 Performance Results for Example 3. N=19,620; NZ=131,620. NCD Case. . 42
9 Performance Results for Example 3. N=1,940; NZ=12,824. Non-NCD Case 43
10 Performance Results for Example 3. N=19,620; NZ=131,620. Non-NCD Case. 44

List of Figures

1 Illustration for example 1 24
2 Illustration for example 2 27
3 Illustration for example 3 29

1 Introduction

It is often possible to represent the behavior of a physical system by describing all the different states which the system can occupy and by indicating how the system moves from one state to another in time. If the time spent in any state is exponentially distributed, or, equivalently, if the probabilities of transition depend only on the state currently occupied by the system and not on the length of time that the state is occupied nor on any previously occupied state, then the system may be represented by a Markov process. Even when the system does not possess this exponential property explicitly, it is usually possible to construct a corresponding implicit representation. Examples of the use of Markov processes may be found extensively throughout the biological, physical and social sciences as well as in business and engineering.

A Markov process is characterized by a (possibly infinite) set of states. The system being modelled by this process is assumed to occupy one and only one of these states at any given time. The evolution of the system is represented by transitions of the Markov process from one state to another. These transitions are assumed to occur instantaneously; in other words, the action of moving from one state to another consumes zero time. The fundamental property of a Markovian system, referred to as the Markov property, is that the future evolution of the system depends only on the current state of the system and not on the past history.

The information we would like to obtain from the system is a knowledge of the probabilities of being in a given state or set of states at a certain time after the system becomes operational. Often this time is taken to be sufficiently long that all influence of the initial starting state has been erased. The probabilities thus obtained are referred to as the *long-run* or *stationary* probabilities. We shall denote by $\pi_i(n)$ the probability that the Markov chain is in state i at step n, i.e. $\pi_i(n) = Prob\{X_n = i\}$. In vector notation we let $\pi(n) = (\pi_1(n),\ \pi_2(n),\ ...,\ \pi_i(n),\ ...,\ )$. Note that the vector π is a row vector. We shall adopt the convention that all probability vectors are row vectors. All other vectors will be considered to be column vectors unless specifically stated otherwise.

Consider a system that is modelled by a continuous time Markov chain (CTMC). Let $\pi_i(t)$ be the probability that the system is in state i at time t. Furthermore, let $q_{ij}, i = 1 \cdots n; j = 1, \cdots n$ denote the rate of transition from state i to state j. Then

$$\pi_i(t + \Delta t) = \pi_i(t) \left(1 - \sum_{j \neq i} q_{ij}(t)\Delta t\right) + \left(\sum_{k \neq i} q_{ki}(t)\pi_k(t)\right) \Delta t. \tag{1}$$

We may set $q_{ii}(t) = -\sum_{j \neq i} q_{ij}(t)$, which yields

$$\pi_i(t + \Delta t) = \pi_i(t) + \left(\sum_k q_{ki}(t)\pi_k(t)\right) \Delta t + o(\Delta t), \tag{2}$$

and

$$\lim_{\Delta t \to 0} \frac{\pi_i(t + \Delta t) - \pi_i(t)}{\Delta t} = \lim_{\Delta t \to 0} \left(\sum_k q_{ki}(t)\pi_k(t) + \frac{o(\Delta t)}{\Delta t}\right), \tag{3}$$

i.e.

$$\frac{d\pi_i(t)}{dt} = \sum_k q_{ki}(t)\pi_k(t). \tag{4}$$

In matrix notation, this gives

$$\frac{d\pi(t)}{dt} = \pi(t)Q(t). \tag{5}$$

When the Markov chain is homogeneous, we may drop the dependence on time and simply write,

$$\frac{d\pi(t)}{dt} = \pi(t)Q. \tag{6}$$

At steady state, the rate of change of $\pi(t)$ is zero and therefore, for a homogeneous CTMC,

$$\pi Q = 0. \tag{7}$$

The vector π , (now written independent of t) is the stationary or long-run probability vector (π_i is the probability of being in state i at statistical equilibrium) and may be obtained by applying equation solving techniques to the homogeneous system of equations (7). Also, this equation may be written as

$$\pi P = \pi, \tag{8}$$

where $P = Q\Delta t + I$ and the desired probability vector appears as the eigenvector corresponding to a unit eigenvalue of P. If Δt is chosen sufficiently small then P is a stochastic matrix. For irreducible Markov chains, the unit eigenvalue of P is the one with largest real part. Matrix techniques may be applied to either (7) or (8) to determine the stationary probability vector. The matrix Q is called the transition rate matrix, or the infinitesimal generator of the Markov chain, while the matrix P is called the transition probability matrix.

One of the practical difficulties is the large size of the matrices involved. For small dense matrices the method of choice for solving linear systems of equations is the LU decomposition and for computing eigenvalues/eigenvectors, it is the well-known QR algorithm. However, as the problem sizes increase these standard methods, or any method intended for dense problems, becomes uneconomical. Moreover, the standard methods do not take advantage of the sparsity of P.

Before discussing the algorithms themselves, we would like to say a few words on the potential difficulty inherent in the problems to be solved. Nonsymmetric eigenproblems can be so sensitive to variations in the original data, namely the matrix P, that any procedure to approximate eigenvalues or eigenvectors of P may encounter serious difficulties. We examine here the particular case of the unit eigenvalue of P. The sensitivity of a given eigenvalue λ_i of P to perturbations is usually measured by its condition number which is defined as the inverse of the cosine of the angle between its corresponding right and left eigenvectors y_i and z_i, i.e.,

$$c(\lambda_i) = \frac{\|z_i\|\|y_i\|}{|(z_i, y_i)|}. \tag{9}$$

In practice this means that a small perturbation to P, of norm ϵ, may disturb the eigenvalue λ_i by as much as $c(\lambda_i)\epsilon$. In the situation of interest to us, the right eigenvector is known to be $(1, 1, ..., 1)^T$ and the left eigenvector consists of non-negative components. As a result we can show that $c(\lambda_i) \leq \sqrt{n}$. The eigenvalue is therefore well-conditioned in general.

The condition number for the eigenvector y_i involves the reduced resolvent $S(\lambda_i)$, defined as the inverse of the restriction of $P - \lambda_i I$ to $\{z_i\}^\perp$, the subspace orthogonal to the left eigenspace associated with λ_i, see [7], p. 17:

$$c(y_i) = \|S(\lambda_i)\|.$$

Though not apparent from the definition, the condition number for the eigenvector is implicitly related to that for the eigenvalues of P, see Wilkinson [28]. Moreover, it is easy to show from the definition that [7]

$$c(y_i) \geq \max_{j \neq i} \frac{1}{|\lambda_j - \lambda_i|}.$$

A consequence of the above inequality is that a poor separation of the unit eigenvalue from the other eigenvalues will cause poor conditioning for the associated eigenvectors.

Another difficulty caused by the poor separation of the unit eigenvalue is a slow rate of convergence. In Markov chain models, there is often a cluster of eigenvalues very close to the unit eigenvalue, a result of the near decomposability of the system. This may render the eigenvalue methods untolerably slow. It is important to detect such cases and use appropriate alternative decomposition methods when they arise.

2 Iterative and Direct Solution Methods

Iterative methods of one type or another are by far the most commonly used methods for obtaining the stationary probability vector from either the stochastic transition probability matrix or from the infinitesimal generator. There are several important reasons for this choice. First, an examination of the iterative methods usually employed shows that the only operation in which the matrices are involved, are multiplications with one or more vectors, or with preconditioners. These operations do not alter the form of the matrix and thus compact storage schemes, which minimize the amount of memory required to store the matrix and which in addition are well suited to matrix multiplication, may be conveniently implemented. Since the matrices involved are usually large and very sparse, the savings made by such schemes can be considerable. One such sparse storage scheme, and the one used in implementing the iterative procedures in this study, is the compressed sparse row format format [9, 20]. In this scheme only the non-zero elements, their column indices and an index to the beginning of each row is kept. With direct equation solving methods, the elimination of one non-zero element of the matrix during the reduction phase often results in the creation of several non-zero elements in positions which previously contained zero. This is called fill-in and not only does it make the organization of a compact storage scheme more difficult, since provision must be made for the deletion and the insertion of elements, but in addition, the amount of fill-in can often be so extensive that available memory is quickly exhausted. A successful direct method must incorporate a means of overcoming these difficulties.

Iterative methods have other advantages. Use may be made of good initial approximations to the solution vector and this is especially beneficial when a series of related experiments is being conducted. In such circumstances the parameters of one experiment

often differ only slightly from those of the previous; many will remain unchanged. Consequently, it is to be expected that the solution to the new experiment will be close to that of the previous and it is advantageous to use the previous result as the new initial approximation. If indeed there is little change, we should expect to compute the new result in relatively few iterations.

An iterative process may be halted once a prespecified tolerance criterion has been satisfied, and this may be relatively lax. For example, it may be wasteful to compute the solution of a mathematical model correct to full machine precision when the model itself contains errors of the order of 5-10%. A direct method is obligated to continue until the final specified operation has been carried out.

And lastly, with iterative methods, the matrix is never altered and hence the build-up of rounding error is, to all intents and purposes, non-existent.

For these reasons, iterative methods have traditionally been preferred to direct methods. However, iterative methods have a major disadvantage in that often they require a very long time to converge to the desired solution. More advanced iterative techniques such as the method of Arnoldi, have helped to alleviate this problem but much research still remains to be done, particularly in estimating a priori, the number of iterations, and hence the time, required for convergence. Direct methods have the advantage that an upper bound on the time required to obtain the solution may be determined before the calculation is initiated. More important, for certain classes of problem, direct methods often result in a much more accurate answer being obtained in *less time*. Since iterative method will in general require less memory than direct methods, these latter can only be recommended if they obtain the solution in less time. Unfortunately, it is often difficult to predict when a direct solver will be more efficient than an iterative solver.

3 Direct Methods and Markov Chains

We are concerned with obtaining the stationary probability vector π from the equations:

$$\pi Q = 0, \pi \geq 0, \pi e = 1. \tag{10}$$

Note that if we try to apply direct methods to the alternate formulation

$$\pi P = \pi, \tag{11}$$

we need to first rewrite this as

$$\pi(I - P) = 0, \tag{12}$$

and in both cases we need to solve a homogeneous system of n linear equations in n unknowns. A homogeneous system of n linear equations in n unknowns has a solution other than the trivial solution ($\pi_i = 0$, for all i) if and only if the determinant of the coefficient matrix is zero, i.e. if and only if the coefficient matrix is singular.

Since the determinant of a matrix is equal to the product of its eigenvalues and since Q and $(I-P)$ both possess a zero eigenvalue, the singularity of Q (and $I-P$) and hence the existence of a non-trivial solution, follows. It is known that if the matrix Q is irreducible, there exists lower and upper triangular matrices L and U such that

$$Q^T = LU. \tag{13}$$

Once an LU decomposition has been determined, a forward substitution step followed by a backward substitution is usually sufficient to determine the solution of the system of equations. For example, suppose we are required to solve $Ax = b$ with $det(A) \neq 0$ and $b \neq 0$ and suppose further that the decomposition $A = LU$ is available so that $LUx = b$. By setting $Ux = z$, the vector z may be obtained by forward substitution on $Lz = b$, since both L and b are known quantities. The solution x may subsequently be obtained from $Ux = z$ by backward substitution since by this time both U and z are known quantities.

However, in the case of the numerical solution of Markov chains, the system of equations, $\pi Q = 0$, is homogeneous, i.e. $b = 0$, and the coefficient matrix is singular. In this case, the final row of U (if the Doolittle decomposition has been performed) is equal to zero. Proceeding as indicated above for the non-homogeneous case, we have

$$Q^T x = (LU)x = 0. \tag{14}$$

If we now set $Ux = z$ and attempt to solve $Lz = 0$ we find that, since L is non-singular, we must have $z = 0$. Let us now proceed to the back substitution on $Ux = z = 0$ when $u_n n = 0$. It is evident that we may assign any non-zero value to x_n, say $x_n = \eta$, and then determine, by simple back-substitution, the remaining elements of the vector x in terms of η. We have $x_i = c_i \eta$ for some constants $c_i, i = 1, 2, ..., n$, and $c_n = 1$. Thus the solution obtained depends on the value of η. There still remains one equation that the elements of a probability vector must satisfy, namely that the sum of the probabilities must be one. Consequently, normalizing the solution obtained from solving $Ux = 0$ so that the conservation of probability condition holds, yields the desired unique stationary probability vector π corresponding to the infinitesimal generator Q.

An alternative approach to this use of the normalization equation is to replace the last equation of the original system with $\pi e = 1$. If the Markov chain is irreducible, this will ensure that the coefficient matrix is non-singular. Furthermore, the system of equations will no longer be homogeneous (since the right hand side is now e_n), and so the solution may be computed without problem.

Of course, it is not necessary to replace the last equation of the system by the normalization equation. Indeed, any equation could be replaced. However, this is generally undesirable, for it will entail more numerical computation. For example, if the first equation is replaced, the first row of the coefficient matrix will contain all ones and the right hand side will be e_1. The first consequence of this is that during the forward substitution stage, the entire sequence of operations must be performed to obtain the vector z; whereas if the last equation is replaced, it is simply possible to read off the solution immediately, i.e. $z_1 = z_2 = ...z_{n-1} = 0$ and $z_n = 1$. The second and more damaging consequence is that substantial fill-in will occur since a multiple of the first row which contains all ones must be added to *all* remaining rows and a cascading effect will undoubtedly occur in all subsequent reduction steps. The problem of fill-in, which plagues direct methods is considered later. However, one should note that available packages such as MA28, see e.g. [9], will prevent this disastrous situation by reordering the equations dynamically. The strategy used in a package such as MA28 attempts to reach a compromise between numerical stability and minimization of fill-in.

3.1 Inverse Iteration

Inverse iteration is the method of choice for the direct solution of $\pi Q = 0$. Although this may sound rather like a contradiction in terms, we shall see that inverse iteration, when applied to an infinitesimal generator matrix Q to obtain the stationary probability vector π requires only a single iteration to determine π at least as accurately as any of the aforementioned direct methods. In fact, this method simply reduces to the standard LU decomposition method with special treatment of the zero pivot and the right-hand side vector.

Consider an iterative scheme based on the relation

$$x^{(k)} = (Q^T - \mu I)^{-1} x^{(k-1)}. \tag{15}$$

Let $x^{(0)}$ be an arbitrary column vector that can be written as a linear combination of the right-hand eigenvectors of Q^T; i.e.

$$x^{(0)} = \sum_{i=1}^{n} \alpha_i v_i, \tag{16}$$

where the vectors v_i are the right eigenvectors of the matrix Q^T corresponding to the eigenvalues λ_i; i.e

$$Q^T v_i = \lambda_i v_i; \quad i = 1, 2, ..., n. \tag{17}$$

Then

$$x^{(k)} = (Q^T - \mu I)^{-k} x^{(0)} = \sum_{i=1}^{n} \alpha_i (\lambda_i - \mu)^{-k} v_i \tag{18}$$

$$= (\lambda_r - \mu)^{-k} [\alpha_r v_r + \sum_{i \neq r}^{n} \alpha_i (\lambda_r - \mu)^k (\lambda_i - \mu)^{-k} v_i]. \tag{19}$$

Consequently, if for all $i \neq r$, $|\lambda_r - \mu| << |\lambda_i - \mu|$ convergence to the eigenvector v_r is rapid since $[(\lambda_r - \mu)/(\lambda_i - \mu)]^k$ will rapidly tend to zero. If $\mu = \lambda_r$, then the summation in equation (19) is zero and the eigenvector v_r will be obtained to full machine precision in a single iteration.

Note that when implementing inverse iteration, there is no need to explicitly form the inverse of the shifted matrix $(Q^T - \mu I)$. Instead, the approach to be adopted is to solve the set of linear equations

$$(Q^T - \mu I) x^{(k)} = x^{(k-1)}.$$

This is obviously identical to the original formulation in equation (15). If μ is not an eigenvalue of Q, then $(Q^T - \mu I)$ is non-singular and for $x^{(k-1)} \neq 0$, an LU decomposition approach can be implemented without further ado.

If μ is an eigenvalue of Q (i.e. $\mu = \lambda_r$), then $(Q^T - \mu I)$ is singular. In this case the zero pivot which arises during the LU decomposition should be replaced by a small value ϵ. This should be chosen as the smallest representable number such that $1 + \epsilon > 1$ on the particular computer being used. After one iteration, this approach results in a very inaccurate solution to the set of equations, but a rigorous error analysis [28] will show that since the elements in the solution vector possess errors in the same ratio, normalizing this vector will yield a very accurate eigenvector.

In our particular case we are looking for the right eigenvector corresponding to the zero eigenvalue of Q^T. Therefore, letting $\mu = 0$ in the iteration formula, we get

$$(Q^T - 0I)x^{(k)} = Q^T x^{(k)} = x^{(k-1)} \tag{20}$$

and thus we are simply required to solve

$$Q^T x^{(1)} = x^{(0)} \tag{21}$$

Note that choosing $x^{(0)} = e_n$ reduces the amount of computation involved. The iteration simply reduces to the back substitution step

$$Ux^{(1)} = \frac{1}{\epsilon e_n}, \tag{22}$$

An appropriate normalization of $x^{(1)}$ will yield the stationary probability vector, i.e.

$$\pi^T = \frac{1}{e^T x^{(1)}} x^{(1)}. \tag{23}$$

3.2 Compact Storage Schemes for Direct Methods

Frequently the matrices generated from Markov models are too large to permit regular two-dimensional arrays to be used to store them in computer memory. Since these matrices are usually very sparse, it is economical, and indeed necessary, to use some sort of packing scheme whereby only the non-zero elements and their positions in the matrix are stored. One of the most commonly used storage schemes for sparse matrices is the row sparse compact storage, sometimes referred to as the a, ja, ia scheme [9]. This involves the use of a real array $A(1 : NZ)$ containing the non-zero elements of the matrix, stored row-wise, an integer array $JA(1 : NZ)$ containing the column positions of the corresponding elements in the real array A, and finally a pointer integer array $IA(1 : N + 1)$ the i-th element of which points to the beginning in the arrays A and JA of the consecutive rows.

When a direct equation solving method is to be applied, provision usually must be made to include elements which become non-zero during the reduction and somewhat less important to remove elements which have been eliminated. If memory locations are not urgently required, the easiest way of removing an element is to set it to zero without trying to recuperate the words which were used to store the element and its location pointers. To include an element into the storage scheme, either some means of appending this element to the end of the storage arrays must be provided, or else sufficient space must be left throughout the arrays so that fill-in can be accommodated as and when it occurs. The first usually requires the use of link pointers and is most useful if the non-zero elements are randomly dispersed throughout the matrix, while the second is more useful if the pattern of non-zero elements is rather regular.

When applying direct equation solving methods such as Gaussian elimination, it is usually assumed that the complete set of linear equations has already been derived and that the entire coefficient matrix is stored somewhere in the computer memory, albeit in a compact form. The reduction phase begins by using the first equation to eliminate all non-zero elements in the first column of the coefficient matrix from column position 2 through

n. More generally, during the i^{th} reduction step, the i^{th} equation is used to eliminate all non-zero elements in the i^{th} column from positions $(i+1)$ through n. (Naturally, it is assumed that the pivot elements are always non-zero, otherwise the reduction breaks down).

However, since we are responsible for both the initial generation of the system of equations and for its solution, it is possible to envisage an alternative approach, and one which has several advantages over the traditional method outlined above. Assume, as is usually the case, that the coefficient matrix can be derived row by row. Then, immediately after the second row has been obtained, it is possible to eliminate the element in position (2,1) by adding a multiple of the first row to it. This process may be continued so that when the i-th row of the coefficient matrix is generated, rows 1 through $(i-1)$ have been derived and are already reduced to upper triangular form. The first $(i-1)$ rows may therefore be used to eliminate all non-zero elements in row i from column positions $(i,1)$ through $(i,i-1)$, thus putting it into the desired triangular form. Note that since this reduction is performed on Q^T, it is the columns of the infinitesimal generator that are required to be generated one at a time and not its rows.

This method has a distinct advantage in that once a row has been generated in this fashion, no more fill-in will occur into this row. It is suggested that a separate storage area be reserved to hold temporarily a single unreduced row. The reduction is performed in this storage area. Once completed, the reduced row may be compacted into any convenient form and appended to the rows which have already been reduced. In this way no storage space is wasted holding subdiagonal elements which, due to elimination, have become zero, nor in reserving space for the inclusion of additional elements. The storage scheme should be chosen bearing in mind the fact that these rows will be used in the reduction of further rows and also later in the algorithm during the back-substitution phase.

Since the form of the matrix will no longer be altered, the efficient storage schemes which are used with many iterative methods can be adopted. Note that this approach can not be used for solving general systems of linear equations because it inhibits a pivoting strategy from being implemented. It is valid when solving irreducible Markov chains since pivoting is not required in order that the LU decomposition of an infinitesimal generator matrix Q^T be performed in a stable manner.

Later in this paper, we report on our computational experience with a direct method programmed according to the guidelines given above. Specifically, we implemented a sparse inverse iteration algorithm called GE (for Gaussian Elimination). This program accepts the transpose of a transition rate matrix which is stored in the usual row compact form [9]. It extracts each row of Q^T one at a time, expands this row into a vector of length n and performs reductions on it by adding multiples of previously reduced rows. When the reduction is completed, the reduced row is compacted once again and appended to previously reduced rows. The multipliers are not kept. As is shown later, this method works extremely well for small and medium sized problems (less than 2,500 states), but it requires too much memory when large problems are involved. We also experimented with the software package MA28, but its performance was always inferior to that of GE. The reason is that GE was designed uniquely for Markov chain problems, while MA28 was designed as a general purpose sparse linear equation solver. We stress that GE should not be used to solve systems of equations which require pivoting. In these, and in a wide

variety of other problems, MA28 has been used very successfully [16].

4 Single vector iterations

4.1 The power method

The simplest iteration method for computing the dominant eigenvector of a matrix A is the single vector iteration

$$x^{(k+1)} = \frac{1}{\xi^{(k)}} A x^{(k)}$$

where $\xi^{(k)}$ is a normalizing factor, typically the component of the vector $Ax^{(k)}$ that has the largest modulus. One problem with this simple scheme is that its rate of convergence can be very slow. The convergence factor for the dominant eigenvalue λ_1 is given by λ_2/λ_1, where λ_2 is the subdominant eigenvalue. In situations where the eigenvalues cluster around λ_1, as is the case for nearly decomposable systems, the convergence can be unacceptably slow.

For our situation the matrix of interest A is P^T. Since we know that the matrix has row sums equal to 1 and has 1 as the dominant eigenvalue, we can safely skip the normalizing factor and the above iteration takes the form

$$\begin{aligned} x^{(k+1)} &= P^T x^{(k)} && (24) \\ &= x^{(k)} - Q^T x^{(k)} && (25) \end{aligned}$$

4.2 Gauss-Seidel iteration and Successive Overrelaxation

Relaxation schemes are based on the decomposition

$$Q^T = D - E - F$$

where D is the diagonal of Q^T, $-E$ is the strict lower part of Q^T and $-F$ its strict upper part. The Gauss-Seidel iteration then takes the form

$$(D - E)x^{(k+1)} = Fx^{(k)}. \tag{26}$$

This corresponds to correcting the j-th component of the current approximate solution, for $j = 1, 2, ..n$, i.e., from top to bottom, by making the j-th component of the residual vector equal to zero. To denote specifically the direction of solution this is sometimes referred to as *forward* Gauss-Seidel. A *backward* Gauss-Seidel iteration takes the form

$$(D - F)x^{(k+1)} = Ex^{(k)} \tag{27}$$

and corresponds to correcting the components from bottom to top.

Note that convergence of the above (forward) iteration is governed by the spectral radius of $(D - E)^{-1}F$. Convergence may sometimes be improved by using the alternative splitting

$$\omega Q^T = (D - \omega E) - (\omega F + (1 - \omega)D)$$

which leads to the iteration, called successive overrelaxation, (SOR)

$$(D - \omega E)x^{(k+1)} = (\omega F + (1 - \omega D))\, x^{(k)}. \tag{28}$$

A backward SOR relaxation may also be written.

For many problems there exist some value of ω which provides the best possible convergence rate. The resulting optimal convergence rate can be a considerably improvement over Gauss-Seidel. The choice of an optimal, or even a reasonable, value for ω has been the subject of much study, especially for problems arising in the numerical solution of partial differential equations [27]. Some results have been obtained for certain classes of matrices. Unfortunately, very little known at present for arbitrary non-symmetric linear systems,

As a general rule, it is best to use a forward iterative method when the preponderance of the elemental mass is to be found below the diagonal for in this case, the iterative method essentially works with the inverse of the lower triangular portion of the matrix and intuitively, the closer this is to the inverse of the entire matrix, the faster the convergence. On the other hand, the backward iterative schemes work with the inverse of the upper triangular portion and these methods work best when the non-zero mass lies above and on the diagonal. We point out that some specialized counter examples exist which makes the above recommendations only rules of thumb.

Little information is available on the effect of the ordering of the state space on the convergence of these iterative methods. Examples are available in which Gauss-Seidel works extremely well for one ordering but not at all for an opposing ordering, [15]. In these examples the magnitude of the non-zero elements appears to have little effect on the speed of convergence. It appears that an ordering that in some sense preserves the direction of probability flow works best.

4.3 SSOR Iteration

The Symmetric Successive Overrelaxation method (SSOR) consists of following a relaxation sweep from top down by a relaxation sweep from bottom up. Thus, the case $\omega = 1$ corresponding to a SGS (Symmetric Gauss Seidel) scheme would be as follows:

$$(D - E)x^{(k+1/2)} = Fx^{(k)} \tag{29}$$

$$(D - F)x^{(k+1)} = Ex^{(k+1/2)} \tag{30}$$

while for arbitrary ω, it is:

$$(D - \omega E)x^{(k+1/2)} = (\omega F + (1 - \omega D))\, x^{(k)} \tag{31}$$

$$(D - \omega F)x^{(k+1)} = (\omega E + (1 - \omega D))\, x^{(k+1/2)} \tag{32}$$

The main attraction of SSOR schemes is that the iteration matrix is similar to a symmetric matrix when the original matrix Q^T is symmetric. This situation rarely occurs in Markov chain models. SSOR does however, help to reduce poor convergence behavior that results from a badly ordered state space.

4.4 Preconditioned power iterations

As was already mentioned the power method can be extremely slow to converge when the subdominant eigenvalue is very close to one. The relaxation schemes described above typically have a better convergence rate. This means that the iteration matrices corresponding to these schemes have an eigenvalue λ_2 farther away from 1 than the original matrix.

Preconditioning is a technique whereby the original system of equations is modified in such a way that the solution is unchanged but the distribution of the eigenvalues is better suited for iterative methods. In a general context, a preconditioning technique consists of replacing a system $Ax = b$ by a modified system such as $M^{-1}Ax = M^{-1}b$. Here M is a preconditioning matrix for which the solution of $Mx = y$ is inexpensive. When the coefficient matrix is singular and the right hand side is zero, the method turns out to be equivalent to the power method applied to the matrix $(I - M^{-1}A)$.

We have seen that for the numerical solution of Markov chain problems, the power method may be written as

$$\begin{aligned} x^{(k+1)} &= x^{(k)} - Q^T x^{(k)} && (33) \\ &= (I - Q^T)x^{(k)} && (34) \end{aligned}$$

Here preconditioning involves premultiplying the matrix Q^T with a matrix M^{-1}, generally chosen so that M approximates Q^T but is such that its LU decomposition can be efficiently determined. In this case, the iteration matrix, $(I - M^{-1}Q^T)$ has one unit eigenvalue and the remaining eigenvalues are (hopefully) all close to zero, leading to a rapidly converging iterative procedure. In this paper we refer to such methods as preconditioned power iterations, or fixed point iterations.

4.5 Gauss-Seidel, SOR and SSOR preconditionings

A look at (26) reveals an interesting connection with the power method. We can rewrite (26) as

$$\begin{aligned} x^{(k+1)} &= (D - E)^{-1}Fx^{(k)} \\ &= (D - E)^{-1}\left((D - E) - Q^T\right)x^{(k)} \\ &= x^{(k)} - (D - E)^{-1}Q^T x^{(k)} \end{aligned}$$

Comparing this with equation (25), we observe that the above iteration is simply the power method applied to the matrix

$$I - (D - E)^{-1}Q^T. \qquad (35)$$

Thus $(D - E)$ performs the role of the preconditioning matrix M. As a result we may view the Gauss-Seidel method as a preconditioned power iteration. It is an attempt to reduce λ_2, without changing the eigenvector.

The solution to the above system is identical with that of the original one. Its rate of convergence, on the other hand, may be substantially faster than that of the original problem. For this reason we will refer to the system (35) as the Gauss-Seidel preconditioned version of $Q^T x = 0$. Similarly one can define an SOR preconditioning and an SSOR preconditioning.

4.6 ILU preconditioning

By far the most popular preconditioning techniques are the incomplete LU factorization techniques. These are sometimes also referred to as "combined direct-iterative" methods. Such methods are composed of two phases. First we start out by initiating an LU decomposition of Q^T. At various points in the computation, non-zero elements may be omitted according to various rules. Some possibilities are discussed in the following paragraphs. In all cases, instead of arriving at an exact LU decomposition, what we obtain is of the form

$$Q^T = LU - E$$

where E, called the remainder, is expected to be small in some sense. When this has been achieved, the "direct" phase of the computation is completed. In the second phase, this (incomplete) factorization is incorporated into an iterative procedure by writing

$$Q^T x = (LU - E)x = 0$$

and then using

$$LUx^{(k+1)} = Ex^{(k)}$$

or equivalently

$$x^{(k+1)} = x^{(k)} - (LU)^{-1}Q^T x^{(k)}$$

as the iteration scheme. Note that this is the same as solving the preconditioned (from the left) system of equations

$$U^{-1}L^{-1}Q^T x = 0$$

by the power method.

In this paper we report on the numerical results obtained with three different incomplete factorizations. The first has been widely discussed and found to be successful especially when applied to systems of equations that arise in the solution of elliptic partial differential equations. Given the matrix Q^T, this ILU factorization consists of performing the usual Gaussian Elimination factorization and dropping any fill-in during the process. In other words,

$$Q^T = LU + E \tag{36}$$

where L is unit lower triangular, U is upper triangular, and $L + U$ has the same zero structure as the matrix Q^T. This is referred to as ILU(0) or IC(0), for Incomplete Choleski, in the symmetric case.

If we denote by $NZ(Q)$ the set of all pairs (i,j) for which $q_{ij} \neq 0$, then a formal description of the ILU(0) algorithm applied to a matrix Q is as follows. Note that the diagonal elements of U are not stored since they are known implicitly to be unity.

Algorithm: ILU(0)

Do $i = 1, n$
  Do $j = 1, n$
    If $(i,j) \in NZ(Q)$ then
      * Compute $s = q_{ij} - \sum_{k=1}^{\min(i,j)-1} l_{ik}u_{kj}$
      * If $(i \geq j)$ then $l_{ij} = s$
      * If $(i<j)$ then $u_{ij} = s/l_{ii}$

ILU(0) is known to exist for non-singular M-matrices [14]. It may also be shown to exist for the matrix Q (and Q^T), by trivially extending the results in [2], page 42.

The second incomplete factorization that we studied is a threshold based scheme. Here the decomposition proceeds in a manner similar to that described for the GE method of section 3.2. However, after a row of the matrix has been reduced and before that row is recompacted and stored, each non-zero element is examined. If the absolute value of any element in the row is less than a prespecified threshold, then it is replaced by zero. Similarly, if any of the multipliers formed during the reduction are less than the threshold, they are dropped from further consideration. The only exception to this drop threshold are the diagonal elements which are kept no matter how small they become. We refer to this incomplete factorization technique as ILUTH.

The final type of incomplete factorization which we examined, is based on a realization that only a fixed amount of memory may be available to store the incomplete factors, L and U, so only a fixed number of non-zero elements are kept in each row. These are usually chosen to be the largest in magnitude. The algorithm proceeds in the same way as ILUTH. When a row has been reduced, a search is conducted to find the K largest elements in absolute value. This search is conducted over both the multipliers and the non-zero elements to the right of the diagonal element. As before, the diagonal elements are kept regardless of their magnitude. In our experiments, this incomplete factorization is referred to as ILUK.

Although the above three ILU factorizations are the only ones we considered, there are other possibilities. For example, some ILU-based methods make use of the symmetric zero structure of a matrix [11]. In other words, LU is the exact decomposition of the symmetric non-zero portion of the matrix. $W = LU$ is chosen as

$$\begin{aligned} w_{ij} &= q_{ji} \quad \text{if} \quad q_{ij}q_{ji} \neq 0, \\ w_{ij} &= 0 \text{ otherwise,} \end{aligned}$$

and now standard symmetric ordering schemes, such as those available in SPARSPAK, can be modified and used quite effectively. However, we believe that ILU0, ILUTH and ILUK will be the most effective for Markov chain problems.

5 Projection Techniques

5.1 General projection processes

An idea that is basic to sparse linear systems and eigenvalue problems is that of projection processes [23]. Given a subspace K spanned by a system of m vectors $V \equiv [v_1, \ldots, v_m]$ a projection process onto $K \equiv span\ \{V\}$ finds an approximation to the original problem from the subspace K. For a linear system $Ax = b$, this is done by writing $x = Vy$ and requiring that the residual vector $b - AVy$ be orthogonal to some subspace L , not necessarily equal to K. If a basis for L is $W = \text{span}\{w_1, w_2, ..., w_m\}$ then this yields the condition:

$$W^T(b - AVy) = 0$$

or

$$\tilde{x} = V[W^T AV]^{-1} W^T b \tag{37}$$

For an eigenvalue problem $Ax = \lambda x$, we seek an approximate eigenvalue $\tilde{\lambda} \in \mathbb{C}$ and an approximate eigenvector $\tilde{x} \in K$ such that the residual vector $Ax - \lambda x$ is orthogonal to the subspace L. Writing again $x = Vy$ and translating this Petrov-Galerkin condition yields,

$$W^T(AVy - \tilde{\lambda} Vy) = 0$$

or

$$W^T AVy = \tilde{\lambda} W^T Vy \tag{38}$$

which is a generalized eigenvalue problem of dimension m. The minimum assumptions that must be made in order for these projection processes to be feasible are that $W^T AV$ be non-singular for linear systems and that $W^T V$ be non-singular for eigenvalue problems. Clearly this will provide m approximate eigenpairs λ_i, x_i. In most algorithms, the matrix $W^T V$ is the identity matrix, in which case the approximate eigenvalues of A are the eigenvalues of the $m \times m$ matrix $C = W^T AV$. The corresponding approximate eigenvectors are the vectors Vy_i where y_i are the eigenvectors of C. Similarly the approximate Schur vectors are the vector columns of VU, where $U = [u_1, u_2, \ldots, u_m]$ are the Schur vectors of C, i.e., $U^H CU$ is quasi-upper triangular. A common particular case is when $K = L$ and $V = W$ is an orthogonal basis of K. This is then referred to as an orthogonal projection process.

Note that we can adopt either of the two view points eigenvalue problem or linear systems. The only possible difficulty is that for the linear systems approach, the original problem is homogeneous ($b = 0$) and the projected problem is not necessarily singular.

We next describe a few of these approaches and describe how preconditioning can be incorporated.

5.2 Subspace Iteration

One of the simplest methods for computing invariant subspaces is the so-called subspace iteration methods well-known to the structural engineers. In its simplest form the subspace iteration can be described as follows, see [13, 26] for details.

Subspace Iteration

1. Choose an initial orthonormal system $V_0 \equiv [v_1, v_2, \ldots, v_m]$ and an integer k;
2. Compute $X = A^k V_0$ and orthonormalize X to get V.
3. Perform an orthogonal projection process onto $span\{V\}$.
4. Test for convergence. If satisfied then exit else continue.
5. Take $V_0 = VU$, the set of approximate Schur vectors, (alternatively take $V_0 = VY$, the set of approximate eigenvectors), choose a new k and go to 2.

The above algorithm utilizes the matrix A only to compute successive matrix by vector products $w = Av$, so sparsity can be exploited. However, it is known to be a slow method, often much slower than some of the alternatives to be described next. In fact a more satisfactory alternative is to use a Chebyshev-Subspace iteration: step 2 is replaced by $X = t_k(A)V_0$, where t_k is obtained from the Chebyshev polynomial of the first kind of degree k, by a linear change of variables. The three-term recurrence of Chebyshev polynomial allows to compute a vector $w = t_k(A)v$ at almost the same cost as $A^k v$. Performance can be dramatically improved. Details on implementation and some experiments are described in [22].

5.3 Arnoldi's method

A second technique used in the literature is Arnoldi's method [1, 24] which is an orthogonal projection process onto $K_m = span\{v_1, Av_1, \ldots, A^{m-1}v_1\}$. The algorithm starts with some non-zero vector v_1 and generates the sequence of vectors v_i from the following algorithm,

Algorithm: Arnoldi

1. *Initialize:*

   Choose an initial vector v_1 of norm unity.

2. *Iterate:* Do $j = 1, 2, ..., m$

   1. Compute $w := Av_j$
   2. Compute a set of j coefficients h_{ij} so that

$$w := w - \sum_{i=1}^{j} h_{ij} v_i \tag{39}$$

      is orthogonal to all previous v_i's.
   3. Compute $h_{j+1,j} = \|w\|_2$ and $v_{j+1} = w/h_{j+1,j}$.

By construction, the above algorithm produces an orthonormal basis of the Krylov subspace $K_m = span\{v_1, Av_1, \ldots, A^{m-1}v_1\}$. The $m \times m$ upper Hessenberg matrix H_m consisting of the coefficients h_{ij} computed by the algorithm represents the restriction of the linear transformation A to the subspace K_m, with respect to this basis, i.e., we have $H_m = V_m^T A V_m$, where $V_m = [v_1, v_2, \ldots, v_m]$. Approximations to some of the eigenvalues of A can be obtained from the eigenvalues of H_m. This is Arnoldi's method in its simplest form.

Note the useful relation

$$AV_m = V_{m+1}\bar{H}_m \tag{40}$$

where $\bar{H}_m$ is the $(m+1) \times m$ upper Hessenberg matrix whose non-zero elements are the h_{ij} defined in the above algorithm. In other words $\bar{H}_m$ is obtained from H_m by appending to it the row $[0, 0, \ldots, 0, h_{m+1,m}]$.

As m increases, the eigenvalues of H_m that are located in the outmost part of the spectrum start converging towards corresponding eigenvalues of A. In practice, however, one difficulty with the above algorithm is that as m increases cost and storage increase rapidly. One solution is to use the method iteratively: m is fixed and the initial vector v_1 is taken at each new iteration as a linear combination of some of the approximate eigenvectors. Moreover, there are several ways of accelerating convergence by preprocessing v_1 by a Chebyshev iteration before restarting, i.e., by taking $v_1 = t_k(A)z$ where z is again a linear combination of eigenvectors.

A technique related to Arnoldi's method is the non-symmetric Lanczos algorithm [17, 8] which delivers a non-symmetric tridiagonal matrix instead of a Hessenberg matrix. Unlike Arnoldi's process, this method requires multiplications by both A and A^T at every step. On the other hand it has the big advantage of requiring little storage (5 vectors). Although no comparisons of the performances of the Lanczos and the Arnoldi type algorithms have been made, the Lanczos methods are usually recommended whenever the

number of eigenvalues to be computed is large which does not correspond to the situation under consideration.

5.4 Preconditioned GMRES for singular systems

In this section we adopt the viewpoint that we are trying to solve the homogeneous system

$$Ax = 0 \tag{41}$$

The case of interest to us is when there is a non-trivial solution to (41), i.e., when A is singular. Then the solution is clearly non-unique and one may wonder whether or not this can cause the corresponding iterative schemes to fail. The answer is usually no and we will illustrate in this section how standard Krylov subspace methods can be used to solve (41). We start by describing the GMRES algorithm for solving the more common linear system

$$Ax = b. \tag{42}$$

in which A is non-singular. GMRES is a least squares procedure for solving (42) on the Krylov subspace K_m. More precisely, assume that we are given an initial guess x_0 to (42) with residual $r_0 = b - Ax_0$. Let us take $v_1 = r_0/\|r_0\|_2$ and perform m steps of Arnoldi's method as described earlier. We seek an approximation to (42) of the form $x_m = x_0 + \delta_m$ where δ_m belongs to K_m. Moreover, we need this approximation to minimize the residual norm over K_m. Writing $\delta_m = V_m y_m$ we see that y_m must minimize the following function of y,

$$\begin{aligned} J(y) &= \|b - A(x_0 + V_m y)\|_2 \\ &= \|r_0 - AV_m y\|_2 \\ &= \|\|r_0\|e_1 - AV_m y\|_2 \end{aligned} \tag{43}$$

Using the relation (40) and letting $\beta \equiv \|r_0\|_2$ this becomes

$$J(y) = \|V_{m+1}[\beta e_1 - \bar{H}_m y]\|_2 = \|\beta e_1 - \bar{H}_m y\|_2 \tag{44}$$

by the orthogonality of V_{m+1}. As a result the vector y_m can be obtained inexpensively by solving a $(m+1) \times m$ least squares problem. We should point out that this procedure is also a projection process. More precisely, as is well-known the minimization of $J(y)$ is equivalent to imposing the Gram condition that

$$r_0 - AV_m y \perp v \quad \forall v \in \quad \text{span}\{AV_m\}$$

which means that we are solving $A\delta = r_0$ with a projection process with

$$K = span\{r_0, Ar_0, \cdots, A^{m-1}r_0\}$$

and $L = AK$.

A brief description of the GMRES algorithm follows. Details can be found in [25].

Algorithm: Preconditioned GMRES

1. *Start:* Choose x_0 and a dimension m of the Krylov subspaces.

2. *Arnoldi process:*

   - Compute $r_0 = b - Ax_0$, $\beta = \|r_0\|_2$ and $v_1 = r_0/\beta$.
   - For $j = 1, 2, .., m$ do:

$$\begin{aligned} h_{i,j} &= (Av_j, v_i), \quad i = 1, 2, ..., j, \\ \hat{v}_{j+1} &= Av_j - \sum_{i=1}^{j} h_{i,j} v_i \\ h_{j+1,j} &= \|\hat{v}_{j+1}\|_2, \\ v_{j+1} &= \hat{v}_{j+1}/h_{j+1,j}. \end{aligned} \tag{45}$$

   Define H_m as the $(m+1)$ x m matrix whose non-zero entries are the coefficients h_{ij}.

3. *Form the approximate solution:*

   - Find the vector y_m which minimizes the function $J(y) = \|\beta e_1 - H_m y\|_2$ where $e_1 = [1, 0, \ldots 0]^T$, among all vectors of R^m.
   - Compute $x_m = x_0 + V_m y_m$

4. *Restart:* If satisfied stop, else set $x_0 \leftarrow x_m$ and goto 2.

Each outer loop of the above algorithm, i.e., the loop consisting of steps 2, 3, and 4, is divided in two main stages. The first stage is an Arnoldi step and consists of building a basis of the Krylov subspace K_m. The second consists of finding in the affine space $x_0 + K_m$ the approximate solution x_m which minimizes the residual norm. This is found by solving the least squares problem of size $m + 1$ of step 3, whose coefficient matrix is the upper Hessenberg matrix H_m.

Note that in practice one computes progressively the least squares solution y_m in the successive steps $j = 1, \ldots m$ of stage 2. This allows to obtain at every step and at virtually no additional cost, the residual norm of the corresponding approximate solution x_k without having to actually compute it, see [25]. As a result one can stop as soon as the desired accuracy is achieved.

GMRES is theoretically equivalent to GCR [10] and to ORTHODIR [12] but is less costly both in terms of storage and arithmetic [25]. Moreover, it can be shown that, in exact arithmetic, the method cannot break down although it may be very slow or even stagnate in cases when the matrix is not positive real.

We now go back to the situation of singular systems of the form (41). In the case where the right-hand side b is zero and the matrix A is singular, the above algorithm can be used as is to compute the approximate solution of (41). The only condition is to take $x_0 \neq 0$ to avoid a break down in the first step. From what was said earlier, the algorithm will compute an approximate solution by attempting to minimize $\|Ax\|_2$ over the affine subspace $x_0 + span\{r_0, Ar_0, ..., A^{m-1}r_0\}$ which is typically of dimension m when $r_0 = Ax_0$ is non-zero. Thus whenever $x_0 \neq 0$ one can expect the method to work without any difference with the non-homogeneous case. It is subject to the same conditions of breakdown as the usual GMRES algorithm for general linear systems: the only possible cases of break-down is when the inital vector r_0 has minimal degree not exceeding $m-1$ with respect to A. In this case K_m becomes invariant under A and the algorithm stops prematurely delivering the exact solution. However, this happens very rarely in practice.

5.5 Preconditioned Arnoldi and GMRES Algorithms

Preconditioning techniques can also be used to improve the convergence rates of Arnoldi's method and GMRES. This typically amounts to replacing the original system (41) by, for example, the system,

$$M^{-1}Ax = 0 \tag{46}$$

where M is a matrix such that $M^{-1}w$ is inexpensive to compute for any vector w.

Thus in both the Arnoldi and the GMRES case, we only need to replace the original matrix A in the corresponding algorithm by the preconditioned matrix $M^{-1}A$. We may also precondition from the right, i.e., we may replace A by AM^{-1}. In this situation if $AM^{-1}z = 0$ we should note that the solution to the original problem is $M^{-1}z$, which require one additional solve with M. If $M = LU$ then the preconditioning can also be split between left and right, by replacing the original matrix by the preconditioned matrix $L^{-1}AU^{-1}$.

6 Numerical tests

In this section we report on some numerical tests to compare the methods described in this paper. We consider three realistic test problems arising from three different applications. The tables at the end of this paper present the results obtained when different numerical solution procedures were used to solve these queueing models. The tests were conducted on a Ardent Titan superworskation with two processors using double precision. The compiler optimization option used was always -O3, i.e., the highest. Before examining the examples and the results, we would like to comment on the accuracy of the results obtained. Observe that in table 1, the residual norm is not always less than 10^{-10}, the tolerance requested of the method. For example, using SOR with $\omega = 1.5$, the residual after the maximum number of iteration permitted (1000) is 0.140E-03. This should *not* be taken to mean that the computed solution is correct to three decimal places. In fact, in this particular example, an examination of the queue length distributions shows that the computed probability vector has *no* decimal places of accuracy. The reason is that this problem is somewhat ill-conditioned, and a small residual does not necessarily indicate a

small error in the solution. We cross-validated all our results and it was found that in the tables that follow, the answers obtained are indeed correct when the method converges in less than the maximum number of iterations. No conclusions should be drawn about the accuracy of the solution in other cases. Note that this does not exclude the possibility of having some decimal digits correct when the maximum number is reached, as indeed is the case of example 1 using SOR with $\omega = 1.9$. However, the reader is urged to use caution in interpreting results in instances in which the maximum number of iterations was reached.

6.1 Example 1: An Interactive Computer System.

The model described in Figure 1 below represents the system architecture of a time-shared, multiprogrammed, paged, virtual memory computer.

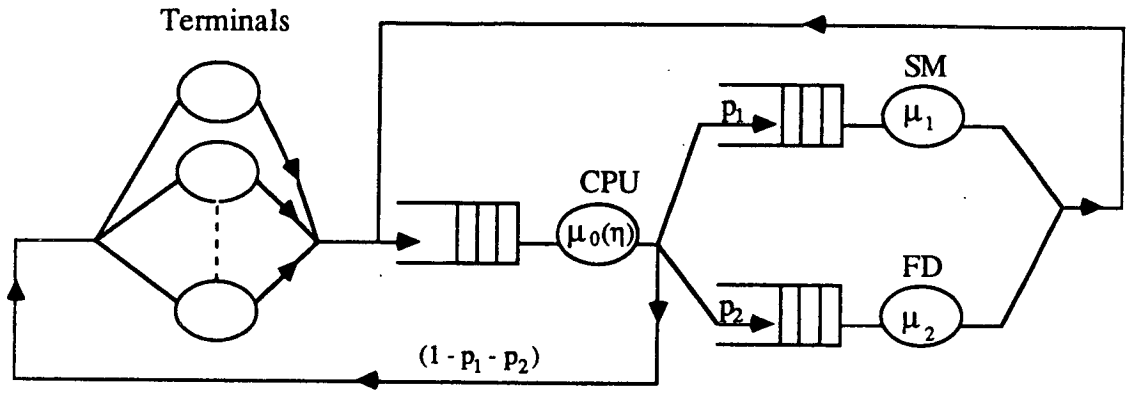

Figure 1: Illustration for example 1

The system consists of

- a set of N terminals from which N users generate commands
- a central processing unit, (CPU)
- a secondary memory device, (SM)
- a filing device, (FD).

A queue of requests is associated with each device and the scheduling is assumed to be FCFS (First Come First Served). When a command is generated, the user at the terminal remains inactive until the system responds. Symbolically, a user having generated a command enters the CPU queue. The behavior of the process in the system is characterized by a compute time followed either by a page fault, after which the process enters the SM queue, or an input/output (file request) in which case the process enters the FD queue. Processes which terminate their service at the SM or FD queue return to the CPU queue. Symbolically, completion of a command is represented by a departure of the process from the CPU to the terminals.

The degree of multiprogramming at any time is given by $\eta = n_0 + n_1 + n_2$, where n_0, n_1 and n_2 are respectively the number of processes in the CPU, SM and FD queues at that moment. If $(\mu_0(\eta))^{-1}$ is the mean service time at the CPU when the degree of multiprogramming is η, then the probabilities that a process leaving the CPU will direct itself to the SM device or to the FD device are respectively given by $p_1(\eta) = (\mu_0 q(\eta))^{-1}$ and $p_2(\eta) = (\mu_0 r(\eta))^{-1}$, where $q(\eta)$ is the mean compute time between page faults, and $r(\eta)$ is the mean compute time between i/o requests. The probability that the process will depart from the CPU queue to the terminals is given by $p_0(\eta) = (\mu_0 c(\eta))^{-1} = 1 - p_1(\eta) - p_2(\eta)$, where $c(\eta)$ is the mean compute time of a process.

The parameter q may be represented as the Belady-Kuehner lifetime function, [4], which for a process executing in memory space m is given by $q = \alpha(m)^k$, where α depends on the processing speed as well as on program characteristics, and k depends both on program locality and on the memory management strategy. If it is assumed that the total primary memory available is of size M and that it is equally shared among processes currently executing in the system, then $q(\eta) = (M/\eta)^k$.

In order to perform the numerical analysis the model parameters were assigned specific values. The mean compute time between page faults $q(\eta)$, was obtained by setting $\alpha = 0.01, M = 128$, and $k = 1.5$ so that $p_1\mu_0 = (q(\eta))^{-1} = 100(\eta/128)^{1.5}$. The mean compute time between two i/o requests $r(\eta)$, was taken as 20 msec. so that $p_2\mu_0 = 0.05$, and the mean compute time of a process, $c(\eta)$ was taken equal to 500 msec giving $p_0\mu_0 = 0.002$. The mean think time of a user at a terminal was estimated to be of the order of $\lambda^{-1} = 10$ secs, the mean service time of the SM was taken as $\mu_1^{-1} = 5$ msec and that of the FD to be $\mu_2^{-1} = 30$ msec.

The model was solved for 20 users in the system, yielding a stochastic matrix of order 1,771 with 11,011 non-zero elements and also for the case of 50 users, yielding a matrix of order 23,426 with 156,026 non-zero elements.

It should be noted that this model is not amenable to solution by analytic techniques, since the CPU service time distribution depends on the degree of multiprogramming η, i.e. on the sum of the number of processes in three distinct queues.

6.1.1 Results for Example 1

We now begin our examination of the results of example 1. The GE method requires the least amount of time, but the largest amount of additional memory. This is what we should expect.

The SOR and SSOR methods do not perform satisfactorily at all. This is because the matrix is nearly completely decomposable into 21 components, yielding 21 eigenvalues pathologically close to unity.

The only fixed point iteration scheme that is successful is when a threshold based preconditioner is used, with a threshold value of $\tau = 10^{-4}$. Here the time is about four times as long as GE but the additional memory requirement is 20 percent that of GE.

The method of Arnoldi fails completely to converge. However, PCARN (Preconditioned Arnoldi) works rather well for all preconditioners and, at least for this example, appears to provide the best processing time/memory trade-off. In fact, in all of the tests, this method is the only one that never failed to yield satisfactory results. We shall see

that in one case it had not quite converged in the maximum number of iterations, but even here, the method was in the process of converging.

Preconditioned GMRES performs satisfactory for all but a few cases. Its performance using the ILUK preconditioner is very similar to that of PCARN. In other cases it performs less well than PCARN.

Finally, GMRES preconditioned with SOR fails completely.

Only a selected few methods were applied to the solution of the larger instance of this model, (with 50 customers and a stochastic matrix of order 23,426 with 156,026 non-zero elements). In this case, the parameters of PCARN and GMRES were chosen so that the additionally memory is roughly equivalent for all preconditioners. PCARN with either ILU0 or ILUK appear to be the winners.

We were interested in finding out how the methods would compare when this example was not NCD. So we artificially adjusted the parameters of the model to ensure this would not happen, (by setting the mean think time, $\lambda = .01$ and $p_1\mu_0 = 1$). Tables 3 and 4 give the results obtained under these circumstances. Note that all the methods now work, for all cases, and it becomes extremely difficult to choose a best method.

6.2 Example 2: A Telecommunications Model.

The model in the figure below has been used to determine the effect of impatient telephone customers on a computerized telephone exchange [5]. In this model a request is made by a customer for service. The customer is prepared to wait for a certain period of time to get a reply. If at the end of that 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.

Station S2 represents a node dedicated to a special processing task and required by all customers. These customers are processed by a single server according to a processor sharing discipline. Each customer possesses a limited amount of patience which is defined as an upper bound on its service duration; when his patience is exhausted, the customer simply gives up processing.

This impatient customer may simply quit the network (with a fixed probability, $1 - h$; otherwise it joins an infinite server station S1 where it remains for a certain period, called the thinking time, and then re-joins station S2 for another attempt.

A state of this network may be described by the pair (i, j) where i, (respectively j) is the number of customers in station S1, (respectively S2)

When $j \geq 1$, the probability of

- a service completion in S2 between t and $t + dt$ is μdt
- a departure due to impatience between t and $t + dt$ is $j\tau dt$.

When $i \geq 1$, the probability of a departure from S1 between t and $t + dt$ is $i\lambda dt$.

External arrivals to S2 are assumed to be Poisson at rate A

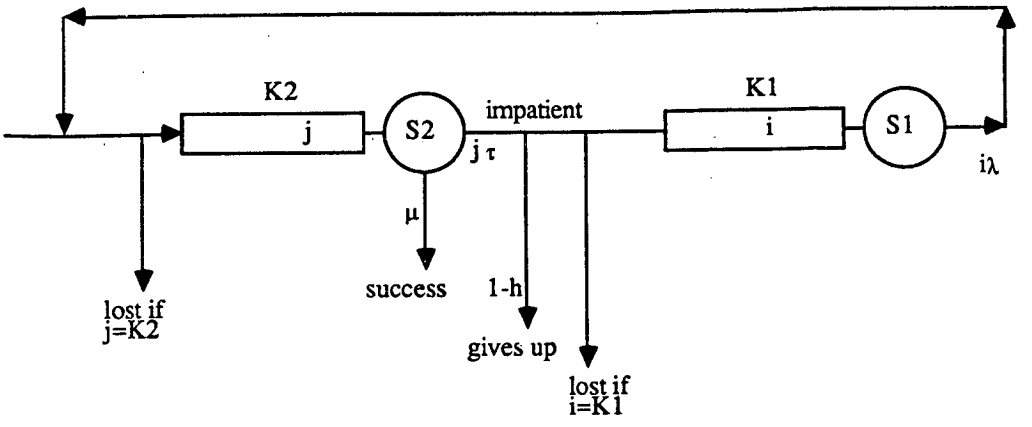

Figure 2: Illustration for example 2

To obtain a finite Markov chain, we assume that K1 is the maximum number of customers permitted in station S1 and K2 is the maximum permitted in S2. Customers arriving to a full station are lost. In the model, it is important to choose values of K1 and K2 large enough so that the probability of saturation is negligible, say less than 10^{-10}. This truncation of the state space will therefore have little effect and the resulting steady state probabilities may be taken as an accurate approximation of those of the infinite capacity network.

The following are realistic values as taken from the various reports of Boyer and his colleagues.

$A = 0.6; \mu = 1.0; \tau = 0.05; h = 0.85$ and $\lambda = 5.0$.

They are the values which we used in our experiments. The values of K1 and K2 were varied to obtain matrices of different order. First we set $K1 = 10$ and $K2 = 220$ which gave a stochastic matrix of order 2,431 with 11,681 non-zero elements and secondly we used the values 30 and 550 respectively, which resulted in 17,081 states and 84,211 non-zero elements.

6.2.1 Results for Example 2

Tables 5 and 6 below show the results obtained for this example.

On both the large and the small example, the method of Gaussian elimination performs exceptionally well from the point of view of computation time. More surprisingly, the amount of additional memory required was smaller that that needed by the preconditioned Arnoldi and GMRES for the smaller case. It was less than twice that needed by these methods for the larger case. For this example, GE must be considered the method of choice. A close investigation of the transition matrix shows that it has a rather narrow bandwidth structure which, as we have already discussed, is ideal for the GE method.

Once again the SOR methods are not very successful. An examination of the results does however show that some small number of digits of accuracy have been obtained

which indicates that given sufficient iterations these methods might indeed converge.

The fixed point iterations succeed, by and large, in obtaining the correct solution in under the maximum number of iterations permitted, but the time taken is very long when compared to GE.

The method of Arnoldi fails complete. However the preconditioned Arnoldi provides the only competition for GE. In an about face for iterative methods, they require more additional memory than GE.

For two sets of parameters, the preconditioned GMRES methods are comparable with PCARN. However, in the other cases they fail. PCARN must be considered to the more robust. GMRES preconditioned with SOR and SSOR is not any more successful.

The same type of comments can be made about the results obtained for the larger model. The only modification is that in two cases, the fixed point iterations become more competitive with the preconditioned Arnoldi.

6.3 Example 3: A Multi-Class, Finite Buffer, Priority System.

This model, like model number 2, is also taken from the telecommunications literature. The model consists of a single service center at which two identical servers provide service to two different classes of customer. The service rates may differ for each class (μ_1 and μ_2, but both are exponentially distributed. The arrival processes of the two classes is not exponential. It has often been observed that teletraffic is rather bursty in nature and to take this into effect, hyper-exponential interarrival times with large coefficients of variance have been associated with these arrival processes.

Class-one customers are assumed to have a high priority. An arriving class-one customer is inserted in the queue before all class-two customers. An idle server will only serve a class-two customer if there are no class-one customers waiting. However, once a server begins to provide service to a class-two customer, it will continue to serve that customer even if a class-one customers arrives and is forced to wait; in other words, the service is non-preemptive.

The effect of the limited capacity buffer is to restrict the number of customers that can enter the system. Class-two customers that arrive to a full buffer are simply lost. If the buffer is full and contains both class-one and class-two customers, an arriving class-one customer will displace a class two-customer. This class two-customer is therefore lost. A class-one customer that arrives to a system that is full of class one-customers is lost.

Figure 3 represents, schematically, this model.

A six-component vector is required to represent any state of the Markov chain which underlies this model. Components 1 and 2 may be used to denote the phase of the arrival process for each of the two classes respectively. Similarly, components 3 and 4 may be used to represent the number of customers of class 1 and class 2 that are already in the system. Components 5 and 6 may be used to indicate the condition of the two servers (viz: idle, serving a class 1 customer, serving a class 2 customer). Since the buffer is finite, only a finite number of states will be generated. As in the other two examples, we generated two different sizes of Markov chains from this model. First we set the bufer

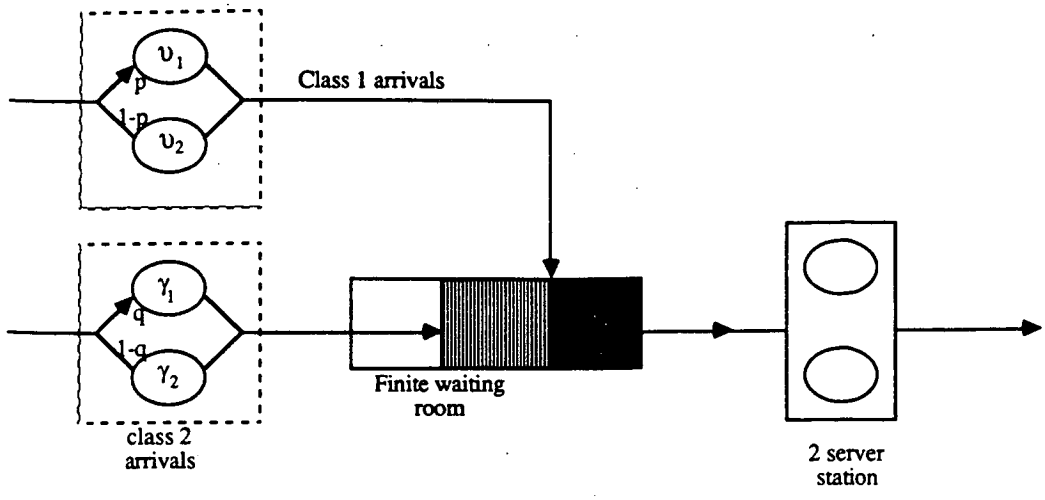

Figure 3: Illustration for example 3

size to 16, which generated 1,940 states and 12,824 non-zero elements. Second we set the buffer size to 50 and obtained a matrix of order 19,620 with 131,620 non-zero elements.

The following values for the parameters indicated on the figure were obtained from Perros, [19], and are representative of values currently used by teletraffic specialists.

$\nu_1 = .00138; \nu_2 = .0000000076; p = .9999, \gamma_1 = .00396; \gamma_2 = .000000018; q = .999995, \mu_1 = 0.002222; \mu_2 = 0.002222.$

6.3.1 Results for Example 3

Tables 7 and 8 below show the results obtained for this example.

This example shows the GE method in its worst light. The time is greater than that of most of the preconditioned Arnoldi and GMRES methods. Worse than that, its additional memory requirements are an order of magnitude greater. This matrix is not as well structured as that of example 2 and consequently a lot of fill-in occurs. Once again the (S)SOR methods fail to converge. The same thing happens when they are used as preconditioners for GMRES. Preconditioned Arnoldi and GMRES perform best, with the sole exception of the ILU0 preconditioner and GMRES. The fixed point iteration method also fails when this preconditioner is used. It also fails when the preconditioner used if that obtained when only the largest five elements per row are kept.

In the larger example, the fixed point iteration method performs well. Observe however, the rather large amount of additional memory needed when a threshold of 10^{-3} is used in ILUTH. We would like to point out that although PCARN failed to satisfy the tolerance criteria in less that 1000 iterations, it had in fact almost converged.

An interesting observation may be made about preconditioners based on the results presented in tables 7 and 8. When the ILUTH preconditioner is used with PCARN or GMRES (see table 7) or with FXPTIT (see table 8) a "better" preconditioner obtained

using $\tau = 0.001$ performs less well than the preconditioner obtained with $\tau = 0.01$. With the smaller threshold value we normally expect that, in the incomplete decomposition of Q^T into $LU + E$, E will be smaller. However, because this example is ill-conditioned, $U^{-1}L^{-1}$ is not necessarily closer to the inverse of Q^T than the decomposition obtained with the larger threshold. We conclude that, for NCD problems, a smaller remainder in an incomplete factorization does not necessarily yield a better preconditioner. We have observed that in cases like this, when a very small threshold is specified, the residual will drop steeply after the first iteration but will not improve substantially beyond that. Most often the residual oscillates around the value it acquires after the first iteration.

This example, like example 1, is also nearly completely decomposable. If we proceed as we did with example 1 and modify the parameters to make it non-NCD, we get the results that are in tables 9 and 10. In this case all of the iterative methods behave satisfactorily.

7 Conclusion

In this paper we explored a wide variety of methods for the numerical solution of Markov chains. We tested these methods on three realistic problems. The question now arises as to which method is to be recommended to our readers? Unfortunately our results do not support the hypothesis that a single "black box" method is available. When the state space is small, or even for moderately sized problems in which the non-zero elements lie close to the diagonal, then a direct method such as Gaussian elimination should be chosen. However, in other cases, the issue is not so clear. When the matrix is reasonably well conditioned all of the methods perform more or less satisfactorily. When the matrix becomes NCD then there is a smaller choice. If forced to make a recommendation, the most robust method appears to be Preconditioned Arnoldi. It is often the fastest, and in all cases tested either converged within the specified number of iterations or was at least close to converging when the maximum was reached. None of the other methods can make this claim. Moreover, note that for the large problems, the preprocessing time to compute the ILUK and ILUTH incomplete factorizations can be high and even often far exceeds the time required in the iteration phase. It is possible that our implementations of the ILUK and ILUTH preprocessing phase could be improved. In obtaining these incomplete factorizations, a full vector is used to reduce each row. The threshold operations and the search for the k maximum elements are performed over this vector from the first to the last non-zero position. If the reduced row contains many zeros, some savings could result by first compacting the row.

We did not cover every possible solution method in this paper. Simultaneous iteration methods were not included because our experience over several years indicates that this is inferior to Arnoldi. The Bi-Conjugate Gradient method and the Conjugate gradient squared method, methods which have had much success in other domains, [21]. were not included. In fact in our initial study, we programmed both these methods but found them unsatisfactory. Both failed to converge when applied to NCD problems and in other cases they performed less well than the methods examined in this report. Potentially com-

petitive alternatives include the techniques based on polynomial acceleration of Arnoldi's method such as a hybrid Chebyshev-Arnoldi algorithm [22]. As a general rule however, we observe that the preconditioner makes a bigger difference than the acceleration procedure itself. Thus, in many cases there is hardly any difference between the performance of GMRES and PCARN when ILUTH is used with a small tolerance. When the preconditioner is excellent then the number of iterations required for convergence is very small and we expect that whichever iterative procedure is used, it will remain small.

Finally, we have not discussed domain decomposition type methods which typically go under the name of aggregation/disaggregation methods, or iterative aggregation methods, [6]. These methods are particularly well suited to matrices that are NCD. However, at each step of these methods we need to find the stationary probability vector of a stochastic matrix and to solve several systems of equations in which the coefficient matrix is almost stochastic and the right-hand side is small. These solutions must be obtained by the methods discussed in this paper.

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 and V. A. Barker. *Finite Element Solution of Boundary Value Problems.* Academic Press, Orlando, Florida, 1984.

[3] V.A. Barker Numerical Solution of Sparse Singular Systems of Equations Arising from Ergodic Markov Chains. *Stochastic Models*, Vol.5. No. 3, 1989.

[4] L.A. Belady and C.J. Kuehner. Dynamic Space Sharing in Computer Systems. *Comm. ACM*, Vol 12, No. 5, 1969, pp 282-288.

[5] P. Boyer, A. Dupuis and A. Khelladi *A Simple Model for Repeated Calls due to Time-Outs.* CNET, LAA/SLC/EVP, Route de Tregastel, 22301 Lannion, France.

[6] W.L. Cao and W.J. Stewart. Iterative Aggregation/Disaggregation Techniques for Nearly Uncoupled Markov Chains *JACM*, Vol. 32, No. 3, 1985, pp. 702-719.

[7] F. Chatelin. *Spectral Approximation of Linear Operators.* Academic Press, New York, 1984.

[8] J. Cullum and R. Willoughby. A Lanczos procedure for the modal analysis of very large non-symmetric matrices. In *Proceedings of the 23rd Conference on Decision and Control, Las Vegas*, 1984.

[9] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices.* Clarendon Press, Oxford, 1986.

[10] H.C. Elman. *Iterative Methods for Large Sparse Nonsymmetric Systems of Linear Equations.* PhD thesis, Yale University, Computer Science Dpt., New Haven, CT., 1982.

[11] R.E. Funderlic and R.J. Plemmons. A Combined Direct-Iterative Method for Certain M-Matrix Linear Systems. *SIAM J. Alg. Disc. Meth.*, Vol. 5, No. 1, 1984, pp. 32-42.

[12] A.L. Hageman and D.M. Young. *Applied Iterative Methods.* Academic Press, New York, 1981.

[13] A. Jennings and W.J. Stewart. A simultaneous iteration algorithm for real matrices. *ACM, Trans. of Math. Software*, 7:184–198, 1981.

[14] 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. Comp.*, 31(137):148–162, 1977.

[15] D. Mitra and P. Tsoucas. Relaxations for the Numerical Solutions of Some Stochastic Problems. *Stochastic Models*, 4, 3. Nov. 1988.

[16] E. Ng. Comparison of Direct Sparse Solvers. Presented at the *SIAM conference on Sparse Matrices*, Glenenden Beach, Oregon, May 22-24, 1989.

[17] B. N. Parlett and D. R. Taylor andf Z. S. Liu. A look-ahead Lanczos algorithm for non-symmetric matrices. *Mathematics of Computation*, 44:105–124, 1985.

[18] B.N. Parlett and Y. Saad. *Complex Shift and Invert Strategies for Real Matrices.* Technical Report YALEU/ DCS-RR-424, Yale University, Computer Science Dept., New-Haven, Connecticut, 1985.

[19] H.G. Perros Private Communication.

[20] S. Pissanetzky *Sparse Matrix Technology* Academic Press, 1984.

[21] G. Radicati and Y. Robert. *A Comparison of Conjugate Gradient-like Algorithms for Solving Sparse Nonsymmetric Linear Systems. A Vector and Parallel Implementation.* Technical Report Laboratoire TIM3, institut National Polytechnique de Grenoble, France, 1987.

[22] Y. Saad. Chebyshev acceleration techniques for solving non-symmetric eigenvalue problems. *Mathematics of Computation*, 42:567–588, 1984.

[23] Y. Saad. Projection methods for solving large sparse eigenvalue problems. In B. Kagstrom and A. Ruhe, editors, *Matrix Pencils, proceedings, Pitea Havsbad*, pages 121–144, University of Umea, Sweden, Springer Verlag, Berlin, 1982. Lecture notes in Math. Series, Number= 973.

[24] Y. Saad. Variations on arnoldi's method for computing eigenelements of large unsymmetric matrices. *Lin. Alg. Appl.*, 34:269–295, 1980.

[25] Y. Saad and M.H. Schultz. GMRES: a generalized minimal residual algorithm for solving non-symmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7:856–869, 1986.

[26] G.W. Stewart. Simultaneous iteration for computing invariant subspaces of non-Hermitian matrices. *Numer. Mat.*, 25:123–136, 1976.

[27] D.M. Young. *Iterative solution of large linear systems.* Academic Press, New York, 1971.

[28] J. H. Wilkinson. *The Algebraic Eigenvalue Problem.* Clarendon Press, Oxford, 1965.

NOTATION FOR TABLES

METHODS:

- GE — Gaussian elimination (section 3)
- SOR — Successive overrelaxation (Section 4.2)
- SSOR — Symmetric SOR (Section 4.3)
- FXPTIT — Fixed point iteration (Section 4.4)
- ARNOLDI — Arnoldi's method (Section 5.3)
- PCARN — Preconditioned Arnoldi (Section 5.5)
- GMRES — Preconditioned GMRES (Section 5.4)

PRECONDITIONERS: (See section 4.6)

- ILU0 — Incomplete factorization with zero fill-in
- ILUK — Incomplete factorization keeping KMAX elements/row ll
- ILUTH — Incomplete factorization with threshold

METHOD PARAMETERS:

- ω — Relaxation parameter for SOR based methods
- m — Subspace size for projection methods
- K — Maximum number of non-zero elements permitted per row
- τ — Threshold for ILUTH preconditioners

TIMES:

- *TOTAL TIME:* Total running time for method on Ardent-Titan computer.
- *SET-UP TIME:* Time taken to compute preconditioner, if applicable.
- *ITER. TIME:* Time to perform "ITERS" iterations of the method.

FLOPS: The number of floating point operations performed by method.

ADDITIONAL MEMORY: The amount of memory required by the algorithm in excess of the original matrix and a single vector.

ITERS: The number of iterations performed by the method. An asterisk before this number means that it is the maximum number of iterations permitted.

RESIDUAL NORM: The two norm of the product of the computed solution and transition rate matrix.

The caption under each table indicates the order of the matrix N, and the number NZ of non-zero elements that it contains.

Method	Method Parameters	Total Time	Set-up Time	Iter. Time	Flops	Additional Memory	Iters	Residual Norm
GE		3.3			4.7	111,989		0.219E-16
SOR	$\omega = 1$	79.4				1,771	*1000	0.494E-04
	$\omega = 1.5$	79.4				1,771	*1000	0.140E-03
	$\omega = 1.9$	79.3				1,771	*1000	0.105E-05
	$\omega = 1.95$	57.7			9.3	1,771	726	0.739E-11
SSOR	$\omega = 1.0$	78.0				1,771	*500	0.509E-04
	$\omega = 1.5$	95.9				1,771	*500	0.360E-04
	$\omega = 1.9$	93.7				1,771	*500	0.140E-03
FXPTIT /								
+ILU0		175.1	0.3			14,553	*1000	0.926E-07
+ILUK	K=5	65.7	3.4	62.2	9.4	12,330	404	0.982E-10
	K=10	177.0	5.0			21,116	*1000	0.226E-08
+ILUTH	$\tau = .01$	124.4	1.5			8,494	*1000	0.273E-07
	$\tau = .001$	77.4	1.7	75.5	11.2	14,413	441	0.997E-10
	$\tau = 10^{-4}$	12.5	2.0	75.5	1.9	21,302	59	0.840E-10
ARNOLDI	m=10	51.4			32.9	17,971	*1010	0.121E-04
	m=20	46.5			51.5	36,341	*1020	0.131E-03
	m=25	83.1			96.8	45,676	*1625	0.184E-03
PCARN/								
+ILU0	m=5	19.8	0.3	19.3	4.8	23,408	160	0.324E-10
	m=10	15.7	0.3	15.2	5.6	32,263	150	0.811E-10
+ILUK	m=10, K=5	9.1	3.3	5.6	2.5	30,050	70	0.291E-10
	m=10, K=10,	5.9	4.3	1.4	0.5	38,735	10	0.409E-11
+ILUTH	m=10, $\tau = .01$	15.1	1.6	13.4	7.1	26,104	230	0.543E-10
	m=10, $\tau = .001$	11.6	1.8	9.7	3.0	32,142	80	0.205E-10
GMRES /								
+ILU0	m=5	149.1	0.3	148.6		23,408	*1600	0.210E-06
	m=10	16.4	0.3	15.9	5.2	32,263	140	0.632E-10
	m=20	16.5	0.3	16.0	8.7	49,973	160	0.715E-10
+ILUK	m=10, K=5	7.5	3.3	4.1	1.8	30,050	50	0.922E-10
	m=10, K=10	5.8	4.2	1.4	0.5	38,735	10	0.438E-11
+ILUTH	m=10, $\tau = .01$	13.0	1.6	11.2	5.5	26,104	180	0.579E-10
	m=5, $\tau = .001$	92.4	1.7	90.5		23,287	*1000	0.298E-06
	m=10, $\tau = .001$	97.2	1.8	95.2		32,142	*1000	0.204E-06
	m=20, $\tau = .001$	9.9	1.8	8.0	4.3	49,852	80	0.746E-10
GMRES /								
+SOR	m=10, $\omega = 1.0$	94.5			37.3	17,710	*1000	0.529E-06
	m=10, $\omega = 1.95$	49.8			18.7	17,710	*500	0.416E-03

Table 1: Performance Results for Example 1. N=1,771; NZ=11,011. NCD Case.

Method	Method Parameters	Total Time	Set-up Time	Iter. Time	Flops	Additional Memory	Iters	Residual Norm
FXPTIT/								
+ILUK	K=16	2,632.0	230.6			420,896	*1000	0.587E-07
+ILUTH	$\tau = .01$	1,584.5	96.3			92,375	*1000	0.177E-05
	$\tau = .001$	830.5	99.2	731.1	114.5	201,550	320	0.998E-10
PCARN/								
+ILU0	m=10	159.1	3.2	154.5	66.1	437,138	130	0.750E-10
+ILUK	m=10, K=7	212.8	198.1	13.3	5.9	444,790	10	0.887E-11
+ILUTH	m=10, $\tau = .01$	670.3	96.0	672.8		326,635	*1000	0.918E-09
GMRES /								
+ILU0	m=10	1,184.0	3.1	1,179.5		437,138	*1000	0.201E-05
+ILUK	m=10, K=7	213.3	198.5	13.4	5.9	444,790	10	0.105E-10
+ILUTH	m=10, $\tau = .01$	761.0	96.8	662.8		326,635	*1000	0.461E-07

Table 2: Performance Results for Example 1. N=23,426; NZ=156,026. NCD Case.

Method	Method Parameters	Total Time	Set-up Time	Iter. Time	Flops	Additional Memory	Iters	Residual Norm
GE		4.3			7.2	111,989		0.139E-16
SOR	$\omega = 1$	57.0			9.1	1,771	711	0.981E-10
	$\omega = 1.2$	37.3			6.0	1,771	464	0.995E-10
	$\omega = 1.3$	31.2			5.0	1,771	387	0.999E-10
SSOR	$\omega = 1$	92.9			14.1	1,771	591	0.986E-10
	$\omega = 1.2$	95.1			14.4	1,771	605	0.984E-10
FXPTIT /								
+ILU0		43.0	0.3	42.7	6.5	14,553	254	0.989E-10
+ILUK	K=5	14.6	3.2	11.4	1.8	12,325	75	0.988E-10
	K=10	12.6	4.6	8.0	1.6	21,206	47	0.724E-10
+ILUTH	$\tau = .01$	34.5	1.8	32.6	5.4	17,385	191	0.953E-10
	$\tau = .001$	13.3	2.2	10.9	2.4	28,198	61	0.807E-10
	$\tau = 10^{-4}$	13.6	2.6	10.9	2.9	36,750	59	0.775E-10
ARNOLDI	m=10	18.5			16.6	17,971	510	0.278E-10
	m=20	21.9			28.3	36,341	560	0.138E-10
	m=25	18.8			26.8	45,676	450	0.875E-10
PCARN /								
+ILU0	m=5	5.3	0.2	4.9	1.5	23,408	50	0.559E-10
	m=10	5.0	0.3	4.6	1.9	32,263	50	0.637E-11
+ILUK	m=10, K=5	5.3	3.4	1.7	0.7	30,080	20	0.556E-10
	m=10, K=10	7.9	6.3	1.5	0.8	38,920	10	0.561E-16
+ILUTH	m=10, $\tau = .01$	4.1	1.7	2.3	0.8	35,113	20	0.101E-11
	m=10, $\tau = .001$	5.6	2.2	3.2	1.1	45,908	20	0.599E-12
GMRES /								
+ILU0	m=5	5.4	0.3	4.9	1.5	23,408	50	0.622E-10
	m=10	5.0	0.3	4.6	1.9	32,263	50	0.470E-10
	m=20	4.2	0.3	3.8	2.2	49,973	40	0.240E-12
+ILUK	m=10, K=5	5.4	3.5	1.8	0.7	30,080	20	0.531E-10
	m=10, K=10	7.8	6.3	1.4	0.7	38,920	10	0.698E-16
+ILUTH	m=10, $\tau = .01$	4.2	1.8	2.2	0.8	35,113	20	0.427E-12
	m=5, $\tau = .001$	5.7	2.2	3.4	1.0	37,053	20	0.281E-11
	m=10, $\tau = .001$	5.6	2.2	3.2	1.1	45,908	20	0.478E-12
	m=20, $\tau = .001$	5.6	2.2	3.3	1.5	63,618	20	0.515E-14
GMRES /								
+SOR	m=10, $\omega = 1.0$	16.2			6.4	17,710	170	0.101E-13
+SSOR	m=10, $\omega = 1.0$	34.2			9.4	17,710	190	0.835E-14

Table 3: Performance Results for Example 1. N=1,771; NZ=11,011 Non-NCD Case

Method	Method Parameters	Total Time	Set-up Time	Iter. Time	Flops	Additional Memory	Iters	Residual Norm
GE		Failed						
FXPTIT /								
+ILU0		477.5	3.3	473.9	75.0	202,878	206	0.964E-10
+ILUK	K=16	479.3	264.9	214.2	59.3	421,518	90	0.952E-10
+ILUTH	$\tau = .01$	493.9	100.8	392.8	68.1	245,705	169	0.915E-10
	$\tau = .001$	364.3	107.5	256.5	61.5	413,483	106	0.780E-10
PCARN/								
+ILU0	m=10	158.3	3.2	153.7	66.1	437,138	130	0.166E-10
+ILUK	m=10, K=7	239.6	188.9	49.4	21.4	444,978	40	0.821E-12
+ILUTH	m=10, $\tau = .01$	143.0	99.6	42.0	17.1	480,013	30	0.622E-10
GMRES /								
+ILU0	m=10	404.2	3.2	399.6	171.7	437,138	340	0.180E-10
+ILUK	m=10, K=7	227.5	188.9	37.2	16.1	444,978	30	0.867E-11
+ILUTH	m=10, $\tau = .01$	143.0	99.6	42.1	17.0	480,013	30	0.341E-10

Table 4: Performance Results for Example 1. N=23,426; NZ=156,026. Non-NCD Case

Method	Method Parameters	Total Time	Set-up Time	Iter. Time	Flops	Additional Memory	Iters	Residual Norm
GE		1.3			.3	28,554		0.404E-17
SOR	$\omega = 1.0$	108.5				2,431	*1000	0.910E-04
	$\omega = 1.2$	108.4				2,431	*1000	0.342E-06
	$\omega = 1.3$	104.3			13.6	2,431	962	0.946E-10
SSOR	$\omega = 1.0$	107.9				2,431	*500	0.321E-03
	$\omega = 1.2$	109.0				2,431	*500	0.896E-03
FXPTIT /								
+ILU0		83.0	0.3	82.5	9.6	16,543	342	0.992E-10
+ILUK	K=5	93.5	1.4	92.0	12.6	16,999	440	0.943E-10
	K=10	28.8	2.5	26.2	4.6	29,066	112	0.927E-10
+ILUTH	$\tau = .01$	85.7	0.6	84.9	11.0	15,051	411	0.947E-10
	$\tau = .001$	8.6	1.2	7.3	1.2	30,571	28	0.856E-10
ARNOLDI	m=10	45.6			41.6	24,571	*1010	0.334E-03
PCARN /								
+ILU0	m=10	22.1	0.3	21.6	7.6	40,853	160	0.536E-11
+ILUK	m=10, K=5	12.0	1.4	10.4	4.8	41,258	100	0.306E-10
	m=10, K=10	4.7	2.6	1.9	0.7	53,382	10	0.195E-12
+ILUTH	m=10, $\tau = .01$	16.4	0.7	15.5	6.0	39,371	130	0.252E-10
	m=10, $\tau = .001$	8.0	1.2	6.6	1.9	54,924	30	0.138E-11
GMRES /								
+ILU0	m=10	101.8	0.3	101.3		40,853	*1000	0.151E-05
+ILUK	m=10, K=5	102.9	1.5	101.3		41,258	*1000	0.233E-07
	m=10, K=10	4.4	2.4	1.8	0.7	53,382	10	0.177E-12
+ILUTH	m=10, $\tau = .01$	99.3	0.7	98.4		39,371	*1000	0.666E-05
	m=10, $\tau = .001$	6.6	1.2	5.2	1.9	54,924	30	0.200E-12
GMRES /								
+SOR	m=10, $\omega = 1.0$	131.2			47.4	24,310	*1000	0.486E-01
+SSOR	m=10, $\omega = 1.0$	246.4			60.3	24,310	*1000	0.202E-03

Table 5: Performance Results for Example 2. N=2,431; NZ=11,681

Method	Method Parameters	Total Time	Set-up Time	Iter. Time	Flops	Additional Memory	Iters	Residual Norm
GE		16.0			12.9	532,059		0.699E-18
FXPTIT /								
+ILU0		1,008.9	1.9	1,006.8	128.6	118,373	635	0.932E-10
+ILUK	K=16	113.8	33.2	80.4	18.0	305,460	47	0.768E-10
+ILUTH	$\tau = .01$	1,372.7	8.3			100,853	*1000	0.259E-09
	$\tau = .001$	144.0	10.7	133.2	21.8	180,126	82	0.613E-10
PCARN/								
+ILU0	m=10	237.6	1.9	234.8	111.3	289,183	330	0.129E-10
+ILUK	m=10, K=5	160.9	16.8	143.3	67.8	290,108	200	0.904E-11
+ILUTH	m=10, $\tau = .01$	229.2	8.5	219.8	111.3	271,684	350	0.961E-10
GMRES/								
+ILU0	m=10	710.2	1.9	707.4		289,183	*1000	0.375E-06
+ILUK	m=10, K=5	729.8	16.8	712.2		290,108	*1000	0.952E-08
+ILUTH	m=10, $\tau = .01$	636.4	8.5	627.1		271,684	*1000	0.502E-06

Table 6: Performance Results for Example 2. N=17,081; NZ = 84,211

Method	Method Parameters	Total Time	Set-up Time	Iter. Time	Flops	Additional Memory	Iters	Residual Norm
GE		25.0			71.8	411,823		0.115E-16
SOR	$\omega = 1$	87.9				1,940	*1000	0.312E-05
	$\omega = 1.2$	90.0				1,940	*1000	0.465E-05
	$\omega = 1.3$	92.8				1,940	*1000	0.104E+02
SSOR	$\omega = 1.0$	178.0				1,940	*1000	0.293E-05
	$\omega = 1.2$	174.6				1,940	*1000	0.857E-05
FXPTIT /								
+ILU0		171.2	0.3			16,604	*1000	0.241E-05
+ILUK	K=5	172.4	8.5			13,525	*1000	0.596E-06
	K=10	30.6	10.8	19.7	4.1	23,228	107	0.965E-10
+ILUTH	$\tau = .01$	34.5	4.5	30.0	7.4	28,218	179	0.994E-10
	$\tau = .001$	27.7	6.6	21.0	8.7	68,076	108	0.980E-10
ARNOLDI	m=10	39.2			36.8	19,661	*1010	0.337E-04
PCARN/								
+ILU0	m=10	51.3	0.3	50.9	21.8	36,104	520	0.754E-10
+ILUK	m=10, K=5	19.3	8.9	10.2	4.6	32,990	120	0.206E-10
	m=10, K=10	18.5	10.5	7.9	3.1	42,646	60	0.246E-10
+ILUTH	m=10, $\tau = 0.01$	11.3	5.0	6.1	2.1	46,597	40	0.238E-11
	m=10, $\tau = 0.001$	21.5	6.0	15.3	5.2	62,546	70	0.584E-10
GMRES /								
+ILU0	m=10	98.8	0.3	98.3		36,104	*1000	0.108E-06
+ILUK	m=10, K=5	19.5	9.4	10.0	3.9	32,990	100	0.168E-11
	m=10, K=10	18.5	10.5	7.8	3.1	42,646	60	0.692E-10
+ILUTH	m=10, $\tau = 0.01$	11.4	5.0	6.2	2.1	46,597	40	0.119E-11
	m=10, $\tau = 0.001$	19.4	6.0	13.2	4.5	62,510	60	0.917E-11
GMRES /								
+SOR	m=10, $\omega = 1.0$	101.6			41.7	19,400	*1000	0.192E-08
+SSOR	m=10, $\omega = 1.0$	194.4			55.9	19,400	*1000	0.484E-09

Table 7: Performance Results for Example 3. N=1,940; NZ=12,824. NCD Case.

Method	Method Parameters	Total Time	Set-up Time	Iter. Time	Flops	Additional Memory	Iters	Residual Norm
GE		Failed						
FXPTIT/								
+ILUK	K=16	804.7	296.3	508.0	110.2	328,701	231	0.952E-10
+ILUTH	$\tau = .01$	562.1	135.0	427.0	99.5	290,948	234	0.996E-10
	$\tau = .001$	957.7	173.6	784.0	360.0	872,623	351	0.977E-10
PCARN/								
+ILU0	m=10	1,023.8	2.4	1,020.2		367,060	*1000	0.524E-08
+ILUK	m=10, K=7	514.2	305.4	207.7	88.8	372,770	200	0.968E-10
+ILUTH	m=10, $\tau = .01$	231.6	152.3	78.2	28.8	484,133	50	0.252E-10
GMRES/								
+ILU0	m=10	1,013.6	2.4	1,010.0		367,060	*1000	0.150E-06
+ILUK	m=10, K=7	1,336.8	305.4	1,030.2		372,770	*1000	0.156E-06
+ILUTH	m=10, $\tau = .01$	232.7	152.2	79.3	28.5	484,148	50	0.852E-11

Table 8: Performance Results for Example 3. N=19,620; NZ=131,620. NCD Case.

Method	Method Parameters	Total Time	Set-up Time	Iter. Time	Flops	Additional Memory	Iters	Residual Norm
GE		25.8			74.5	411,823		0.159E-16
SOR	$\omega = 1$	12.4			2.1	1,940	139	0.913E-10
	$\omega = 1.2$	8.4			1.4	1,940	93	0.801E-10
	$\omega = 1.3$	9.6			1.6	1,940	107	0.828E-10
SSOR	$\omega = 1$	13.5			2.1	1,940	77	0.856E-10
	$\omega = 1.2$	9.9			1.6	1,940	56	0.979E-10
FXPTIT /								
+ILU0		11.0	0.3	10.7	1.8	16,704	63	0.959E-10
+ILUK	K=5	21.4	8.1	13.2	2.2	13,486	83	0.790E-10
	K=10	18.8	9.8	8.9	1.9	23,257	49	0.835E-10
+ILUTH	$\tau = .01$	10.3	4.8	5.5	1.6	36,937	29	0.859E-10
	$\tau = .001$	16.6	12.8	3.8	3.0	128,199	16	0.640E-10
ARNOLDI	m=10	4.8			4.4	19,661	120	0.411E-10
PCARN /								
+ILU0	m=10	3.5	0.3	3.1	1.3	36,104	30	0.420E-12
+ILUK	m=10, K=5	11.6	8.8	2.7	1.2	32,990	30	0.369E-10
	m=10, K=10	16.5	13.7	2.7	1.4	42,640	20	0.154E-10
+ILUTH	m=10, $\tau = .01$	9.5	5.4	4.0	1.4	56,178	20	0.217E-10
	m=10, $\tau = .001$	23.6	12.2	11.2	4.7	139,185	20	0.889E-13
GMRES /								
+ILU0	m=10	3.5	0.2	3.1	1.3	36,104	30	0.502E-12
+ILUK	m=10, K=5	11.5	8.8	2.6	1.2	32,990	30	0.217E-10
	m=10, K=10	16.6	13.8	2.7	1.4	42,640	20	0.133E-10
+ILUTH	m=10, $\tau = .01$	9.4	5.4	3.9	1.4	56,178	20	0.861E-11
	m=10, $\tau = .001$	23.6	12.3	11.2	4.7	139,189	20	0.650E-13
GMRES /								
+SOR	m=10, $\omega = 1.0$	17.4			7.0	19,400	168	0.707E-14
+SSOR	m=10, $\omega = 1.0$	8.2			2.3	19,400	40	0.974E-14

Table 9: Performance Results for Example 3. N=1,940; NZ=12,824. Non-NCD Case

Method	Method Parameters	Total Time	Set-up Time	Iter. Time	Flops	Additional Memory	Iters	Residual Norm
GE		Failed						
FXPTIT /								
+ILU0		179.9	3.1	176.6	31.4	170,860	104	0.848E-10
+ILUK	K=16	398.7	293.6	104.9	29.7	329,692	55	0.648E-10
+ILUTH	$\tau = .01$	219.3	137.9	81.2	23.0	376,620	43	0.726E-10
PCARN /								
+ILU0	m=10	74.7	2.4	71.0	29.9	367,060	70	0.650E-10
+ILUK	m=10, K=7	371.3	317.8	52.2	25.1	372,774	50	0.332E-11
+ILUTH	m=10, $\tau = .01$	215.8	155.1	59.5	21.1	572,967	30	0.252E-10
GMRES /								
+ILU0	m=10	105.3	2.4	101.7	42.5	367,060	100	0.827E-11
+ILUK	m=10, K=7	391.8	317.9	72.8	33.6	372,774	70	0.159E-10
+ILUTH	m=10, $\tau = .01$	215.8	155.1	59.4	21.0	572,966	30	0.844E-11

Table 10: Performance Results for Example 3. N=19,620; NZ=131,620. Non-NCD Case.

LISTE DES DERNIERES PUBLICATIONS INTERNES

PI 486 **SYNTHESIS OF A NEW SYSTOLIC ARCHITECTURE FOR THE ALGEBRAIC PATH PROBLEM**
Abdelhamid BENAINI, Patrice QUINTON, Yves ROBERT, Yannick SAOUTER, Bernard TOURANCHEAU
34 Pages, Juillet 1989.

PI 487 **PLANS SIMULATION USING TEMPORAL LOGICS**
Eric RUTTEN, Lionel MARCE
40 Pages, Juillet 1989.

PI 488 **ON FINITE LOOPS IN LOGIC PROGRAMMING**
Philippe BESNARD
20 Pages, Septembre 1989.

PI 489 **LTA : UN LANGAGE DE TRAITEMENT D'ARBRES**
Dalila HATTAB
24 Pages, Septembre 1989.

PI 490 **THE SIGNAL SOFTWARE ENVIRONMENT FOR REAL-TIME SYSTEM SPECIFICATION, DESIGN, AND IMPLEMENTATION**
Albert BENVENISTE, Paul LE GUERNIC, Christian JACQUEMOT
34 Pages, Septembre 1989.

PI 491 **PHYSIQUE QUALITATIVE : PRESENTATION ET COMMENTAIRES**
Qinghua ZHANG
48 Pages, Septembre 1989.

PI 492 **SPARSE MATRIX MULTIPLICATION ON VECTOR COMPUTERS**
Jocelyne ERHEL
20 Pages, Septembre 1989.

PI 493 **THE SUPERIMPOSITION OF ESTELLE PROGRAMS : A TOOL FOR THE IMPLEMENTATION OF OBSERVATION AND CONTROL ALGORITHMS**
Benoît CAILLAUD
30 Pages, Septembre 1989.

PI 494 **EMPLOI DU TEMPS : PROBLEME MATHEMATIQUE OU PROBLEME POUR LA PROGRAMMATION EN LOGIQUE AVEC CONTRAINTES**
Xavier COUSIN
64 Pages, Septembre 1989.

PI 495 **NUMERICAL METHODS IN MARKOV CHAIN MODELING**
Bernard PHILIPPE, Youcef SAAD, William J. STEWART
46 Pages, Septembre 1989.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique