EXPERIMENTS WITH TWO-STAGE ITERATIVE SOLVERS AND PRECONDITIONED KRYLOV SUBSPACE METHODS ON NEARLY COMPLETELY DECOMPOSABLE MARKOV CHAINS

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER

ENGINEERING AND INFORMATION SCIENCE

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Wail Gueaieb

June, 1997

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Tuğrul Dayar(Principal Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Cevdet Aykanat

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Mustafa Ç. Pınar

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet Baray, Director of Institute of Engineering and Science

ABSTRACT

EXPERIMENTS WITH TWO-STAGE ITERATIVE SOLVERS AND PRECONDITIONED KRYLOV SUBSPACE METHODS ON NEARLY COMPLETELY DECOMPOSABLE MARKOV CHAINS

Wail Gueaieb
M.S. in Computer Engineering and Information Science
Supervisor: Assistant Professor Dr. Tuğrul Dayar
June, 1997

Preconditioned Krylov subspace methods are state-of-the-art iterative solvers developed mostly in the last fifteen years that may be used, among other things, to solve for the stationary distribution of Markov chains. Assuming Markov chains of interest are irreducible, the problem amounts to computing a positive solution vector to a homogeneous system of linear algebraic equations with a singular coefficient matrix under a normalization constraint. That is, the $(n \times 1)$ unknown stationary vector x in

$$Ax = 0, \ \ \|x\|_1 = 1 \tag{0.1}$$

is sought. Here $A = I - P^T$, an $n \times n$ singular M-matrix, and P is the one-step stochastic transition probability matrix.

Albeit the recent advances, practicing performance analysts still widely prefer iterative methods based on splittings when they want to compare the performance of newly devised algorithms against existing ones, or when they need candidate solvers to evaluate the performance of a system model at hand. In fact, experimental results with Krylov subspace methods on Markov chains, especially the ill-conditioned nearly completely decomposable (NCD) ones, are few. We believe there is room for research in this area specifically to help us understand the effect of the degree of coupling of NCD Markov chains and their nonzero structure on the convergence characteristics and space requirements of preconditioned Krylov subspace methods.

The work of several researchers have raised important and interesting questions that led to research in another, yet related direction. These questions are the following: "How must one go about partitioning the global coefficient matrix A in equation (0.1) into blocks if the system is NCD and a two-stage iterative solver (such as block successive overrelaxation—SOR) is to be employed? Are block partitionings dictated by the NCD normal form of P necessarily superior to others? Is it worth investing alternative partitionings? Better yet, for a fixed labelling and partitioning of the states, how does the performance of block SOR (or even that of point SOR) compare to the performance of the iterative aggregation-disaggregation (IAD) algorithm? Finally, is there any merit in using two-stage iterative solvers when preconditioned Krylov subspace methods are available?"

Experimental results show that in most of the test cases two-stage iterative solvers are superior to Krylov subspace methods with the chosen preconditioners, on NCD Markov chains. For two-stage iterative solvers, there are cases in which a straightforward partitioning of the coefficient matrix gives a faster solution than can be obtained using the NCD normal form.

Key words: Markov chains, near complete decomposability, stationary iterative methods, projection methods, block iterative methods, preconditioning, ill-conditioning.

ÖZET

İKİ SEVİYELİ DOLAYLI ÇÖZÜCÜLER VE İYİLEŞTİRİLMİŞ KRYLOV ALTUZAY YÖNTEMLERİ İLE NEREDEYSE BÖLÜNEBİLİR MARKOV ZİNCİRLERİ ÜZERİNDE DENEYLER

Wail Gueaieb
Bilgisayar ve Enformatik Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Yrd. Doç. Dr. Tuğrul Dayar
Haziran, 1997

İyileştirilmiş Krylov altuzay yöntemleri çoğunlukla son onbeş yılda geliştirilmiş, başka şeyler yanında, Markov zincirlerinin durağan dağılımlarını elde etmede kullanınan en son dolaylı çözücülerdir. İlgilenilen Markov zincirlerinin indirgenemez olduğu varsayılırsa, problem tekil bir katsayı matrisine sahip bir türdeş lineer cebirsel denklemler takımına bir normalleştirme şartı altında pozitif bir çözüm vektörü hesaplamaktan ibarettir. Yani,

$$Ax = 0, \;\; \|x\|_1 = 1 \tag{0.1}$$

deki $(n \times 1)$'lik bilinmeyen durağan vektör x aranmaktadır. Burada $A = I - P^T$ $n \times n$ tekil bir M-matrisi ve P bir-adımlık rassal geçiş olasılık matrisidir.

Son gelişmelere rağmen, mesleklerini icra eden başarım çözümleyicileri, yeni tasarlanmış algoritmaların başarımını var olanlarla kıyaslamak istediklerinde, veya eldeki bir sistem modelinin başarımını değerlendirmek için aday çözücülere gerek duyduklarında, hala çoğunlukla bölmeye dayanan dolaylı yöntemleri tercih etmektedirler. Esasında, Markov zincirleri, özellikle de hastalıklı neredeyse bölünebilir olanları üzerinde Krylov altuzay yöntemleri ile deneysel sonuçlar pek azdır. Biz bu alanda, özellikle de neredeyse bölünebilir Markov zincirlerinin bağlanma derecelerinin ve sıfırdan farklı yapılarının iyileştirilmiş Krylov altuzay yöntemlerinin yakınsama özellikleri ve yer gerekleri üzerindeki etkilerini anlamamıza yardım edecek araştırmalar için yer olduğuna inanıyoruz.

Bazı araştırmacıların çalışmaları başka fakat ilintili yönde araştırmaları neden olan önemli ve ilginç sorular ortaya çıkardı. Bu sorular şunlardır: “Eğer

sistem neredeyse bölünebilirse ve (blok ardarda üst yumuşatma—SOR gibi) iki seviyeli bir dolaylı çözücü kullanılacaksa, (0.1) denklemindeki global katsayı matrisi A'yı nasıl parçalara ayırmalı? P'nin neredeyse bölünebilir normal yapısının zorunlu kıldığı blok ayrıştırmalar diğerlerine oranla mutlaka daha mı üstündür? Alternatif ayrıştırmalara yatırım yapmaya değer mi? Hatta, durumlar sabit adlandırılıp ayrıştırıldığında blok ardarda üst yumuşatmanın (hatta nokta ardarda üst yumuşatmanın) başarımı dolaylı birleştirme-ayrıştırma algoritmasının başarımı ile nasıl kıyaslar? Son olarak, iyileştirilmiş Krylov altuzay yöntemleri varken iki seviyeli dolaylı çözücüleri kullanmanın bir değeri var mıdır?"

Deneysel sonuçlar pek çok test vakasında iki seviyeli dolaylı çözücülerin seçilmiş iyileştiriciler için Krylov altuzay yöntemlerine göre neredeyse bölünebilir Markov zincirlerinden daha üstün olduklarını göstermektedir. İki seviyeli dolaylı çözücüler için, katsayı matrisinin basit bir ayrıştırılmasının neredeyse bölünebilir yapısının kullanılarak bulunacak bir taneden daha hızlı çözüm verdiği vakalar vardır.

Anahtar kelimeler: Markov zincirleri, neredeyse bölünebilirlik, durağan dolaylı yöntemler, projeksiyon yöntemleri, blok dolaylı yöntemler, iyileştirme, hastalıklılık.

To my parents, brothers, and sister

ACKNOWLEDGMENTS

I would like to thank Dr. Tuğrul Dayar for his invaluable help, time, persistent encouragement, advice, ideas and suggestions throughout this study. He was the one to introduce me to the area, and he not only has been a role model but also more than a friend. Without his continuing support and supervision, this work would not have been possible. Special thanks are due to Dr. Aykanat and Dr. Pınar for the time they provided to read and review this thesis and for their keen acceptance to be in my committee. I also would like to extend my gratitude to the professors I have taken courses with and I have got to know during the past six years. I especially would like to acknowledge Dr. Ulusoy, Dr. Flenner, Dr. Güvenir, Dr. Kerimov and Dr. Dikovsky from whom I have learned a lot. They will greatly influence my future work.

I am grateful to the graduate administration of the Department of Computer Engineering and Information Science for the financial support provided throughout my graduate studies at Bilkent University and to the faculty and staff of the same department for creating a pleasant environment for learning and research.

I also would like to thank all my friends here for their friendship and continuous support.

Lastly, I am forever grateful to my parents, brothers and sister for their unwavering support throughout my student life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Overview

1.1 Markov Chains

1.1.1 Definitions

Understanding the behavior of physical systems is often achieved by modeling the system as a set of states which it can occupy and determining how it moves from one state to another in time. If the future evolution of the system does not depend on the past history but only on the current state, the system may be represented by a *stochastic process*. Stochastic processes arise extensively throughout queueing network analysis, computer systems performance evaluation, large scale economic modeling, biological, physical, and social sciences, engineering, and other areas.

A *stochastic process* is a family of random variables $\{X(t), t \in T\}$ defined on a given probability space and indexed by the parameter t, where t varies over some *index set (parameter space)* T [36]. T is a subset of $(-\infty, +\infty)$ and is usually thought of as the time parameter set. $X(t)$ denotes the observation at time t. If the index set is discrete, e.g., $T = \{0, 1, \ldots\}$, then we have a *discrete(-time) stochastic process*; otherwise, if T is continuous, e.g., $T = \{t : 0 \leq t \leq +\infty\}$, we call the process a *continuous (-time) stochastic process*. The values assumed by the random variable $X(t)$ are called *states*. The set of all possible states represent the *state space* $\mathcal{S}$ of the process. This state space may be discrete or continuous.

A *Markov process* is a stochastic process whose next state depends on the current state only, and not on the previous states. By this, it is said to satisfy the "Markov property." When the transitions out of state $X(t)$ depend on the time t, the Markov process is said to be *nonhomogeneous*. However, if the state transitions are independent of time, the Markov process is said to be *homogeneous*. If the state space of a Markov process is discrete, the Markov process is referred to as a *Markov chain*. Throughout this work, we concentrate on discrete-time Markov chains (DTMCs) and continuous-time Markov chains (CTMCs) with finite state space.

To satisfy the Markov property, the time spent in a state of a Markov chain must satisfy the *memoryless* property: At any time t, the remaining time the chain will spend in its current state must be independent of the time already spent in that state. This means that this time must be exponentially distributed for CTMCs and geometrically distributed for DTMCs. These are the only distributions that possess the memoryless property.

1.1.2 Discrete and Continuous Time Markov Chains

In this section, we will provide formal definitions of DTMCs and CTMCs.

For a DTMC, $X(t)$ is usually represented by $X_n \quad (n = 0, 1, \ldots)$ as we observe the system at a discrete set of times. $\{X_n, \quad n = 0, 1, \ldots\}$ is called a *stochastic sequence*. A DTMC satisfies the following relationship for all natural numbers n and all states x_n.

$$\begin{aligned} &Prob\{X_{n+1} = x_{n+1} | X_0 = x_0, X_1 = x_1, \ldots, X_n = x_n\} \\ &\quad = Prob\{X_{n+1} = x_{n+1} | X_n = x_n\}, \quad n \geq 0. \end{aligned} \tag{1.1}$$

The conditional probabilities $Prob\{X_{n+1} = x_{n+1} | X_n = x_n\}$ are called the *single-step transition probabilities*, or just the *transition probabilities*, of the Markov chain.

For a time homogeneous DTMC, transition probabilities are independent of n, and hence may be written as

$$p_{ij} = Prob\{X_{n+1} = j | X_n = i\}, \quad \forall n = 0, 1, \ldots, \text{ and } i, j \in \mathcal{S}. \tag{1.2}$$

The matrix P whose ijth element is given by p_{ij}, for all i and j, is called the *transition probability matrix*, or the *chain matrix* [36]. P is a stochastic matrix, i.e., its elements p_{ij} satisfy the following two properties

$$0 \leq p_{ij} \leq 1, \qquad \forall i, j \in \mathcal{S}, \tag{1.3}$$
$$\sum_j p_{ij} = 1, \qquad \forall i \in \mathcal{S}. \tag{1.4}$$

Let $\pi_j(n) = Prob\{X_n = j\}, \ \forall j \in \mathcal{S}$. Note that $\sum_{j \in \mathcal{S}} \pi_j(n) = 1$. Then $\pi(n) = (\pi_1(n), \pi_2(n), \ldots, \pi_i(n), \ldots)$ denotes the *state probability vector* at step n. Note that 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. The probability of being at a particular state j just after the nth transition may be expressed as

$$\pi_j(n) = \sum_{i \in \mathcal{S}} \pi_i(n-1)\, p_{ij}, \quad n \geq 1. \tag{1.5}$$

Equation (1.5) can be rewritten in matrix form as

$$\pi(n) = \pi(n-1)\, P, \quad n \geq 1. \tag{1.6}$$

Analogously, a CTMC may be described as

$$\begin{aligned} &Prob\{X(t_{n+1}) = x_{n+1} | X(t_0) = x_0, X(t_1) = x_1, \ldots, X(t_n) = x_n\} \\ &\qquad = Prob\{X(t_{n+1}) = x_{n+1} | X(t_n) = x_n\}, \quad n \geq 0. \end{aligned} \tag{1.7}$$

For the transition probabilities, we write

$$p_{ij}(s,t) = Prob\{X(t) = j | X(s) = i\}, \qquad t, s \geq 0, \quad i, j \in \mathcal{S}. \tag{1.8}$$

When the CTMC is homogeneous—and that is what we are interested in—these transition probabilities depend only on the difference $\Delta t = t - s$, and not on s and t. So we can rewrite equation (1.8) as

$$p_{ij}(\Delta t) = Prob\{X(s + \Delta t) = j | X(s) = i\}, \qquad s, \Delta t \geq 0, \quad i, j \in \mathcal{S}.$$

For a better understanding of the relationship between CTMCs and DTMCs, consider the time axis as a sequence of mutually disjoint sufficiently small intervals so that there is at most one transition in each subinterval. At the end of each Δt time interval, there is exactly one transition, and hence the system behaves like a DTMC.

Let $\pi_j(t+\Delta t) = Prob\{X(t+\Delta t) = j\}, \quad \forall j \in \mathcal{S}$. Note that $\sum_{j\in\mathcal{S}} \pi_j(t+\Delta t) = 1$. Then $\pi(t+\Delta t)$ denotes the state probability vector at time $t+\Delta t$.

We determine the probability of being in state j at time $t+\Delta t$ by

$$\pi_j(t+\Delta t) = \sum_{i\in\mathcal{S}} \pi_i(t)\, p_{ij}(\Delta t), \qquad t, \Delta t \geq 0, \quad \forall j \in \mathcal{S}.$$

This can be written in matrix form as

$$\pi(t+\Delta t) = \pi(t)\, P(\Delta t), \qquad t, \Delta t \geq 0.$$

Here $P(\Delta t)$ is the *one-step transition probability matrix for the interval* Δt whose ijth entry is given by $p_{ij}(\Delta t)$.

Let $q_{ij}(t)$ be the *rate* at which transitions occur from state i to state j at time t. The transition rate is an *instantaneous* quantity that denotes the number of transitions that occur per unit time.

$$q_{ij}(t) = \lim_{\Delta t \to 0} \left\{ \frac{p_{ij}(t+\Delta t)}{\Delta t} \right\} \qquad \text{for } i \neq j.$$

This leads to the following transition probability

$$p_{ij}(t+\Delta t) = q_{ij}(t)\Delta t + o(\Delta t), \qquad \text{for } i \neq j,$$

where $o(\Delta t)$ is the "little oh" notation such that $o(\Delta t)$ tends to 0 faster than Δt.

$$\lim_{\Delta t \to 0} \frac{o(\Delta t)}{\Delta t} = 0.$$

Starting from the concept of probability conservation, we can write

$$\begin{aligned} 1 - p_{ii}(t, t+\Delta t) &= \sum_{j\neq i} p_{ij}(t, t+\Delta t) \\ &= \sum_{j\neq i} [q_{ij}(t)\Delta t + o(\Delta t)] \end{aligned}$$

Dividing by Δt and taking the limit as $\Delta t \rightarrow 0$, we obtain

$$q_{ii}(t) \equiv \lim_{\Delta t \rightarrow 0} \left\{ \frac{p_{ii}(t, t+\Delta t) - 1}{\Delta t} \right\} = \lim_{\Delta t \rightarrow 0} \left\{ \frac{-\sum_{j \neq i} [q_{ij}(t)\Delta t + o(\Delta t)]}{\Delta t} \right\}.$$

Hence,

$$q_{ii}(t) = -\sum_{j \neq i} q_{ij}(t). \tag{1.9}$$

The matrix $Q(t)$ whose ijth element is $q_{ij}(t)$ is called the *infinitesimal generator matrix*, or *transition rate matrix*, for the CTMC. In matrix form, it is

$$Q(t) = \lim_{t \rightarrow 0} \left\{ \frac{P(t, t+\Delta t) - I}{\Delta t} \right\},$$

where $P(t, t+\Delta t)$ is the transition probability matrix whose ijth element is $p_{ij}(t, t+\Delta t)$ and I is the identity matrix. When the CTMC is homogeneous, the transition rates q_{ij} are independent of time, and the transition matrix is simply written as Q.

The matrix Q has row sums of 0 as each of its diagonal elements is the negated sum of the corresponding off-diagonal elements (of that row). From equation (1.9) we get

$$q_{ii} = -\sum_{j \neq i} q_{ij}. \tag{1.10}$$

1.1.3 Probability Distributions

Determining the stationary distribution of a Markov chain is the core of this study. As defined in the previous section, for a DTMC, $\pi_i(n)$ denotes the probability that a Markov chain is in state i at step n, i.e.,

$$\pi_i(n) = Prob\{X_n = i\}, \quad n \geq 0, \quad i \in \mathcal{S}.$$

Definition 1.1 (Stationary distribution) *[36] Let P be the transition probability matrix of a DTMC, and let the vector z whose elements z_j denote the probability of being in state j be a probability distribution; i.e.,*

$$z_j \in \mathbb{R}, \quad 0 \leq z_j \leq 1, \quad \textit{and} \quad \sum_j z_j = 1.$$

Then z is said to be a stationary distribution if and only if $zP = z$.

Definition 1.2 (Limiting distribution) *[36] Given an initial probability distribution $\pi(0)$, if the limit*

$$\lim_{n\to\infty} \pi(n),$$

exists, then the limit is called the limiting distribution, and we write

$$\pi = \lim_{n\to\infty} \pi(n).$$

The limiting distribution is also known as the *steady-state*, or *equilibrium*, distribution. Informally speaking, the steady-state distribution is the probability distribution which, if it exists, the process will reach after sufficiently many transitions, independently of the initial probability distribution, and will remain in that distribution for all further transition steps. We will come back to this in later sections.

Now, taking the limit as $n \to \infty$ of both sides of equation (1.6), we obtain

$$\pi = \pi P, \qquad \sum_{i\in\mathcal{S}} \pi_i = \|\pi\|_1 = 1. \tag{1.11}$$

Analogously to DTMCs, a stationary probability vector for a CTMC can be defined as any vector z such that $zQ = 0, \ \ \|z\|_1 = 1$. The steady-state distribution of a CTMC, if it exists, is written as

$$\pi = \lim_{t\to\infty} \pi(t)$$

and is independent of the initial probability distribution $\pi(0)$. Recall that $\pi_i(t)$ is the probability that a CTMC is in state i at time t, i.e.,

$$\pi_i(t) = Prob\{X(t) = i\}, \quad \forall i \in \mathcal{S}.$$

Then,

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

Since $q_{ii}(t) = -\sum_{j\neq i} q_{ij}(t)$, we have

$$\pi_i(t+\Delta t) = \pi_i(t) + \left(\sum_{k} q_{ki}(t)\pi_k(t)\right)\Delta t + o(\Delta t),$$

and

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

i.e.,

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

In matrix notation, this gives

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

When the Markov chain is homogeneous, we may drop the time parameter t from the transition matrix Q and simply write

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

If the limiting distribution π exists, then after sufficiently long time t, $\pi(t)$ will converge to π and $d\pi(t)/dt$ will be equal to 0. Hence, for a homogeneous CTMC,

$$\pi Q = 0, \qquad \sum_{i \in \mathcal{S}} \pi_i = \|\pi\|_1 = 1. \tag{1.12}$$

1.1.4 Numerical Properties

As we discussed in section (1.1.2), the transition matrix P is a stochastic matrix (see equations (1.3) and (1.4)). Besides, P is singular, and its order is equal to the cardinality of $\mathcal{S}$. Note that equation (1.11) is an eigensystem in which the unit left-hand eigenvector of P, corresponding to the unit eigenvalue (= 1), is sought.

To proceed further, we need to introduce the definitions of the spectrum and the spectral radius of a matrix.

The *spectrum* of a matrix A is the set of all eigenvalues of A, and it is denoted by $\sigma(A)$. In mathematical notation,

$$\sigma(A) = \{\lambda | A\vartheta = \lambda\vartheta, \quad \vartheta \neq 0\}.$$

It is worth mentioning that the spectrum of a matrix A is equal to the spectrum of the transpose of A. In other words, $\sigma(A) = \sigma(A^T)$.

The *spectral radius* of a matrix A is the largest eigenvalue of A, in magnitude, and it is denoted by $\rho(A)$. In mathematical notation,

$$\rho(A) = \max\{|\lambda|, \lambda \in \sigma(A)\}.$$

One way of solving equation (1.11) is to transform it to a homogeneous linear system

$$\pi(P - I) = 0, \qquad \|\pi\|_1 = 1,$$

where π is the unknown vector.

Unlike P, the infinitesimal generator matrix Q has row sums of 0 (see equation (1.10)). Equation (1.12) represents a homogeneous system of linear equations with Q as the coefficient matrix and π as the unknown vector.

Definition 1.3 (M-matrix) *[10] An M-matrix A is any finite square matrix with nonpositive off-diagonal elements and nonnegative diagonal elements which may be written in the form*

$$A = rI - G, \qquad G \geq 0, \quad r \geq \rho(G),$$

where r is a real scalar ($r \in \mathbb{R}$), G is a square matrix, and I is the identity matrix.

The matrix $-Q$ has nonpositive off-diagonal elements. Let's illustrate how $-Q$ can be written in the form $-Q = rI - G$. Let $G = rI + Q$, where $r = \max_{i \in \mathcal{S}} |q_{ii}| > 0$. The matrix $(1/r)G$ is a stochastic matrix, hence $\rho(G) \leq r$. So all the conditions are satisfied and $-Q$ is verified to be an M-matrix.

For determining the stationary distribution vector of a Markov chain, a DTMC formulation may be transformed to a CTMC formulation and vice versa.

$$DTMC \Longleftrightarrow CTMC$$

$DTMC \Longrightarrow CTMC: \quad \pi = \pi P \Longrightarrow 0 = \pi Q$, where $Q = P - I$,
$CTMC \Longrightarrow DTMC: \quad 0 = \pi Q \Longrightarrow \pi = \pi P$, where $P = (1/\mu)Q + I$ and $\mu = \max_{i \in \mathcal{S}}\{|q_{ii}|\}$.

1.2 State Classification

In order to be able to classify the states of a Markov chain, we need to introduce some new definitions. Without loss of generality, the following definitions will be valid for DTMCs only. However, it is easy to figure out the corresponding homogeneous CTMC definitions since we remarked the relationship between the two types.

Let $p_{ij}^{(n)}$ be the probability of going from state i to state j in n steps. Then

$$p_{ij}^{(n)} = Prob\{X_{m+n} = j \mid X_m = i\}, \qquad m, n = 0, 1, 2 \ldots, \quad i, j \in \mathcal{S}.$$

We define

$$p_{ij}^{(0)} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases} .$$

Now we are ready to introduce the *Chapman-Kolmogorov* equation for Markov chains:

$$p_{ij}^{(n)} = \sum_{k \in \mathcal{S}} p_{ik}^{(l)} p_{kj}^{(n-l)} \qquad \text{for } 0 < l < n.$$

In matrix notation, it may be written as

$$P^{(n)} = P^{(l)} P^{(n-l)},$$

where $P^{(n)}$ is the matrix of n-step transition probabilities with entries $p_{ij}^{(n)}$ for all $i, j \in \mathcal{S}$. $P^{(n)}$ is obtained by raising P to the nth power. In other words, $P^{(n)} = P^n$.

If the steady-state distribution π of a DTMC exists, then

$$\lim_{n \to \infty} P^{(n)} = \lim_{n \to \infty} P^n = \begin{pmatrix} \pi \\ \pi \\ \vdots \\ \pi \end{pmatrix},$$

and hence

$$\lim_{n\to\infty} p_{ij}^{(n)} = \pi_j, \qquad \forall i, j \in \mathcal{S}.$$

1.2.1 Definitions

Classifying states of Markov chains requires some definitions to be made, and we mainly follow [10] in that respect.

- ▷ State j is said to be *accessible* from state i if $\exists n \geq 0$ for which $p_{ij}^{(n)} > 0$.

  ▷ two states i and j are said to *communicate* if state i is accessible from state j and state j is accessible from state i.

  ▷ A nonempty set $\mathcal{C} \subset \mathcal{S}$ is said to be *closed* if and only if $\forall i \in \mathcal{C}$ and $\forall j \notin \mathcal{C}$, j is not accessible from i.

  ▷ A Markov chain is said to be *irreducible* if all states communicate with each other.

- ▷ A state is said to be *recurrent* if the process, once in that state, returns to that state with probability 1. If the probability of returning to that state is strictly less than 1, then the state is said to be *transient*. Mathematically speaking, we can write

  $$\sum_{n=1}^{\infty} p_{ii}^{(n)} = \infty \Longleftrightarrow i \text{ is recurrent,}$$
  $$\sum_{n=1}^{\infty} p_{ii}^{(n)} < \infty \Longleftrightarrow i \text{ is transient.}$$

  ▷ A recurrent state i for which $p_{ii} = 1$ is said to be an *absorbing* state.

  ▷ If the mean time to return to a state is finite, the state is said to be *positive recurrent*, or *recurrent nonnull*. Otherwise, if the mean time to return to a state is infinite, given that the state is recurrent, then the state is said to be *null recurrent*.

  ▷ If all states of a Markov chain are positive recurrent, null recurrent, or transient, then we respectively have a positive recurrent, null recurrent, or transient Markov chain.

- ▷ The period of state i, written $d(i)$, is the greatest common divisor of all integers $n \geq 1$ for which $p_{ii}^{(n)} > 0$. If $p_{ii}^{(n)} = 0$ for all $n \geq 1$, then we define $d(i) = 0$.

  ▷ A Markov chain in which each state has period 1 is said to be *aperiodic*, or *acyclic*; whereas, a Markov chain in which each state has the same periodicity and this period is greater than 1, is said to be *periodic*, or *cyclic*.

- ▷ A positive recurrent, aperiodic state is said to be *ergodic*.

  ▷ If a Markov chain is irreducible, positive recurrent, and aperiodic, then it is said to be an ergodic Markov chain.

- ▷ A Markov chain with a finite state space $\mathcal{S}$ is said to be *regular* if

$$\lim_{n \to \infty} P^n > 0.$$

  Hence, a steady-state distribution exists for a regular Markov chain.

  ▷ A Markov chain with a finite state space is said to be *doubly stochastic* if

$$\sum_{k \in \mathcal{S}} p_{ik} = \sum_{k \in \mathcal{S}} p_{kj} = 1, \qquad \forall i, j \in \mathcal{S}.$$

1.2.2 Following Properties

In this section we will derive some important properties as a consequence of the definitions provided in the previous section.

- If two states communicate, then they are of the same type. That is, they are both either positive recurrent, null recurrent, or transient.
- States that communicate have the same periodicity.
- If the state space $\mathcal{S}$ is finite, then at least one of the states is positive recurrent.
- If the state space $\mathcal{S}$ is finite and the Markov chain is irreducible, then every state in $\mathcal{S}$ is positive recurrent.

- No state of a Markov chain with a finite state space can be null recurrent.
- If π is the stationary distribution for a Markov chain, then $\pi_j = 0$ if state j is transient or null recurrent.
- If a Markov chain does not have any positive recurrent states, then the Markov chain does not have any stationary distribution. Besides, the state space of such a chain has to be infinite.
- Suppose π and π' are two different stationary distribution vectors for a Markov chain. Then there exists infinitely many stationary distribution vectors for the chain. In essence, any convex combination of π and π' is also a stationary distribution for the Markov chain.
- Let $\mathcal{C}$ be any irreducible closed set of positive recurrent states in a Markov chain. Then there exists a unique stationary distribution π for the chain that is concentrated on $\mathcal{C}$. Stationary distribution probabilities for states outside $\mathcal{C}$ are all 0.
- Let $\mathcal{S} = \mathcal{S}_t \cup \mathcal{S}_{nr} \cup \mathcal{S}_{pr}$, be the state space of a Markov chain, where $\mathcal{S}_t$ is the set of transient states, $\mathcal{S}_{nr}$ is the set of null recurrent states, and $\mathcal{S}_{pr}$ is the set of positive recurrent states. The following is a summary regarding the stationary distributions of the chain:
  - ▹ If $\mathcal{S}_{pr} = \emptyset$, then there is no stationary distribution.
  - ▹ If $\mathcal{S}_{pr} \neq \emptyset$, then there is at least one stationary distribution.
  - ▹ If $\mathcal{S}_{pr} \neq \emptyset$, and $\mathcal{S}_{pr}$ is irreducible and closed, then there is a unique stationary distribution concentrated on $\mathcal{S}_{pr}$.
  - ▹ If $\mathcal{S}_{pr} = \cup_i \mathcal{C}_i$ and $\mathcal{C}_i \cap \mathcal{C}_j = \emptyset$ for all i, j, then there is a unique stationary distribution vector π_i concentrated on $\mathcal{C}_i$ for all i. Furthermore,

$$\pi = \sum_i \alpha_i \pi_i,$$

    where $\alpha_i \geq 0$ for all i and $\sum_i \alpha_i = 1$, is also a stationary distribution vector.
- Any irreducible, positive recurrent Markov chain has a unique stationary distribution.

- If π is a steady-state distribution for a Markov chain, then π is the only stationary distribution.
- A regular Markov chain is irreducible, positive recurrent, and aperiodic.
- A steady-state distribution exists for an ergodic Markov chain.

One of the most important theorems in the domain of Markov chains is the Perron-Frobenius theorem. This theorem is very helpful because of its strong application to stochastic matrices (see [36, 17]).

Theorem 1.4 (Perron-Frobenius) *[36] Let $A \geq 0$ be an irreducible square matrix of order n. Then,*

1. - *A has a positive real eigenvalue, λ_1, such that $\lambda_1 = \rho(A)$.*
   - *To $\rho(A)$ there corresponds an eigenvector $x > 0$, i.e.,*

     $$Ax = \lambda_1 x \quad \textit{and} \quad x > 0.$$

   - *$\rho(A)$ increases when any entry of A increases.*
   - *$\rho(A)$ is a simple eigenvalue of A, i.e., λ_1 is a simple root of*

     $$det(\lambda I - A) = 0.$$

2. *Let S be a matrix of complexed-valued elements and S^* obtained from S by replacing each element by its modulus. If $S^* \leq A$, then any eigenvalue μ of S satisfies*

   $$|\mu| \leq \lambda_1.$$

   Furthermore, if for some μ, $|\mu| = \lambda_1$, then $S^* = A$. More precisely, if $\mu = \lambda_1 e^{i\theta}$, then

   $$S = e^{i\theta} DAD^{-1},$$

   where $D^* = I$.

3. *if A has exactly p eigenvalues equal in modulus to λ_1, then these numbers are all different and are the roots of the equation*

$$\lambda^p - \lambda_1^p = 0.$$

When plotted as points in the complex plane, this set of eigenvalues is invariant under a rotation of the plane through the angle $2\pi/p$ but not through smaller angles. When $p > 1$ then A can be symmetrically permuted to the following cyclic form

$$A = \begin{pmatrix} 0 & A_{12} & 0 & \cdots & 0 \\ 0 & 0 & A_{23} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & A_{p-1,p} \\ A_{p1} & 0 & 0 & \cdots & 0 \end{pmatrix},$$

in which the diagonal submatrices A_{ii} are square and the only nonzero submatrices are $A_{12}, A_{23}, \ldots, A_{p-1,p}, A_{p1}$.

1.3 Decomposable Probability Matrices

A special case of particular interest in Markov chains is when the chain is reducible. In such a case, the probability matrix may be transformed to a particular nonzero structure and is said to be *decomposable* [36].

Definition 1.5 (Decomposable Matrix) *A square matrix A is said to be decomposable if it can be brought by symmetric permutations of its rows and columns to the form*

$$A = \begin{pmatrix} U & 0 \\ W & V \end{pmatrix}, \tag{1.13}$$

where U and V are square nonzero matrices and W is, in general, rectangular.

If a Markov chain is reducible then there exists at least one ordering of the state space such that the probability matrix is in the form of (1.13). If U and

V are of orders n_1 and n_2, respectively, where $n(= n_1+n_2)$ is the total number of states, then the state space may be decomposed to two disjoint sets

$$\begin{aligned} \mathcal{B}_1 &= \{s_1, s_2, \ldots, s_{n_1}\}, \quad \text{and} \\ \mathcal{B}_2 &= \{s_{n_1+1}, \ldots, s_n\}, \end{aligned}$$

where s_i's $(i = 1, 2, \ldots, n)$ are the states of the Markov chain. For the time being, let's suppose that $W \neq 0$, i.e., there exists at least one nonzero entry in W. Observing the nonzero structure in (1.13), we can see that once the process is in one of the states of $\mathcal{B}_1$, it can never pass to a state in $\mathcal{B}_2$. Consequently, $\mathcal{B}_1$ is known to be an *isolated*, or an *essential* set. On the other hand, being in one of the states of the set $\mathcal{B}_2$ guarantees staying in that set until the first transition to one of the states of $\mathcal{B}_1$. Then set $\mathcal{B}_2$ is said to be transient, or *nonessential*. In the particular case where $W = 0$, the matrix is said to be *completely decomposable*. In this case, both $\mathcal{B}_1$ and $\mathcal{B}_2$ are isolated.

The matrix U may be decomposable, and hence can be permuted to the form in (1.13). If we continue in this pattern, the matrix A may be brought to a special form called the *normal form* of a decomposable nonnegative matrix, given by

$$A = \begin{pmatrix} A_{11} & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & A_{22} & 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & A_{kk} & 0 & \cdots & 0 \\ A_{k+1,1} & A_{k+1,2} & \cdots & \cdots & A_{k+1,k} & A_{k+1,k+1} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ A_{m,1} & A_{m,2} & \cdots & \cdots & A_{m,k} & A_{m,k+1} & \cdots & A_{m,m} \end{pmatrix}. \quad (1.14)$$

The diagonal blocks A_{ii} $(i = 1, \ldots, m)$ are square nondecomposable matrices. All the blocks to the left of the first k diagonal blocks (i.e., A_{ii} for $i = 1, 2, \ldots, k$), and to the right of all diagonal blocks, are 0. For the submatrices to the left of the last $m-k$ diagonal blocks (i.e., A_{ij} for $i = k+1, \ldots, m, \quad j = 1, \ldots, i-1$), there exits at least one nonzero submatrix per row of blocks. Therefore, the diagonal blocks possess the following characteristics:

- $A_{ii}, \quad i = 1, \ldots, k:$ isolated and nondecomposable,

- A_{ii}, $i = k+1, \ldots, m$: transient, and again, nondecomposable.

This is quite logical and easy to figure out bearing in mind the nonzero structure of (1.14).

A probability matrix having the form in (1.14) according to the Perron-Frobenius theorem has a unique eigenvalue of multiplicity k. Besides, there exist k linearly independent left-hand eigenvectors corresponding to this unit eigenvalue. The last $m - k$ entries of all these eigenvectors are all 0's as they correspond to transient states. Further details and proofs may be found in [36].

1.4 NCD Markov Chains

Consider a probability matrix having the form in (1.14). If we introduce "small" perturbations on some of the zero off-diagonal blocks to make them nonzeros (still conserving its stochastic properties (1.3) and (1.4)), the matrix is no longer decomposable and the Markov chain becomes irreducible. However, since the perturbations are small, meaning that the introduced nonzero entries have small values compared to those within the diagonal blocks, the matrix is said to be *nearly decomposable*. If, now, the nonzero elements in all the off-diagonal submatrices are small in magnitude compared with those in the diagonal blocks, then the matrix is said to be *nearly completely decomposable* (NCD). In NCD Markov chains, the interactions between the blocks is weak, whereas interactions among the states of each block are strong.

For an example, consider the following simple completely decomposable probability matrix:

$$P = \begin{pmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{pmatrix}.$$

If we introduce small perturbations to the elements of P giving it the form

$$P = \begin{pmatrix} 1.0 - \epsilon_1 & \epsilon_1 \\ \epsilon_2 & 1.0 - \epsilon_2 \end{pmatrix},$$

such that $1.0 - \epsilon_1 \gg \epsilon_1$ and $1.0 - \epsilon_2 \gg \epsilon_2$, then the system becomes nearly completely decomposable.

Interestingly, NCD Markov chains arise frequently in many applications. It was noticed that a small perturbation in the matrix leads to a larger perturbation in the stationary distribution causing the computation of the stationary vector to be usually not as accurate as it is for ordinary irreducible Markov chains. Hence, NCD Markov chains are known to be ill-conditioned. Throughout this work, we are mainly concerned with NCD systems. We will come back to this in later chapters where we present different iterative solution techniques and compare and contrast their competitivity in computing the stationary probability vector of finite NCD Markov chains.

Chapter 2

Numerical Solution Methods

Many advanced scientific problems are practically impossible to solve analytically. As an alternative, numerical methods were introduced and they showed to be very efficient in solving a wide range of problems. We are interested in using numerical techniques to compute the stationary distribution vector of a finite irreducible Markov chain [23, 18, 9, 27, 21]. Our aim is to solve the homogeneous system of linear algebraic equations

$$Ax = 0, \ \|x\|_1 = 1, \tag{2.1}$$

where A is a $(n \times n)$ singular, irreducible M-matrix [7] and x is the unknown $(n \times 1)$ positive vector to be determined. Since our problems stem from Markov chain applications, the coefficient matrix A is taken as $A = I - P^T$ in case the one-step transition probability matrix is provided, or as $A = -Q^T$ if we are supplied with the infinitesimal generator matrix. The solution vector x corresponds to π^T (the transpose of the stationary distribution vector of the Markov chain). This explains the normalization constraint in equation (2.1) which also guarantees the uniqueness of the solution. The Perron-Frobenious theorem guarantees the existence of the solution since A is an M-matrix [7]. In the rest of this chapter we discuss the numerical techniques used to solve equation (2.1).

2.1 Direct Methods

Numerical methods that compute solutions of mathematical problems in a fixed number of floating-point operations are known as *direct methods*. The classical *Gaussian elimination* (GE) is a typical example of direct methods and it is suitable for irreducible Markov chains [36]. For a full $(n \times n)$ system of linear equations, the total number of operations required by GE is $O(n^3)$. The space complexity is $O(n^2)$. As it can be seen, these complexities grow rapidly with the problem size making GE (and direct methods in general) not suitable for large sparse matrices. Another problem with direct-solving methods is that the elimination of nonzero elements of the matrix during the reduction phase often results in the creation of several nonzero elements in positions that previously contained zero. This is called *fill-in*, and in addition to making the organization of a compact storage scheme more difficult (since provision must be made for the deletion and insertion of elements), the amount of fill-in can often be so extensive that available memory is quickly exhausted. Moreover, altering the form of the matrix may cause buildup of rounding errors [10].

It is known that if the coefficient matrix A is irreducible, there exist [24] lower and upper triangular matrices L and U such that

$$A = LU.$$

This LU decomposition is not unique. It is called the *Doolittle* decomposition if the diagonal elements of L are set to 1, and the *Crout* decomposition if the diagonal elements of U are set to 1. Usually, Gaussian elimination refers to the Doolittle decomposition.

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 linear system. For example, suppose that we are required to solve $Ax = b$ where A is nonsingular, $b \neq 0$, and the decomposition of $A = LU$ is available so that $LUx = b$. The idea is to set $Ux = z$, then the vector z may be obtained by forward substitution on $Lz = b$. Note that both L and b are known. The solution x may subsequently be obtained from $Ux = z$ by backward substitution since by this time both U and z are known.

However, for homogeneous system of equations (i.e., $b = 0$) with a singular coefficient matrix, the last row of U (supposing that the Doolittle decomposition is performed) is equal to zero. Proceeding as indicated above for the nonhomogeneous case, we get

$$Ax = LUx = 0.$$

If now we set $Ux = z$ and attempt to solve $Lz = 0$, we end-up finding that $z = 0$ since L is nonsingular ($\det(L) = 1$). Proceeding to the backward substitution on $Ux = z = 0$ when $u_{nn} = 0$, we find that it is evident that x_n may assume any nonzero value, say $x_n = \eta$. Hence, the remaining elements of x can be determined in terms of η. The solution vector is then normalized if required. Note that for homogeneous linear systems the elimination is only needed to be carried out for the first $n-1$ steps.

In the Doolittle decomposition L^{-1} exists and is called the *multiplier matrix*. L^{-1} is lower triangular and its ith column is composed of the multipliers that reduce the ith column below the main diagonal of A to zero to form the matrix U [36]. This phase is called the *reduction phase*.

Assume that U and L overwrite the upper triangular (including the diagonal) and the strictly lower triangular (excluding the diagonal) parts of A. Let $A^{(k)}$ represent the altered coefficient matrix at the kth step of the forward elimination. Then

$$a_{ij}^{(k)} = \begin{cases} a_{ij}^{(k-1)}, & i \le k,\ \forall j \\ a_{ij}^{(k-1)} + \mu_{ik}\, a_{kj}^{(k-1)}, & i > k,\ \forall j \end{cases},$$

where the *multipliers* are given by

$$\mu_{ik} = -a_{ik}^{(k-1)} / a_{kk}^{(k-1)}.$$

The elements $a_{kk}^{(k)}$ are called the *pivots* and must be nonzero if the reduction is to terminate satisfactorily. For purposes of stability, it is generally necessary to interchange the rows of the matrix so that the pivotal element is the largest in modulus in its column in the unreduced portion of the matrix (called *partial pivoting*). This ensures that the absolute values of the multipliers do not exceed 1. For some cases, it is necessary to interchange both rows and columns so that the pivotal element is the largest among all elements in the unreduced part of the matrix (*full pivoting*). However, for irreducible Markov chains no pivoting is necessary.

2.2 Iterative Methods

The term *iterative methods* refers to a wide range of techniques that use successive approximations to obtain more accurate solutions to a linear system at each step. Iterative methods of one type or another are the most commonly used methods for obtaining the stationary probability from either the stochastic transition probability matrix or from the infinitesimal generator. This choice is due to several reasons. First, in iterative methods, the only operations in which the matrices are involved are multiplications with one or more vectors. These operations conserve the form of the matrix. This may lead to considerable savings in memory required to solve the system especially when dealing with large sparse matrices. Besides, an iterative process may be terminated once a prespecified tolerance criterion has been satisfied, and this may be relatively lax. For instance, it may be wasteful to compute the solution of a mathematical model correct to full machine precision when the model itself contains error. However, a direct method is obligated to continue until the final operation has been carried out.

In this chapter we discuss three types of iterative methods: stationary iterative methods, block iterative methods, and projection methods. Throughout our work we experimented with the (point) *successive overrelaxation* (SOR) method [36, 4] as a stationary technique. Two types of block iterative methods were considered: *block SOR* and *iterative aggregation-disaggregation* (IAD) [22, 33, 12]. As for projection techniques, we chose to implement and experiment with the methods of *Generalized Minimum Residual* (GMRES) [25, 30, 31], *Biconjugate Gradient* (BCG) [15, 4], *Conjugate Gradient Squared* (CGS) [32, 37], *Biconjugate Gradient Stabilized* (BCGStab) [37], and *Quasi-Minimal Residual* (QMR) [13, 14]. SOR and the two block iterative methods we used are part of the Markov Chain Analyzer (MARCA) [35] software package version 3.0.

2.2.1 SOR: A Stationary Iterative Method

Stationary iterative methods are iterative methods that can be expressed in the simple form [4]

$$x^{(k+1)} = T\,x^{(k)} + c, \qquad k = 0, 1, \ldots \tag{2.2}$$

where neither T nor c depend upon the iteration count k. Equation (2.1) can be written in the form above by splitting the coefficient matrix A. Given a splitting

$$A = M - N$$

with nonsingular M, we have

$$(M - N)\,x = 0,$$

or

$$Mx = Nx,$$

which leads to the iterative procedure

$$x^{(k+1)} = M^{-1}N\,x^{(k)} = T\,x^{(k)}, \qquad k = 0, 1, \ldots, \tag{2.3}$$

where $x^{(0)}$ is the initial guess. Note that in our case the vector c appearing in equation (2.2) is just the zero vector. The matrix $T = M^{-1}N$ is called the *iteration matrix*.

For convergence of equation (2.3) it is required that $\lim_{k\to\infty} T^k$ exists (since $x^{(k)} = T^k\,x^{(0)}$). A necessary, but not sufficient, condition for this to be satisfied is that all the eigenvalues of T must be less than or equal to 1 in modulus, i.e., $\rho(T) \le 1$, where $\rho(T)$ is the spectral radius of T. When $\rho(T) = 1$, the unit eigenvalue of T must be the only eigenvalue with modulus 1 for convergence.

In order to have a better understanding of the convergence properties[1] of stationary iterative methods, we adopt the following definitions and theorems (from [36], pp. 169–173).

Definition 2.1 (Semiconvergent Matrices) *A matrix T is said to be semi-convergent whenever* $\lim_{k\to\infty} T^k$ *exists. This limit need not be zero.*

[1] Readers interested in learning more about the convergence behavior of stationary iterative methods are advised to see [36].

Definition 2.2 (Regular and Weak Regular Splittings) *A splitting $A = M - N$ is called a regular splitting, if $M^{-1} \geq 0$ and $N \geq 0$. It is called a weak regular splitting if $M^{-1} \geq 0$ and $M^{-1}N \geq 0$.*

Definition 2.3 (Convergent Iterative Methods) *An iterative method is said to converge to the solution of a given linear system if the iteration*

$$x^{(k+1)} = T\, x^{(k)} + c$$

associated with that method converges to the solution for every starting vector $x^{(0)}$.

Let $\gamma(A)$ denotes the maximum magnitude over all elements in $\sigma(A)\backslash\{1\}$, where $\sigma(A)$ stands for the set of eigenvalues of A, i.e.,

$$\gamma(A) = max\{|\lambda|,\ \lambda \in \sigma(A),\ \lambda \neq 1\}.$$

Note that $\gamma(A) = \rho(A)$ *iff* $1 \notin \sigma(A)$.

Theorem 2.4 *T is semiconvergent iff all of the following conditions hold*

1. *$\rho(T) \leq 1$.*
2. *If $\rho(T) = 1$, then all the elementary divisors associated with the unit eigenvalue of T are linear.*
3. *If $\rho(T) = 1$, then $\lambda \in \sigma(T)$ with $|\lambda| = 1$ implies that $\lambda = 1$.*

In general, stationary iterative methods differ in the way the coefficient matrix is split. This splitting uniquely defines the iteration matrix and hence determines the convergence rate of the method. For the SOR method with *relaxation parameter* ω, the splitting is

$$A = (\frac{1}{\omega}D - L) - (\frac{1-\omega}{\omega}D + U), \tag{2.4}$$

where $D, -L, -U$ [2] represent respectively diagonal, strictly lower triangular, strictly upper triangular parts of A. The method is said to be one of *overrelaxation* if $\omega > 1$, and one of *underrelaxation* if $\omega < 1$. For $\omega = 1$, the method

[2] L and U should not be confused with the lower and the upper triangular matrices of the LU decomposition.

reduces to another stationary iterative method called *Gauss-Seidel* (discussed in [11]). In our case, it is clear that $(\frac{1}{\omega}D - L)$ is nonsingular. Since A is an M-matrix, $(\frac{1}{\omega}D - L)^{-1} \geq 0$ regardless of the value of ω. However, $(\frac{1-\omega}{\omega}D + U) \geq 0$ is true only for $0 < \omega < 1$ which makes (2.4) a regular splitting. The iteration matrix for (*forward*) SOR is then given by

$$T_{SOR} = (\frac{1}{\omega}D - L)^{-1}(\frac{1-\omega}{\omega}D + U).$$

The iteration may be expressed as

$$x_i^{(k+1)} = (1-\omega)x_i^{(k)} + \omega \left\{ \frac{1}{d_{ii}} \left(\sum_{j=1}^{i-1} l_{ij} x_j^{(k+1)} + \sum_{j=i+1}^{n} u_{ij} x_j^{(k)} \right) \right\}, \quad i = 1, 2, \ldots, n,$$

or in matrix form as

$$x^{(k+1)} = (1-\omega)x^{(k)} + \omega \left\{ D^{-1}(Lx^{(k+1)} + Ux^{(k)}) \right\}. \tag{2.5}$$

A *backward* SOR relaxation may also be obtained by rewriting equation (2.5) as

$$x^{(k+1)} = (D - \omega L)^{-1}[(1-\omega)D + \omega U]x^{(k)}. \tag{2.6}$$

The pseudocode for the SOR algorithm is given below. The algorithm below is for any linear system of the form $Ax = b$.

It can be verified for equation (2.6) that the solution vector x (which is the transpose of the stationary probability vector) is the eigenvector corresponding to a unit eigenvalue of the SOR iteration matrix. It is worth stressing that for the SOR method, it is not necessarily true that the unit eigenvalue is the dominant eigenvalue, because the magnitude of the subdominant eigenvalue depends on the choice of the relaxation parameter ω (see [36] p. 131).

The SOR method converges only if $0 < \omega < 2$. The optimal value of ω is that which maximizes the difference between the unit eigenvalue and the subdominant eigenvalue of T_{SOR}. Therefore, the convergence rate of SOR is highly dependent on ω. In general, it is not possible to compute in advance the optimal value of ω. Even when this is possible, the cost of such computation is usually prohibitive.

Table 2.1 shows a summary of the operations per iteration and the storage requirement for the SOR method. Only the space required to store the matrices

Algorithm: SOR

1. Choose an initial guess $x^{(0)}$ to the solution x.
2. for $k = 1, 2, \dots$
3.   for $i = 1, 2, \dots, n$
4. $\sigma = 0$
5.   for $j = 1, 2, \dots, i-1$
6. $\sigma = \sigma + a_{ij} x_j^{(k)}$
7.   end
8.   for $j = i+1, \dots, n$
9. $\sigma = \sigma + a_{ij} x_j^{(k-1)}$
10.   end
11. $\sigma = (b_i - \sigma)/a_{ii}$
12. $x_i^{(k)} = x_i^{(k-1)} + \omega(\sigma - x_i^{(k-1)})$
13.   end
14.   check convergence; continue if necessary
15. end

Method	Inner Product	SAXPY	Matrix-Vector Product	Precond Solve	Storage Requirement
SOR	0	1	1 [a]	0	matrix+n

Table 2.1: Summary of Operations and Storage Requirements for SOR.

[a] The method performs no real matrix-vector product or preconditioner solve, but the number of operations is equivalent to a matrix-vector multiply.

and vectors that appear in the outermost loop of the algorithm is considered. The *SAXPY* column gives the number of vector operations (excluding inner products) per iteration. n denotes the order of the coefficient matrix. Step 12 of the algorithm contains 1 vector addition. Steps 6, 9, and 12 compute together one component of the corresponding matrix vector product.

2.2.2 Block Iterative Methods

The second type of iterative methods we experimented with is *block iterative methods*. Block iterative methods follow a decompositional approach to solving systems of linear equations. If the model is too large or complex to analyze as an entity, it is divided into subsystems, each of which is analyzed separately, and a global solution is then constructed from the partial solutions. Ideally, the problem is broken into subproblems that can be solved independently, and the global solution is obtained by concatenating the subproblem solutions. When applied to NCD Markov chains, the state space may be ordered and partitioned so that the stochastic matrix of transition probabilities has the form

$$P_{n\times n} = \begin{array}{cc} \begin{array}{cccc} n_1 & n_2 & \cdots & n_N \end{array} & \\ \begin{pmatrix} P_{11} & P_{12} & \cdots & P_{1N} \\ P_{21} & P_{22} & \cdots & P_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ P_{N1} & P_{N2} & \cdots & P_{NN} \end{pmatrix} & \begin{array}{c} n_1 \\ n_2 \\ \vdots \\ n_N \end{array} \end{array}, \tag{2.7}$$

in which the nonzero elements of the off-diagonal blocks are small compared to those of the diagonal blocks. The subblocks P_{ii} are square and of order n_i, for $i = 1, 2, \ldots, N$ and $n = \sum_{i=1}^{N} n_i$. Let $P = \mathrm{diag}(P_{11}, P_{22}, \ldots, P_{NN}) + E$. The quantity $\|E\|_\infty$ is called the *degree of coupling* and it is taken to be the measure of the decomposability of the matrix [12]. Obviously a zero degree of coupling (i.e., $\|E\|_\infty = 0$) implies a completely decomposable matrix. We can also partition the coefficient matrix A (whether it is taken as $-Q^T$ or $I - P^T$) to have a form as in (2.7).

Let the coefficient matrix A be partitioned as

$$\begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1N} \\ A_{21} & A_{22} & \cdots & A_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ A_{N1} & A_{N2} & \cdots & A_{NN} \end{pmatrix} \tag{2.8}$$

To study the convergence of block iterative methods consider the splitting $A = M - N$, where A has the form in (2.8) and M is a nonsingular block diagonal matrix such that

$$M = \mathrm{diag}(A_{11}, A_{22}, \ldots, A_{NN}). \tag{2.9}$$

Theorem 2.5 *[22] Let B be a transition matrix of a finite homogeneous Markov chain. Consider $A = I - B^T$ partitioned as in (2.8) and the splitting $A = M - N$ defined in (2.9). If each matrix A_{ll}, $1 \leq l \leq N$, is either strictly or irreducibly column diagonally dominant, then $\rho(M^{-1}N) = 1$.*

In nearly completely decomposable systems there are eigenvalues close to 1. The poor separation of the unit eigenvalue results in slow rate of convergence for standard matrix iterative methods. Block iterative methods in general do not suffer from this limitation which makes them suitable for such systems. In general block iterative methods require more computation per iteration than stationary iterative methods, but this is usually offset by a faster rate of convergence. In the following subsection we will discuss the two block iterative methods we experimented with: block SOR and iterative aggregation-disaggregation (IAD).

Block SOR

Let us partition the defining homogeneous system of equations $Ax = 0$ as

$$\begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1N} \\ A_{21} & A_{22} & \cdots & A_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ A_{N1} & A_{N2} & \cdots & A_{NN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix} = 0.$$

We now introduce the block splitting:

$$A = D_N - (L_N + U_N),$$

where D_N is a *block* diagonal matrix and L_N and U_N are respectively strictly lower and upper *block* triangular matrices. We then have

$$D_N = \begin{pmatrix} D_{11} & 0 & \cdots & 0 \\ 0 & D_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & D_{NN} \end{pmatrix},$$

$$L_N = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ L_{21} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ L_{N1} & L_{N2} & \cdots & 0 \end{pmatrix}, \quad U_N = \begin{pmatrix} 0 & U_{12} & \cdots & U_{1N} \\ 0 & 0 & \cdots & U_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}.$$

In analogy with equation (2.5), the block SOR method is given by

$$x^{(k+1)} = (1-\omega)x^{(k)} + \omega \left\{ D_N^{-1}(L_N x^{(k+1)} + U_N x^{(k)}) \right\}.$$

If we write this out in full, we get

$$x_i^{(k+1)} = (1-\omega)x_i^{(k)} + \omega \left\{ D_{ii}^{-1} \left(\sum_{j=1}^{i-1} L_{ij} x_j^{(k+1)} + \sum_{j=i+1}^{N} U_{ij} x_j^{(k)} \right) \right\},$$

where the subvectors x_i are partitioned conformally with D_{ii} for $i = 1, 2, \ldots, N$. This implies that at each iteration we must solve N systems of linear equations

$$D_{ii} x_i^{(k+1)} = z_i, \quad i = 1, 2, \ldots, N, \tag{2.10}$$

where

$$z_i = (1-\omega)\, D_{ii}\, x_i^{(k)} + \omega \left(\sum_{j=1}^{i-1} L_{ij} x_j^{(k+1)} + \sum_{j=i+1}^{N} U_{ij} x_j^{(k)} \right), \quad i = 1, 2, \ldots, N.$$

The right-hand side z_i may always be computed before the ith system has to be solved.

The pseudocode of block SOR is given by the following algorithm. Table 2.2 provides the number of operations per iteration in addition to the storage requirement of the method. Operations in steps 6, 9, and 11 are equivalent to 1 *SAXPY* and 1 matrix vector multiplication, where each of the vector length and matrix order is n. Step 12 solves a preconditioned system and is considered as 1 *Precond Solve*. The two vectors that we need to store are x and z.

If the matrix A is irreducible (which is the case in our experiments) then it is clear from (2.10) that at each iteration we are going to solve N nonhomogeneous systems of equations with nonsingular coefficient matrices. This can be achieved by employing either direct or iterative methods. Different criteria may affect the choice of the method to be used for solving a diagonal block as there is no requirement to stick to the same method to solve all diagonal

Algorithm: Block SOR

1. Choose an initial guess $x^{(0)}$ to the solution x.
2. for $k = 1, 2, \ldots$
3.   for $i = 1, 2, \ldots, N$
4. $z_i = 0$
5.   for $j = 1, 2, \ldots, i-1$
6. $z_i = z_i - A_{ij} x_j^{(k)}$
7.   end
8.   for $j = i+1, \ldots, N$
9. $z_i = z_i - A_{ij} x_j^{(k-1)}$
10.   end
11. $z_i = \omega z_i + (1-\omega) A_{ii} x_i^{(k-1)}$
12.   solve $D_{ii} x_i^{(k)} = z_i$
13.   end
14.   check convergence; continue if necessary
15. end

blocks; we will come to the implementation details later in this chapter. In general, for a given coefficient matrix A, the larger the block sizes (and hence the smaller the number of blocks), the fewer the (outer) iterations needed to achieve convergence [36]. The reduction in the number of iterations is usually offset to a certain degree by an increase in the number of operations that are to be performed at each iteration. However, this is not always true as it is highly dependent on the matrix structure.

Method	Inner Product	SAXPY	Matrix-Vector Product	Precond Solve	Storage Requirement
Block SOR	0	1	1	N^a	matrix+2n

Table 2.2: Summary of Operations and Storage Requirements for Block SOR at iteration k.

[a] Since blocks in the partition are not necessarily of the same size, the size of the operands in the given counts are most likely different.

Iterative Aggregation-Disaggregation (IAD)

Suppose we have an NCD Markov chain characterized by a probability matrix P having a block structure as in (2.7), and let the stationary distribution of P, π (i.e., $\pi P = \pi$, $\|\pi\|_1 = 1$), be partitioned conformally with P such that $\pi = (\pi_1, \pi_2, \ldots, \pi_N)$.

For each diagonal block P_{ii}, in the transition probability matrix P, there exists a stochastic complement S_{ii} [21, 36] given by

$$S_{ii} = P_{ii} + P_{i*}(I - P_i)^{-1} P_{*i},$$

where

P_{i*}: $n_i \times (n - n_i)$ matrix is composed of the ith row of blocks of P with P_{ii} removed,

P_{*i}: $(n - n_i) \times n_i$ matrix is composed of the ith column of blocks of P with P_{ii} removed,

P_i: $(n - n_i) \times (n - n_i)$ is the principal submatrix of P with ith row and ith column of blocks removed.

The stochastic complement reflects the behavior of the system within the corresponding block of states. Each stochastic complement is, itself, a stochastic transition probability matrix of an irreducible Markov chain whose state space is composed of the states of the block. The probability that the system is in a certain state of block i given that the process is in one of the states of that block, can be determined from the conditional stationary probability vector of the ith block, $\pi_i/\|\pi_i\|_1$. This can be computed by solving $(\pi_i/\|\pi_i\|_1)S_{ii} = \pi_i/\|\pi_i\|_1$. As can be inferred, a stochastic complement may be too expensive to compute as it has an embedded matrix inversion. One way to overcome this problem is to approximate S_{ii} by accumulating the off-diagonal mass P_{i*} into the diagonal block P_{ii} on a row-by-row basis. This can be achieved in various ways. An approximation to the conditional stationary vector of the corresponding block can then be found by solving the linear system as described before.

It is possible to compute the probability of being in a given block of states if we have an $N \times N$ stochastic matrix whose ijth element denotes the probability of transitioning from block i to block j. This matrix is called the *coupling matrix* and it characterizes the interactions among blocks. To construct this matrix, we need to shrink each block P_{ij} of P down to a single element. This is accomplished by first replacing each row of each block by the sum of its elements. Mathematically, the operation performed for each block is $P_{ij}\,e$. In what follows, e is a column vector of 1's whose length is determined by the context in which it is used. The sum of elements of row k of block P_{ij} gives the probability of leaving state k of block i and entering one of the states of block j. To determine the total probability of leaving (any state of) block i to enter (any state of) block j, we need to sum the elements of $P_{ij}\,e$ after each of these elements has been weighed by the probability that the system is in (one of the states of) block i. These weighing factors may be obtained from the elements of the stationary probability vector; they are the components of $\pi_i/\|\pi_i\|_1$. Hence the ijth element of the coupling matrix is given by $c_{ij} = (\pi_i/\|\pi_i\|_1)P_{ij}e$. The stationary vector of the coupling matrix gives the stationary probability of being in each block of states. More precisely, the multiplicative constants mentioned before, form the elements of the stationary vector of the coupling matrix. However, forming the coupling matrix requires computing the stationary vector. This can be achieved by approximating the coupling matrix by starting with an approximate stationary vector and improving the approximate solution iteratively [12, 33, 36].

The following is an IAD algorithm in which point SOR is used to solve diagonal blocks.

In the IAD algorithm, steps 2 and 3 form the aggregation step. Step 4(b), which is nothing but a block SOR iteration, forms the disaggregation step. Diagonal blocks in step 4(b) are solved using either Gaussian elimination or point SOR, depending on the memory available. An approximation to the stationary distribution of the stochastic complement of P_{ii} is computed in step 2 as $\pi_i^{(k-1)}/\|\pi_i^{(k-1)}\|_1$. In step 3, $\xi^{(k-1)}$ approximates the weighing factors $(\|\pi_1\|_1, \|\pi_2\|_1, \ldots, \|\pi_N\|_1)$. In the IAD algorithm, the residual error (i.e., $\|\pi(I - P)\|$) decreases by a factor of $\|E\|$ at each iteration [12].

Algorithm: IAD

1. Choose an initial guess $\pi^{(0)} = (\pi_1^{(0)}, \pi_2^{(0)}, \ldots, \pi_N^{(0)})$ to the solution π.

for $k = 1, 2, \ldots$

2. Construct the coupling matrix $C_{ij}^{(k-1)}$ such that

$$c_{ij}^{(k-1)} = \frac{\pi_i^{(k-1)}}{\|\pi_i^{(k-1)}\|_1} P_{ij} e.$$

3. Solve the eigenvector problem

$$\xi^{(k-1)} C^{(k-1)} = \xi^{(k-1)}, \quad \|\xi^{(k-1)}\|_1 = 1$$

for $\xi^{(k-1)} = (\xi_1^{(k-1)}, \xi_2^{(k-1)}, \ldots, \xi_N^{(k-1)})$.

4. (a) Compute the row vector

$$z^{(k)} = \left(\xi_1^{(k-1)} \frac{\pi_1^{(k-1)}}{\|\pi_1^{(k-1)}\|_1}, \ldots, \xi_N^{(k-1)} \frac{\pi_N^{(k-1)}}{\|\pi_N^{(k-1)}\|_1} \right)$$

(b) Compute the N systems of equations

$$\begin{aligned} \pi_i^{(k)} &= \pi_i^{(k)} P_{ii} + (1-\omega) z_i^{(k)} (I - P_{ii}) \\ &\quad + \omega \left(\sum_{j<i} \pi_j^{(k)} P_{ji} + \sum_{j>i} z_j^{(k)} P_{ji} \right) \end{aligned}$$

for $\pi_i^{(k)}, \quad i = 1, 2, \ldots, N$.

5. Check for convergence; continue if necessary

end

One of the crucial steps in the IAD algorithm is solving equation $\xi^{(k)} C^{(k)} = \xi^{(k)}$ subject to $\|\xi^{(k)}\|_1 = 1$ (step 3 of the IAD algorithm). The coupling matrix is a singular irreducible stochastic matrix of order N whose states form a single communicating class. Consequently $C^{(k)}$ has a unique unit eigenvalue and $(N-1)$ other eigenvalues close to 1. The smaller the degree of coupling the closer these other eigenvalues to 1. A careful inspection leads us to notice that the whole problem boils down to solving the system of linear equations

$$(I - C^{(k)})^T (\xi^{(k)})^T = 0, \quad \|\xi^{(k)}\|_1 = 1, \tag{2.11}$$

which is similar to the original problem (i.e., $(I-P)^T \pi^T = 0$, subject to the normalization constraint $\|\pi\|_1 = 1$).

Method	Inner Product	SAXPY	Matrix-Vector Product	Precond Solve	Storage Requirement
IAD	1	2	2	$(N+1)^a$	2 matrices[b]+N+2n

Table 2.3: Summary of Operations and Storage Requirements for IAD at iteration k.

[a] Since the number of blocks and the order of diagonal blocks in the partition are not necessarily of the same size, the size of the operands in the given counts are most likely different.

[b] Two square matrices of orders n and N.

Table 2.3 shows the number of computations and the storage required by IAD for the outer loop. The matrix vector products required for building the coupling matrix in step 2 are equivalent to 1 order n matrix vector product. The same argument is valid for the matrix vector products in step 4(b). There is roughly 1 inner product of length n, which comes from step 2. The 2 order n *SAXPY* operations come from step 4(b). Step 3 consists of solving a linear system of order N and is 1 *Precond Solve*. Step 4(b) solves N linear systems of order n_i each, rising the total number of *Precond Solve* to $N+1$. To perform these operations, the coupling matrix and the transition matrix, and vectors ξ, z, and π are needed to be stored.

There is no requirement to solve equation (2.11) with the same method used to solve the original problem. We choose to use Gaussian elimination (GE) for several reasons. $(I - C^{(k)})^T$ is a singular M-matrix with 0 column sums. For such a matrix, GE preserves column diagonal dominance throughout its computation so that the multiplier element at each step is bounded by 1 thereby avoiding the need of pivoting. This follows from the fact that at each step the pivot has the largest magnitude among all elements lying in the unreduced part of its respective column. Besides, iterative methods tend to converge slowly as all the nonunit eigenvalues of $C^{(k)}$ are close to 1. On the other hand, (ordinary) GE suffers from unstability in the presence of rounding errors on such coupling matrices [12, 10].

Partitioning Techniques

Three block partitioning techniques are considered. The first one, *near-complete decomposability test (ncd test)*, determines the strongly connected components of the transition probability matrix by ignoring the elements that are less than a prespecified decomposability parameter γ. If the matrix is not already in the form (2.7), then symmetric permutations are applied to put it into the form in which the diagonal blocks form the strongly connected aggregates. These strongly connected aggregates are determined using Tarjan's algorithm. The parameter γ is taken as an approximation of the degree of coupling.

The two other partitioning techniques are based on straightforward algorithms. The *equal* partitioning has $\sqrt{n}$ equal sized blocks of order $\sqrt{n}$ if n is a perfect square. If $n \neq \lfloor\sqrt{n}\rfloor^2$, there is an extra block of order $n - \lfloor\sqrt{n}\rfloor^2$. The second straightforward partitioning, *other*, has nb blocks of orders respectively $1, 2, \ldots, nb$ if $n = \sum_{i=1}^{nb} i$ (and possibly an extra block of order $n - \sum_{i=1}^{nb} i$ if the difference is positive). This last partitioning ensures that there are about $\sqrt{2n}$ blocks and the largest block solved is of order roughly $\sqrt{2n}$. All three partitionings are part of MARCA [34].

2.2.3 Projection Methods

Projection methods differ from stationary and block iterative methods in that successive approximations are computed from small dimension subspaces. Projection methods, themselves, differ from each other in the way subspaces are selected and solution approximations are extracted from them. A projection step is defined formally with two objects: a subspace $\mathcal{K}$ of dimension m from which the approximation is to be selected and another subspace $\mathcal{L}$ (of the same size m) that is used to set the constraints necessary to extract the new approximated solution vector from $\mathcal{K}$ [24, 28]. Consider the linear system

$$Ax = b. \tag{2.12}$$

Throughout this section we will base our discussion on equation (2.12) though it does not exactly correspond to our real problem (see equation (2.1)). To make it so, we just have to set the right-hand side vector b to 0 and satisfy the normalization constraint for the computed approximation.

Let $V = [v_1, v_2, \ldots, v_m]$ and $W = [\omega_1, \omega_2, \ldots, \omega_m]$ be respectively the bases of $\mathcal{K}$ and $\mathcal{L}$. Then we can write the approximate solution as $\tilde{x} = Vy$, where y is now a vector of $\mathbb{R}^m$. This gives us m degrees of freedom, and in order to extract a unique y we require that the residual vector $b - A\tilde{x}$ be orthogonal to $\mathcal{L}$. i.e.,

$$b - AVy \perp \omega_i, \qquad i = 1, 2, \ldots, m.$$

In matrix form this can be written as

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

which yields,

$$y = [W^T AV]^{-1} W^T b.$$

Thus the minimum assumption that must be made in order for those projection processes to be feasible is that $W^T AV$ be nonsingular. If we start with $x^{(0)}$ as an initial approximate solution to the system, then $x^{(0)}$ may be adjusted by a vector δ such that $x^{(0)} + \delta$ is a solution, i.e., $A(x^{(0)} + \delta) = b$. If we set $r_0 \equiv b - Ax^{(0)}$, then

$$A(x^{(0)} + \delta) = b \quad \Rightarrow \quad Ax^{(0)} + A\delta = b \quad \Rightarrow \quad A\delta = b - Ax^{(0)} = r_0,$$

and hence the projection step is applied to the system $A\delta = r_0$ to compute the unknown vector δ. It follows that a general projection algorithm is as follows:

Algorithm: Prototype Projection Method

Until Convergence Do:

1. Select a pair of subspaces $\mathcal{K}$ and $\mathcal{L}$, and an initial guess $\tilde{x}$.
2. Choose bases $V = [v_1, v_2, \ldots, v_m]$ and $W = [\omega_1, \omega_2, \ldots, \omega_m]$ for $\mathcal{K}$ and $\mathcal{L}$.
3. Compute

$$\begin{aligned} r &\leftarrow b - A\tilde{x}, \\ y &\leftarrow [W^T AV]^{-1} W^T r, \\ \tilde{x} &\leftarrow \tilde{x} + Ay. \end{aligned}$$

Let (x, y) denote the inner product of vectors x and y. For a matrix A we will denote by $\|x\|_A$ the A-norm of vector x, defined by $\|x\|_A \equiv (Ax, x)^{1/2}$.

Projection methods are classified in two main groups [24, 28]. The first is when the *Krylov subspace* $\mathcal{K}$ is taken as $\mathcal{K} = \mathcal{L} = span\{r_0, Ar_0, \ldots, A^{m-1}r_0\}$ and $V = W$ is an orthogonal basis of $\mathcal{K}$. This represents the class of *Galerkin projection methods* (also known as *orthogonal projection methods*). In this type of methods, each iteration minimizes $\|x - \tilde{x}\|_A$ in the direction of the residual vector r $(= b - A\tilde{x})$. The second type of projection methods is when $\mathcal{L} = A\mathcal{K} = span\{Ar_0, A^2r_0, \ldots, A^m r_0\}$ (and hence $W = AV$). Each iteration of this kind of methods minimizes the 2-norm of the residual vector, i.e., $\|b - A\tilde{x}\|_2 = \min_{z \in \mathcal{K}} \|b - Az\|_2$. This explains why these methods are referred to as the *minimal residual methods*.

It is well known that orthogonal projection methods generally converge faster than minimal residual methods for symmetric positive definite matrices [24]. This is not the case for nonsymmetric problems as the A-norm may be degenerate if the coefficient matrix is not positive definite.

Suppose we are going to use a projection method to solve the eigensystem $Ax = \lambda_i x$, where λ_i is the ith largest eigenvalue of A in modulus. Let m be the dimension of the Krylov subspace. Then the rate of convergence [36] of the method will be $|\lambda_{m+1}/\lambda_i|$. When the eigenvector corresponding to the dominant eigenvalue is required, this convergence factor is given by $|\lambda_{m+1}/\lambda_1|$. Recall that our goal is to solve an eigensystem of a stochastic matrix for an eigenvector of unit 1-norm corresponding to the dominant eigenvalue. Since the dominant eigenvalue of a stochastic matrix is 1, the convergence factor is simply $|\lambda_{m+1}|$.

Generalized Minimal Residual (GMRES)

The Generalized Minimal Residual method lies in the class of minimal residual methods and is designed to solve unsymmetric linear systems. The GMRES version [4] discussed in this section is based on the *Arnoldi method* which is a modified *Gram-Schmidt orthogonalization procedure* applied to the Krylov subspace $span\{r_0, Ar_0, \ldots, A^{m-1}r_0\}$ to form the basis of the subspace and store it in a Hessenberg matrix.

The GMRES iterates are constructed as

$$x^{(i)} = x^{(0)} + y_1 v^{(1)} + \cdots + y_i v^{(i)},$$

where y_k are the coefficients that minimize the residual 2-norm $\|b - Ax^{(i)}\|_2$. The GMRES algorithm has the property that this residual norm can be determined before computing the iterate. This enables us to postpone the expensive operation of forming the iterate until the residual norm is deemed small enough. To control the storage requirements, restarts are used, i.e., the iterate is formed after each m iterations. At each restart a new basis of the Krylov subspace is formed. The crucial element for successful application of GMRES(m) resolves around the decision of when to restart, that is the choice of m. Obviously if no restarts are used (i.e., $m = n$), GMRES, and all orthogonalizing Krylov subspace methods, converge in n steps. However, because of storage limitation this may not be feasible for a large n.

We are providing the pseudocode [4] for the restarted GMRES(m) algorithm with preconditioner M. The main idea behind preconditioning is to retransform the linear system so that the difference between the dominant and the subdominant eigenvalues of the preconditioned coefficient matrix is larger than what it used to be in the original system. We will come back to preconditioning techniques later in this chapter.

Table 2.4 shows the number of operations and storage requirement for GMRES at iteration i. In the table we do not consider steps 14 through 16 as they depend on the way the algorithm is coded. In our code, for instance, we do not store matrices $J_1, \ldots, J_i$ explicitly and the Jacobi rotations are performed on the nonzero elements which are the only values we store. The $i + 1$ inner products, reported in the table, come from steps 9 and 12. Computing the new approximation in step 17 (or 19) requires i *SAXPY*. The other i *SAXPY* operations come from step 10. One matrix vector product and one *Precond Solve* come from step 7. To perform these operations we need to store matrices A and M, the first $i + 1$ vectors of the Krylov subspace basis, the first i vectors of H, and vectors $\tilde{s}$, ω, $\tilde{x}$, and $x^{(0)}$. As can be seen from Table 2.4, the amount of computation and storage required by GMRES in one iteration increases linearly with the (inner) iteration count i. This is regarded as the major drawback of the method.

Method	Inner Product	SAXPY	Matrix-Vector Product	Precond Solve	Storage Requirement
GMRES(m)	$i+1$	$2i$	1	1	2 matrices+ $(n+m)i+5n$

Table 2.4: Summary of Operations and Storage Requirements for GMRES(m) at iteration i.

BiConjugate Gradient (BCG)

The BiConjugate Gradient method is an orthogonal projection method and it takes an advantage over GMRES by reducing the storage demand [4]. This is achieved by replacing the orthogonal sequence of residuals (formed by GMRES to build the basis of the Krylov subspace) by two mutually orthogonal sequences of residual vectors

$$r^{(i)} = r^{(i-1)} - \alpha_i A p^{(i)}, \qquad \tilde{r}^{(i)} = \tilde{r}^{(i-1)} - \alpha_i A^T \tilde{p}^{(i)}.$$

The two sequences of search directions are

$$p^{(i)} = r^{(i-1)} + \beta_{i-1} p^{(i-1)}, \qquad \tilde{p}^{(i)} = \tilde{r}^{(i-1)} + \beta_{i-1} \tilde{p}^{(i-1)}.$$

To ensure the bi-orthogonality relation

$$\tilde{r}^{(i)^T} r^{(j)} = \tilde{p}^{(i)^T} A p^{(j)} = 0 \qquad \text{for } i \neq j,$$

we set

$$\alpha_i = \frac{\tilde{r}^{(i-1)^T} r^{(i-1)}}{\tilde{p}^{(i)^T} A p^{(i)}}, \qquad \beta_i = \frac{\tilde{r}^{(i)^T} r^{(i)}}{\tilde{r}^{(i-1)^T} r^{(i-1)}}.$$

It is observed that the convergence behavior of BCG is quite irregular. The method breaks down when $z^{(i-1)^T} \tilde{r}^{(i-1)} \approx 0$. Another possible breakdown

Algorithm: Preconditioned GMRES(m)

```
Choose an initial guess $x^{(0)}$ to the solution $x$.
for $j = 1, 2, \ldots$
    Solve $r$ from $Mr = b - Ax^{(0)}$
    $v^{(1)} = r/\|r\|_2$
    $s := \|r\|_2 e_1$
    for $i = 1, 2, \ldots, m$
        Solve $\omega$ from $M\omega = Av^{(i)}$
        for $k = 1, 2, \ldots, i$
            $h_{k,i} = (\omega, v^{(k)})$
            $\omega = \omega - h_{k,i} v^{(k)}$
        end
        $h_{i+1,i} = \|\omega\|_2$
        $v^{(i+1)} = \omega / h_{i+1,i}$
        apply $J_1, \ldots, J_{i-1}$ on $(h_{1,i}, \ldots, h_{i+1,i})$
        construct $J_i$ acting on $i$th and $(i+1)$st component
        of $h_{.,i}$ such that $(i+1)$st component of $J_i h_{.,i}$ is 0
        $s := J_i s$
        if $s(i+1)$ is small enough then (UPDATE($\tilde{x}, i$) and quit)
    end
    UPDATE($\tilde{x}, m$)
end
```

In this scheme UPDATE($\tilde{x}, i$)
replaces the following computations:

Compute y as the solution of $Hy = \tilde{s}$, in which
the upper $i \times i$ triangular part of H has $h_{i,j}$ as
its elements (in least squares sense if H is singular),
$\tilde{s}$ represents the first i components of s
$x^{(i)} = x^{(0)} + y_1 v^{(1)} + \cdots + y_i v^{(i)}$
$s^{(i+1)} = \|b - A\tilde{x}\|_2$
if $\tilde{x}$ is an accurate enough approximation the quit
else $x^{(0)} = \tilde{x}$

Algorithm: Preconditioned BCG

1. Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$.
2. Choose $\tilde{r}^{(0)}$ (for example, $\tilde{r}^{(0)} = r^{(0)}$).
3. for $i = 1, 2, \ldots$
4.   Solve $Mz^{(i-1)} = r^{(i-1)}$
5.   Solve $M^T\tilde{z}^{(i-1)} = \tilde{r}^{(i-1)}$
6. $\rho_{i-1} = z^{(i-1)^T}\tilde{r}^{(i-1)}$
7.   if $\rho_{i-1} = 0$, method fails
8.   if $i = 1$
9. $p^{(i)} = z^{(i-1)}$
10. $\tilde{p}^{(i)} = \tilde{z}^{(i-1)}$
11.   else
12. $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
13. $p^{(i)} = z^{(i-1)} + \beta_{i-1}p^{(i-1)}$
14. $\tilde{p}^{(i)} = \tilde{z}^{(i-1)} + \beta_{i-1}\tilde{p}^{(i-1)}$
15.   endif
16. $q^{(i)} = Ap^{(i)}$
17. $\tilde{q}^{(i)} = A^T\tilde{p}^{(i)}$
18. $\alpha_i = \rho_{i-1}/\tilde{p}^{(i)^T}q^{(i)}$
19. $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$
20. $r^{(i)} = r^{(i-1)} + \alpha_i q^{(i)}$
21. $\tilde{r}^{(i)} = \tilde{r}^{(i-1)} + \alpha_i\tilde{q}^{(i)}$
22. end

situation is when $\tilde{p}^{(i)^T}q^{(i)} \approx 0$. To increase the effectiveness of BCG, variants such as CGS and BCGStab have been proposed.

Table 2.5 shows the number of operations and the storage requirement for BCG per iteration. The 2 inner products come from steps 6 and 18. The 5 *SAXPY* come from steps 13, 14, 19, 20, and 21. The algorithm contains 2 matrix vector products in steps 16 and 17 (one with A and one with A^T). Steps 4 and 5 correspond to solving two linear systems (with M and M^T as coefficient matrices). We need to store matrices A and M, and 9 other vectors that we use in the algorithm.

Method	Inner Product	SAXPY	Matrix-Vector Product	Precond Solve	Storage Requirement
BCG	2	5	1/1	1/1	2 matrices+$9n$

Table 2.5: Summary of Operations and Storage Requirements for BCG at iteration i. "a/b" means "a" operations with the matrix and "b" with its transpose.

Conjugate Gradient Squared (CGS)

Consider the residual vector $r^{(i)}$ computed at the ith iteration of BCG. This vector may be written as a product of $r^{(0)}$ and an ith degree polynomial in A [4] such that

$$r^{(i)} = P_i(A)\, r^{(0)}.$$

The same polynomial is applicable to $\tilde{r}^{(i)}$ (i.e., $\tilde{r}^{(i)} = P_i(A)\,\tilde{r}^{(0)}$).

As it can be inferred, the role of the polynomial $P_i(A)$ is to reduce the initial residual $r^{(0)}$ to $r^{(i)}$ in i iterations. Therefore, applying the same polynomial twice (i.e., $r^{(i)} = P_i^2(A)\, r^{(0)}$) will logically reduce $r^{(0)}$ much faster. This approach leads to the Conjugate Gradient Squared method.

The rate of convergence of CGS is generally twice that of BCG. However, this is not always the case since a reduced residual vector $r^{(k)}$ may not be reduced any further. This explains the highly irregular behavior of CGS. Moreover, rounding errors are very likely to occur in CGS as local corrections to the current solution may be very large, and hence the finally computed solution may not be very accurate [4]. Another property which seems to be paradoxic at first glance is that the method tends to diverge if we choose to start with an initial guess close to the solution. For what concerns the time complexity, CGS is almost as expensive as BCG. However, it is worth mentioning that CGS does not involve computations with A^T.

Table 2.6 shows the number of operations and the storage requirement for CGS per iteration. The 2 inner products come from steps 4 and 16. The 6

Algorithm: Preconditioned CGS

1. Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$.
2. Choose $\tilde{r}^{(0)}$ (for example, $\tilde{r}^{(0)} = r^{(0)}$).
3. for $i = 1, 2, \ldots$
4. $\quad \rho_{i-1} = \tilde{r}^T r^{(i-1)}$
5. $\quad$ if $\rho_{i-1} = 0$, method fails
6. $\quad$ if $i = 1$
7. $\quad\quad u^{(1)} = r^{(0)}$
8. $\quad\quad p^{(1)} = u^{(1)}$
9. $\quad$ else
10. $\quad\quad \beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
11. $\quad\quad u^{(i)} = r^{(i-1)} + \beta_{i-1} q^{(i-1)}$
12. $\quad\quad p^{(i)} = u^{(i)} + \beta_{i-1}(q^{(i-1)} + \beta_{i-1} p^{(i-1)})$
13. $\quad$ endif
14. $\quad$ Solve $M\hat{p} = p^{(i)}$
15. $\quad \hat{v} = A\hat{p}$
16. $\quad \alpha_i = \rho_{i-1}/\tilde{r}^T \hat{v}$
17. $\quad q^{(i)} = u^{(i)} - \alpha_i \hat{v}$
18. $\quad$ Solve $M\hat{u} = u^{(i)} + q^{(i)}$
19. $\quad x^{(i)} = x^{(i-1)} + \alpha_i \hat{u}$
20. $\quad \hat{q} = A\hat{u}$
21. $\quad r^{(i)} = r^{(i-1)} - \alpha_i \hat{q}$
22. $\quad$ check convergence; continue if necessary
23. end

SAXPY operations come from steps 11, 12, 17, 18, 19, and 21. Each of steps 15 and 20 contains 1 matrix vector product. In each of steps 14 and 18, the algorithm solves a linear system. The algorithm requires storage for matrices A, M and 10 other vectors.

BiConjugate Gradient Stabilized (BCGStab)

The BiConjugate Gradient Stabilized method was developed so that it is as fast as CGS while avoiding the often irregular convergence patterns of the latter [4]. It can be then said that BCGStab is suitable for nonsymmetric linear systems. The idea behind this method is to use an ith degree polynomial other than P_i (say Q_i) to further reduce the residual vector [37]. In other words, instead of

Method	Inner Product	SAXPY	Matrix-Vector Product	Precond Solve	Storage Requirement
CGS	2	7	2	2	2 matrices+$10n$

Table 2.6: Summary of Operations and Storage Requirements for CGS at iteration i.

Method	Inner Product	SAXPY	Matrix-Vector Product	Precond Solve	Storage Requirement
BCGStab	4	6	2	2	2 matrices+$9n$

Table 2.7: Summary of Operations and Storage Requirements for BCGStab at iteration i.

writing the residual as $r^{(i)} = P_i^2(A)r^{(0)}$, we write $r^{(i)} = Q_i(A)P_i(A)r^{(0)}$.

Table 2.7 shows the number of operations and the storage requirement for BCGStab per iteration. Each of steps 4 and 14 contains 1 inner product whereas step 19 contains 2. The highest number of *SAXPY* operations that the algorithm may perform per iteration is 6, which comes from steps 10, 15, 20, and 21. The algorithm performs 2 matrix vector multiplications in steps 13 and 18. Steps 12 and 17 correspond to solving 2 linear systems. It is necessary to store the matrices A, M and 9 additional vectors. BCGStab requires slightly more computations per iteration than CGS and BCG as it requires two matrix-vector products and four inner products.

Quasi-Minimal Residual (QMR)

The Quasi-Minimal Residual method [13, 14] attempts to overcome the problems of irregular convergence behavior and breakdowns observed in some of the projection methods such as BCG. QMR uses a least squares approach similar to that followed in GMRES. However, GMRES uses an orthogonal basis

Algorithm: Preconditioned BCGStab

Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$.
Choose $\tilde{r}^{(0)}$ (for example, $\tilde{r}^{(0)} = r^{(0)}$).
for $i = 1, 2, \ldots$
    $\rho_{i-1} = \tilde{r}^T r^{(i-1)}$
    if $\rho_{i-1} = 0$, method fails
    if $i = 1$
        $p^{(1)} = r^{(0)}$
    else
        $\beta_{i-1} = (\rho_{i-1}/\rho_{i-2})(\alpha_{i-1}/\omega_{i-1})$
        $p^{(i)} = r^{(i-1)} + \beta_{i-1}(p^{(i-1)} - \omega_{i-1} v^{(i-1)})$
    endif
    Solve $M\hat{p} = p^{(i)}$
    $\hat{v} = A\hat{p}$
    $\alpha_i = \rho_{i-1}/\tilde{r}^T \hat{v}$
    $s = r^{(i-1)} - \alpha_i v^{(i)}$
    check norm of s; if small enough: set $x^{(i)} = x^{(i-1)} + \alpha_i \hat{p}$ and stop
    Solve $M\hat{s} = s$
    $t = A\hat{s}$
    $\omega_i = t^T s / t^T t$
    $x^{(i)} = x^{(i-1)} + \alpha_i \hat{p} + \omega_i \hat{s}$
    $r^{(i)} = s - \omega_i t$
    check convergence; continue if necessary
    for continuation it is necessary that $\omega_i \neq 0$
end

for the constructed Krylov subspace whereas QMR uses a bi-orthogonal one. Thereby, the obtained solution is viewed as quasi-minimal residual solution, which explains the name.

To avoid breakdowns, QMR uses look-ahead techniques which makes it more robust than BCG. These techniques enable QMR to prevent all breakdowns except the so-called "incurable breakdown". The version of QMR [4] we used in our experiments is simpler than the full QMR method with look-ahead, but it is still more robust than BCG. The algorithm we used includes a relatively inexpensive recurrence relation for computing the residual vector at the expense of a few extra vectors of storage and vector update operations per iteration. It also avoids performing a matrix-vector product to compute the residual vector. A full-fledged implementation of QMR with look-ahead is available

notation	M_1	M_2
QMR 1	I	$\tilde{L}\tilde{U}$
QMR 2	$\tilde{L}\tilde{U}$	I
QMR 3	$\tilde{L}$	$\tilde{U}$

Table 2.8: Preconditioners used in the QMR versions.

Method	Inner Product	SAXPY	Matrix-Vector Product	Precond Solve	Storage Requirement
QMR	2	8	1/1	1/1	2 matrices+15n

Table 2.9: Summary of Operations and Storage Requirements for QMR at iteration i.

through *netlib*.

In the QMR pseudocode provided, we use a preconditioner $M = M_1 M_2$. Based on the choice of M_1 and M_2 we implemented three versions of QMR, which we denote by *QMR 1*, *QMR 2*, and *QMR 3*. If an incomplete LU decomposition is used as a preconditioner so that $M = \tilde{L}\tilde{U}$, where $\tilde{L}$ and $\tilde{U}$ are lower and upper triangular matrices formed by an incomplete factorization of the coefficient matrix, then the choices of M_1 and M_2 in each version are as shown in Table 2.8.

Table 2.9 shows the number of operations and the storage requirement for QMR per iteration. Steps 19 and 25 contain one matrix vector product each. The 8 *SAXPY* operations come from steps 16, 17, 22, 25, 33, 34, 36 and 37. Steps 11 and 23 together require the same number of operations as solving a linear system directly with M as the coefficient matrix. The same argument is valid for steps 12 and 26, but this time for the transpose system. Storage for matrices A, M and 15 additional vectors are required by the algorithm.

Algorithm: Preconditioned QMR without Look-ahead

Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$
$v^{(1)} = r^{(0)}$; solve $M_1 y = v^{(1)}$; $\rho_1 = \|y\|_2$
Choose $\tilde{\omega}^{(1)}$, for example $\tilde{\omega}^{(1)} = r^{(0)}$
Solve $M_2 z = \tilde{\omega}^{(1)}$; $\xi_1 = \|z\|_2$
$\gamma_0 = 1$; $\eta_0 = -1$
for $i = 1, 2, \ldots$
 if $\rho_i = 0$ or $\xi_i = 0$ method fails
 $v^{(i)} = \tilde{v}^{(i)}/\rho_i$; $y = y/\rho_i$
 $\omega^{(i)} = \tilde{\omega}^{(i)}/\xi_i$; $z = z/\xi_i$
 $\delta_i = z^T y$; if $\delta_i = 0$ method fails
 solve $M_2 \tilde{y} = y$
 solve $M_1^T \tilde{z} = z$
 if $i = 1$
  $p^{(1)} = \tilde{y}$; $q^{(1)} = \tilde{z}$
 else
  $p^{(i)} = \tilde{y} - (\xi_i \delta_i / \epsilon_{i-1}) p^{(i-1)}$
  $q^{(i)} = \tilde{z} - (\rho_i \delta_i / \epsilon_{i-1}) q^{(i-1)}$
 endif
 $\tilde{p} = A p^{(i)}$
 $\epsilon_i = {q^{(i)}}^T \tilde{p}$; if $\epsilon_i = 0$ method fails
 $\beta_i = \epsilon_i / \delta_i$; if $\beta_i = 0$ method fails
 $v^{(i+1)} = \tilde{p} - \beta v^{(i)}$
 solve $M_1 y = v^{(i+1)}$
 $\rho_{i+1} = \|y\|_2$
 $\tilde{\omega}^{(i+1)} = A^T q^{(i)} - \beta_i \omega^{(i)}$
 solve $M_2^T z = \tilde{\omega}^{(i+1)}$
 $\xi_{i+1} = \|z\|_2$
 $\theta_i = \rho_{i+1} / (\gamma_{i-1} |\beta_i|)$; $\gamma_i = 1/\sqrt{1 + \theta_i^2}$; if $\gamma_i = 0$ method fails
 $\eta_i = -\eta_{i-1} \rho_i \gamma_i^2 / (\beta_i \gamma_{i-1}^2)$
 if $i = 1$
  $d^{(1)} = \eta_1 p^{(1)}$; $s^{(1)} = \eta_1 \tilde{p}$
 else
  $d^{(i)} = \eta_i p^{(i)} + (\theta_{i-1} \gamma_i)^2 d^{(i-1)}$
  $s^{(i)} = \eta_i \tilde{p} + (\theta_{i-1} \gamma_i)^2 s^{(i-1)}$
 endif
 $x^{(i)} = x^{(i-1)} + d^{(i)}$
 $r^{(i)} = r^{(i-1)} - s^{(i)}$
 check convergence; continue if necessary
end

2.2.4 Stopping Criteria

One of the most critical steps in iterative methods is to decide when to stop the iteration. A good stopping criterion should

1. identify when the error $e^{(i)} \equiv x^{(i)} - x$ is small enough to stop,
2. stop if the error is no longer decreasing, or decreasing too slow, and
3. limit the maximum amount of time spent iterating.

Ideally the iteration should stop when the magnitudes of entries of the error $e^{(i)}$ fall below a user supplied threshold. Nevertheless, since the exact solution x is generally not known, it is practically not feasible to compute $e^{(i)}$. Instead, the residual vector $r^{(i)} = b - Ax^{(i)}$ which is more readily computed, is used. We will later show how we can bound $e^{(i)}$ in terms of $r^{(i)}$.

The stopping criterion we used in the projection methods of interest is

$$\textbf{stop if} \quad i \geq maxit \quad \textbf{or} \quad \|r^{(i)}\|_\infty \leq stop\_tol,$$

where

- i is the iteration count,
- $maxit$ is the maximum number of iterations the algorithm will be permitted to perform, and
- $stop\_tol$ is the user-specified parameter which should be less than 1 and greater than machine ε.[3]

The user may choose the value of $stop\_tol$ as the approximate uncertainty in the entries of A and b relative to $\|A\|$ [4] and $\|b\|$, respectively. The stopping tolerance $stop\_tol$ we used in our experiments is 10^{-10} which means that we are considering the entries of A (our b is 0) to have errors in the range $\pm 10^{-10}\|A\|$.

[3]On a machine with IEEE Standard Floating Point Arithmetic, $\varepsilon = 2^{-24} \approx 10^{-7}$ in single precision, and $\varepsilon = 2^{-53} \approx 10^{-16}$ in double precision.

[4]The norm of the vector is not important as long as we are consistent.

We should point out that the stopping criterion we use for GMRES is slightly different than the one stated before. In GMRES we use the previously mentioned criterion as a convergence test at the end of each restart, but at each inner-loop termination we compare $\|r^{(i)}\|_2$ (and not $\|r^{(i)}\|_\infty$) with *stop_tol* since it is readily available.

Since $\|e^{(i)}\|$, which is known as the *forward error*, is hard to estimate directly, we usually use the *backward error* as a tool to bound the forward error. The norm wise backward error is defined [3] as the smallest possible value of $\max\{\|\delta A\|/\|A\|, \|\delta b\|/\|b\|\}$ where $x^{(i)}$ is the exact solution of $(A+\delta A)x^{(i)} = (b+\delta b)$ (here δA denotes a general matrix, not $\delta \times A$; the same goes for δb), and it can also be written as $\|b - Ax^{(i)}\|/\|A\|$. The backward error is more practical to use than the forward error as it can be easily computed from $r^{(i)}$.

$$e^{(i)} = x^{(i)} - x = A^{-1}(Ax^{(i)} - b),$$

hence

$$\begin{aligned} \|e^{(i)}\| &\le \|A^{-1}\| \cdot \|Ax^{(i)} - b\| && (2.13) \\ &= \|A^{-1}\| \cdot \|A\| \cdot \frac{\|r^{(i)}\|}{\|A\|}. && (2.14) \end{aligned}$$

For a singular matrix A, the *group inverse* $A^{\#}$ can replace A^{-1} in equations (2.13) and (2.14). The expression $\|A^{-1}\| \cdot \|A\|$ is referred to as the *condition number* of A. From equation (2.13) we can see that if the algorithm stops due to the test $\|r^{(i)}\| \le stop\_tol$, the forward error can be upper-bounded by $stop\_tol\,\|A^{-1}\|$. There also exist the concepts of *relative forward error*, defined by $\|e^{(i)}\|/\|x^{(i)}\|$, and *relative backward error*, defined by $\|b - Ax^{(i)}\|/(\|A\| \cdot \|x^{(i)}\|)$. Directly from equation (2.14) we can upper-bound the relative forward error in the following way:

$$\frac{\|e^{(i)}\|}{\|x^{(i)}\|} \le \|A^{-1}\| \cdot \|A\| \cdot \frac{\|r^{(i)}\|}{\|A\| \cdot \|x^{(i)}\|}.$$

We are reporting the relative backward error in all the experiments we conducted (see appendix A).

The stopping criteria we used in SOR and the block iterative methods we discussed are respectively

$$\textbf{stop if} \quad i \ge maxit \quad \textbf{or} \quad \|x^{(i)} - x^{(i-1)}\|_\infty \le stop\_tol,$$

and

$$\textbf{stop if} \quad i \geq maxit \ \ \textbf{or} \quad \|x^{(i)} - x^{(i-1)}\|_\infty \leq stop\_tol \ \ \textbf{or}$$
$$(\|x^{(i)} - x^{(i-1)}\|_\infty \leq stop\_tol_1 \ \ \textbf{and} \quad \|r^{(i)} - r^{(i-1)}\|_\infty \leq stop\_tol_2).$$

In the experiments we set $stop\_tol$, $stop\_tol_1$, $stop\_tol_2$ to 10^{-10}, 10^{-6}, 10^{-12} respectively. $stop\_tol_2$ forces the algorithm to terminate when the norm of the residual is decreasing too slowly while the difference between two successive iterates is relatively small enough.

Different stopping criteria has been suggested for the convergence test of iterative methods. Several criteria are discussed in [3]. Unfortunately, there is no single stopping criterion known to be suitable for all iterative methods. Hence, selecting the most appropriate one is a difficult decision to make during the implementation of the solver. However, knowing the solvers and their byproduct helps. The amount of computation required by the convergence test is another constraint which should be taken into consideration.

2.2.5 Preconditioners

A very important issue in iterative methods in the concept of *preconditioning*. Although preconditioning can be used in all iterative methods, we employed it in projection methods only. The idea behind preconditioning is to accelerate the convergence process by redistributing the eigenvalues of the coefficient matrix so that the difference between the dominant and the subdominant eigenvalues becomes larger without changing the solution vector sought. Therefore, the need for a preconditioner becomes vital when dealing with NCD systems.

Again let us consider the system of linear equations

$$Ax = b.$$

This can be transformed into the *right-preconditioned* equivalent system

$$AM^{-1}(Mx) = b,$$

or into the *left-preconditioned* equivalent system

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

where the preconditioned matrix M (also called *preconditioner*) has the property that it is a cheap approximation of A. The more M resembles A, the faster the method converges [20]. In the case of right-preconditioning the system $AM^{-1}y = b$ is solved for the unknown $y \equiv Mx$, and the final solution x is obtained through the post-transformation $x = M^{-1}y$. To use right-preconditioning, M should also be chosen so that $M^{-1}v$ is cheap to compute for any arbitrary vector v.

In the left-preconditioning case, the system is solved based on imposing the necessary stopping constraints on the preconditioned residual vector $r = M^{-1}(b - Ax)$. In this case, M^{-1} may not be formed explicitly and the preconditioned residual is computed by solving the system $Mr = b - Ax$. Therefore, the preconditioner M should be chosen so that solving any linear system of the form $Mv = u$ for any vector v cheap.

Various types of preconditioners have been (and are still) developed (see [29, 6]). Their efficiency is highly dependent on the system to be solved and it is quite difficult to forecast which preconditioner is the best for a given system. In this study, we are only considering preconditioners obtained from *incomplete LU factorization (ILU)* [8]. First, an LU decomposition of the coefficient matrix A is initiated. Throughout the decomposition, nonzero elements are omitted according to different rules. These rules characterize the ILU type. Thus, instead of ending up with an exact LU decomposition, what we obtain is of the form

$$A = \tilde{L}\tilde{U} + E,$$

where E, called the *remainder*, is expected to be small in some sense. $\tilde{L}$ and $\tilde{U}$ respectively are lower and upper triangular matrices. In all the projection methods we implemented, we stick to left-preconditioning and take $M = \tilde{L}\tilde{U}$, thereby the fest choice on the preconditioner M is to take it equal $\tilde{L}\tilde{U}$.

Recall that the coefficient matrices appearing in the systems of interest are irreducible M-matrices. It has been shown that incomplete LU decompositions exist for such matrices (in exact arithmetic) and that they are even more stable than the complete LU decomposition without partial pivoting (see [20] p. 152).

Theorem 2.6 *If A is an M-matrix, then the construction of an incomplete LU decomposition is at least as stable as the construction of a complete decomposition without pivoting.*

Three types of incomplete LU factorizations are considered. The first imposes on the computed preconditioner the same nonzero structure as the original matrix and is called $ILU(0)$. The second is called $ILUTH$ and is a threshold-based approach. The third forces the computed factors to have at most a prespecified fixed number of nonzero elements per row and is called $ILUK$.

The ILU(0) Incomplete Factorization

The idea of $ILU(0)$ is to drop all fill-in elements which occur during the LU decomposition (recall that a fill-in element refers to a nonzero element introduced in the matrix which holds the LU factors in a location where there was initially a zero element in the original matrix). Thus, if we denote by $NZ(A)$ the nonzero structure of A, i.e., the set of all pairs (i, j) such that $a_{ij} \neq 0$ then $ILU(0)$ can be described as follows:

Algorithm: $ILU(0)$

for $i = 2, \ldots, n$
    for $k = 1, \ldots, i-1$ and for $(i, k) \in NZ(A)$
        compute $a_{ik} = a_{ik}/a_{kk}$
        for $j = k+1, \ldots, n$ and for $(i, j) \in NZ(A)$
            compute $a_{ij} := a_{ij} - a_{ik}a_{kj}$
        end
    end
end

The ILUTH Incomplete Factorization

In $ILUTH$, the decomposition takes place in a row-by-row manner. The dropping rule of this preconditioning technique is to zero out all elements having an absolute value less than a prespecified threshold. The only exception is that the dropping rule does not apply for the diagonal elements which are kept no matter how small they become. The dropping rule is applied just after the multipliers are formed, once, and applied one more time right after the reduction of a row is over. We experimented with two different threshold values, 10^{-2} and 10^{-3}, for each Krylov subspace solver. In the $ILUTH$ algorithm provided, a_{i*} denotes the ith row of A.

Algorithm: $ILUTH$

for $i = 1, \ldots, n$
  $\omega = a_{i*}$
  for $k = 1, \ldots, i-1$ and when $\omega_k \neq 0$
    $\omega_k := \omega_k / a_{kk}$
    Apply the dropping rule to ω_k
    if $\omega_k \neq 0$
      $\omega := \omega - \omega_k * u_{k*}$
    endif
  end
  Apply the dropping rule to row ω
  $l_{i,j} := \omega_j$ for $j = 1, \ldots, i-1$
  $u_{i,j} := \omega_j$ for $j = i+1, \ldots, n$
  $\omega := 0$
end

The ILUK Incomplete Factorization

The last type of incomplete factorization that we consider is based on the idea of keeping at most a prespecified fixed number of nonzero elements in each reduced row. This method enables the user to control the number of fill-ins allowed. Therefore, it is suitable in case there is only a fixed amount of memory available to store the incomplete factors $\tilde{L}$ and $\tilde{U}$. Each time a row has been reduced, a search is conducted to find the K largest elements in absolute value. All other elements in the row are annihilated. As for $ILUTH$, the diagonal elements are preserved regardless of their magnitude. Again, in the algorithm we use a_{i*} to denote the ith row of A.

Algorithm: *ILUK*

for $i = 1, \ldots, n$
  $\omega = a_{i*}$
  for $k = 1, \ldots, i-1$ and when $\omega_k \neq 0$
    $\omega_k := \omega_k / a_{kk}$
    $\omega := \omega - \omega_k * u_{k*}$
  end
  Apply the dropping rule to row ω
  $l_{i,j} := \omega_j$ for $j = 1, \ldots, i-1$
  $u_{i,j} := \omega_j$ for $j = i+1, \ldots, n$
  $\omega := 0$
end

Finally, we should stress that not much work has been done in studying what constitutes a good incomplete factorization for Markov chain models. The concept is still in its infancy in this domain and further studies are still needed.

2.3 Implementation Considerations

In this section we focus on various aspects that should be taken into account during implementation phase. As we are dealing with large sparse systems[5], the first thing we should be thinking of is to find an efficient storage scheme. One popular scheme is the one we used, which is known as the *compact sparse row (CSR)* format [26, 36]. This format uses a real (double-precision) one-dimensional array aa and two integer arrays ja and ia. Array aa is of size nz (the number of nonzero elements in the matrix) and is used to hold the nonzero elements of the matrix. The elements are stored by rows in such a way that elements of row i come before those of row $i+1$, but the elements within a row need not be in order. Array ja holds the column position of each element, i.e., $ja(k)$ gives the column position of the element stored in the kth position of aa ($aa(k)$). Array ja is also of size nz. Array ia is a pointer array whose

[5]The average order of the problems we experimented with is 33,278; the largest matrix is of order 104,625.

lth element indicates the position in aa and ja at which the elements of the lth row begin. The size of ia is $n+1$, where n is the order of the matrix and $ia(n+1) = nz+1$. In this way, $ia(l+1) - ia(l)$ always gives the number of nonzero elements in row l, $\quad l = 1, 2, \ldots, n$.

To illustrate the use of the CSR format, consider the following matrix

$$A = \begin{pmatrix} -2.1 & 0.0 & 1.7 & 0.4 \\ 0.8 & -0.8 & 0.0 & 0.0 \\ 0.2 & 1.5 & -1.7 & 0.0 \\ 0.0 & 0.3 & 0.2 & -0.5 \end{pmatrix}.$$

Then one possible way to store this matrix in the CSR format is

Real array	aa :	-2.1	1.7	0.4	-0.8	0.8	-1.7	0.2	1.5	-0.5	0.3	0.2
Column array	ja :	1	3	4	2	1	3	1	2	4	2	3
Pointer array	ia :	1	4	6	9	12						

In addition to its efficiency, the CSR format can be easily used for implementing matrix-vector operations. This property makes it very adequate for iterative methods.

Unless otherwise specified, by reductions we mean row-reductions. This strategy is used to take full advantage of the row-by-row storage of the CSR format.

The code is implemented in Fortran 77, and as it only supports static memory allocation, two one-dimensional arrays are defined at the beginning of the program to hold double precision and integer values. The dimensions of these arrays are set to $3,500,000$ for the double precision array and $2,100,000$ for the integer array.

The initial approximation $x^{(0)}$ is always chosen to have a uniform distribution, i.e., $x_i^{(0)} = 1/n$, $i = 1, 2, \ldots, n$, where n is the vector length.

In order to regulate the amount of fill-in produced, $ILUTH$ is implemented in such a way that before the reduction of any row, the number of free entries in the double precision work array is divided by the number of rows still to be reduced. This gives us the maximum number of nonzero elements that we allow to be stored for that row. If the reduction leads to a number of nonzero elements higher than the precomputed one, the threshold is multiplied by 10 and the dropping rule is applied again. This is repeated until the number of nonzero elements in a given row becomes less than or equal to the maximum number allowed. The first row of the matrix is not reduced, and the method is forced to fail if the magnitude of any reduced diagonal element is less than 10^{-300}.

The way $ILUK$ is implemented is to compute the Kth largest value, in magnitude, say max, in the reduced row. Then, all the elements having an absolute value less than max are set to 0. If the number of nonzero elements in the row is still higher than K, the reduced row is scanned from left to right zero out any element having an absolute value equal to max until the number of nonzero elements decreases to K. As in $ILUTH$, the reduction does not include the first row of the matrix and the method fails if any reduced diagonal element is found to be less than 10^{-300}.

In block iterative methods we have attempted at solving diagonal blocks, and the coupling matrix in IAD, directly by Gaussian elimination. The memory needed to solve the coupling matrix is set aside at the beginning and what is left is used for the diagonal blocks. If there is not enough space for solving the coupling matrix, the method fails. Blocks of order 1 and 2 are treated separately. In two-stage solvers (block iterative methods), we obtain the LU decompositions of as many diagonal blocks as possible given available memory and do this in such a way that smaller blocks are treated first, leaving the big blocks to be solved using point SOR when there is insufficient memory. In order to accelerate this process we use a considerably large tolerance 10^{-3} with the point SOR algorithm when solving diagonal blocks. The SOR algorithm is always used with the optimal relaxation parameter ω (up to one digit after the decimal point). This is achieved by repeating the same experiments with different values of ω.

As an attempt to minimize the probability of underflow and overflow, each row of the coefficient matrix is multiplied by the inverse of the largest value in magnitude in that row (the absolute value of the diagonal element). This is called a scaling operation and it just retransforms the system to a more suitable form without altering the global solution. Another way to limit the effect of underflow and overflow and to control the irregular convergence behavior of some iterative methods up to a certain extent, is to normalize the solution vector at each iteration. The drawback of this strategy is that it may lead to considerable loss of precision due to roundings that occur at each iteration. In block iterative methods and point SOR (which are part of MARCA) the coefficient matrix is scaled and the solution vector is normalized at each iteration. Nevertheless, in the projection methods we implemented, the coefficient matrix is not scaled and the solution vector is only normalized at the termination of the algorithm.

Chapter 3

Models Used

In this chapter we will discuss the problems we used in our numerical studies. Seven models are considered, six of which appear in [34] and one is discussed in [2, 5]. All seven models rise from Markov chain applications. Three of these are chosen and two ill-conditioned test cases from each one are generated giving us a total of thirteen test cases. From these thirteen cases, twenty seven sparse [1] test matrices are obtained with which to experiment. A few of the test matrices have symmetric nonzero structures (Appendix B shows the nonzero structures of the transposed infinitesimal generator matrices, which are formed from the stochastic test matrices). The majority of the test matrices would be ranked among the largest of the matrices considered in the Matrix Market [19].

3.1 Complete Buffer Sharing With Pushout Thresholds in ATM Networks

Broadband Integrated Services Digital Networks (B-ISDNs) are to support multiple types of traffic such as voice, video, and data. The Asynchronous Transfer Mode (ATM) is the support technique of choice for B-ISDNs by the standards committees. In this mode of operation, all information is carried using fixed size packets (called 'cell's) so as to share the network among multiple classes of traffic. Since multiclass traffic will be carried on B-ISDNs, different quality of service requirements will be imposed by different applications.

One type of congestion control for ATM networks deals with discarding cells at ATM buffers in order to guarantee a prespecified cell loss rate. One bit in each ATM header is reserved to assign the space priorities of cells. This bit indicates whether the given cell is high or low priority. Priority cell discarding is a buffer management scheme in which higher priority cells are favored in receiving buffer space. An efficient technique for determining the cells to be discarded when congestion occurs is the complete buffer sharing scheme with pushout thresholds.

In the system under consideration, there are two classes of traffic arriving to an ATM buffer of size K. Time is divided into fixed size slots of length equal to one cell transmission time. The arrival of traffic class l $(= 1, 2)$ to the buffer is modelled as a Bernoulli process with probability of cell arrival p_l in a slot.

The states of the corresponding queueing system may be represented by the ordered pair (i, j), where i and j are the number of class 1 and class 2 cells in the buffer, respectively [2, 5]. Let k $(= i+j)$ denote the total number of cells in the buffer at state (i, j). Then, a natural state space ordering that places the states with the same number of total cells in the buffer (i.e., k) consecutively, gives rise to a block matrix with $\sum_{k=0}^{K}(k+1) = (K+1)(K+2)/2$ states. The first block consists of the state (0,0) (i.e., the state in which the buffer is empty), the second block has states (0,1), (1,0), the third block has states (0,2),(1,1),(2,0), and so on. The kth block has $k+1$ states. That is, we have the following ordering:

$$(0,0) \prec (0,1) \prec (1,0) \prec (0,2) \prec (1,1) \prec (2,0) \prec \cdots \prec (K,0)$$

During a time slot, no cells, one cell, or two cells may arrive. If one or two cells arrive, then this happens at the beginning of a slot. A cell departure occurs by the end of the slot if the buffer has at least one cell at the beginning of the slot. Hence, an arriving cell cannot be transmitted before the end of the next slot. With these assumptions, a cell is discarded if and only if two cells arrive to a full buffer. The pushout threshold for class 2 cells is given by T_2 and the pushout threshold of class 1 cells is given by T_1 $(= K - T_2)$. If two cells arrive to a full buffer (i.e., $i + j = K$), then a class 2 cell is discarded if $j > T_2$, otherwise a class 1 cell is discarded if $j < T_2$. When $j = T_2$, the lower priority traffic class cell is discarded. One may view the system as if there is temporary space to store up to two arrivals while the buffer is full and a decision as to which class of cell will be discarded is made.

It is assumed that at steady-state the head of the queue (i.e., the cell that will be leaving the buffer at the end of the current time slot—if there was one to begin with) is a type 1 cell with probability $i/(i + j)$ and it is a type 2 cell with probability $j/(i + j)$.

The DTMC corresponding to these assumptions is block tridiagonal (with the exception of the first row of blocks) where each diagonal block is tridiagonal and has a different block size. Depending on the selected threshold, the nonzero elements in the last row of blocks change making it very difficult to apply analytical solution techniques to such a system with control.

To study the effect of the threshold, three test cases are generated. In all these test cases, K and T_2 are fixed to 200 and 20, respectively. In the first test case, which we call *easy*, we set $p_1 = 0.99$ and $p_2 = 0.99$. The second test case, *medium*, is more ill-conditioned and is generated by choosing $p_1 = 0.1$ and $p_2 = 0.5$. Setting p_1 and p_2 respectively to 0.1 and 0.9 gives us a third even more ill-conditioned test case which we call *hard*. The coefficient matrices of the three test cases are of the same order $n = 20,301$ and have the same number of nonzero elements $nz = 140,504$, bandwidth, and nonzero structure. Tables 3.2, 3.3 and 3.4 show the results of ncd test, equal and other partitionings (discussed in section 2.2.2) applied to the three coefficient matrices. γ is the decomposability parameter used in ncd test. equal and other partitionings are identical in the three cases since the matrices are of the same order. Based on the results of ncd test we decided to experiment with two permuted versions of *easy*: with $\gamma = 10^{-4}$ we obtained the *easy_gm4* test matrix and with $\gamma = 10^{-3}$ we obtained *easy_gm3*. As can be seen from Table 3.2, choosing $\gamma = 10^{-2}$ causes the coefficient matrix to be partitioned into blocks of size 1 with the exception of one block which is of size 2. Such cases are not interesting and we do not consider them in our experiments. Besides, round-off errors due to input/output operations caused some elements of the permuted stochastic matrices of *medium* and *hard* read from input files, to differ from their values computed on computer. This prevented us from experimenting with the permuted versions of *medium* and *hard*. Tables 3.1 and 3.5 give information about the symmetric nonzero structure status and the bandwidth[1] of the five test matrices. Since *easy*, *medium*, and *hard* test matrices have the same nonzero structure, their results are given in the first row as *pushout threshold*.

n	nz	symmetric nz structure
20,301	140,504	no

Table 3.1: Characteristics of the Pushout Threshold Problem.

γ	number of blocks	smallest block size	largest block size
10^{-4}	308	1	5,050
10^{-3}	4,060	1	162
10^{-2}	20,300	1	2

	number of blocks	last block size
equal	143	137
other	201	0

Table 3.2: Partitioning Results for the *easy* Test Case.

γ	number of blocks	smallest block size	largest block size
10^{-3}	4	1	20,295
10^{-2}	720	1	19,477
10^{-1}	20,300	1	2

Table 3.3: Partitioning Results for the *medium* Test Case.

[1]We adopt the convention that higher and lower bandwidths do not include the diagonal.

γ	number of blocks	smallest block size	largest block size
10^{-4}	2	1	20,300
10^{-3}	134	1	20,168
10^{-2}	2,286	1	18,016

Table 3.4: Partitioning Results for the *hard* Test Case.

matrix	lower bandwidth	higher bandwidth
pushout threshold	201	201
easy_gm4	20,103	20,103
easy_gm3	381	381

Table 3.5: Lower and Higher Bandwidths of the Pushout Threshold Test Matrices.

3.2 A Two-Dimensional Markov Chain Model

As the title suggests, a two-dimensional Markov chain is considered in this problem. In the first dimension of the chain, the state variable assumes all values from 0 through N_x. Similarly, in the second dimension, the state variable takes on values from 0 through N_y. The state space is sketched in Figure 3.1.

This two-dimensional Markov chain model allows for transitions from any non-boundary state to adjacent states in the North, South, East, West, North-East, North-West, South-East and South-West directions. However, in the model we used in our experiments, only transitions to the South, East and North-West are permitted (taking the others to be 0). From any non-boundary state (u, v), transitions to the South are assigned the value v, transitions to the East are assigned the value 2025.0, and transitions to the North-West are assigned the value u [34]. The state space of the Markov chain is of size $(N_x+1)(N_y+1)$. The values of N_x and N_y are both set to 128, yielding a matrix (*2D*) of order $n = 16,641$ and number of nonzero elements $nz = 66,049$. The partitioning results of the *2D* test matrix are illustrated in Table 3.7. These results suggested experimenting with two more test matrices, *2D_gm3* and *2D_gm2*, formed by permuting the *2D* matrix using decomposability parameters $\gamma = 10^{-3}$ and $\gamma = 10^{-2}$, respectively. The characteristics of the three test matrices and their bandwidths are reported in Tables 3.6 and 3.8.

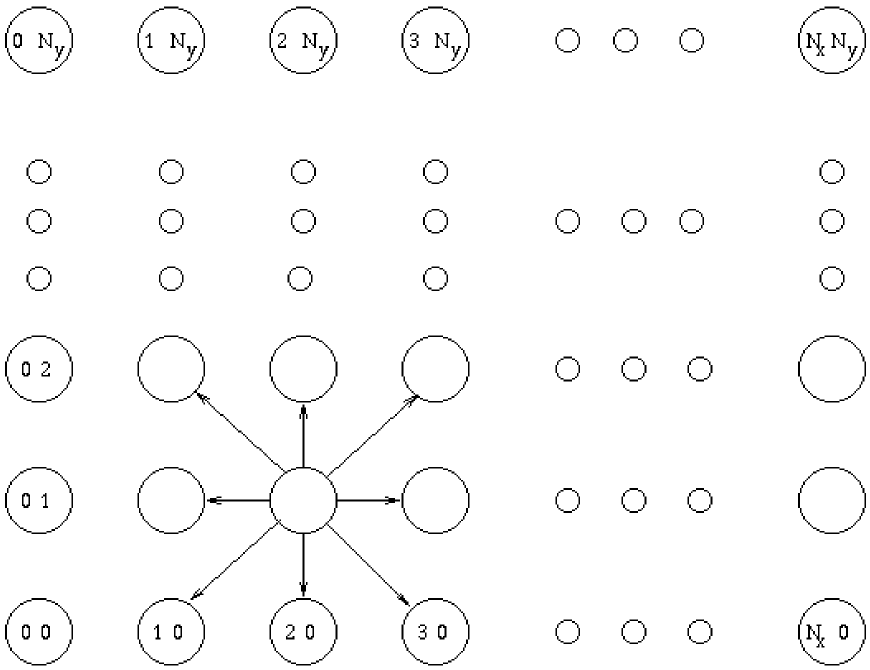

Figure 3.1: A Two-Dimensional Markov Chain Model Model.

n	nz	symmetric nz structure
16,641	66,049	no

Table 3.6: Characteristics of the Two-Dimensional Markov Chain Problem.

3.3 An NCD Queueing Network of the Central Server Type

The model illustrated in Figure 3.2 represents the system architecture of a time-shared, multiprogrammed, paged, virtual memory computer. The system [34] consists of

γ	number of blocks	smallest block size	largest block size
10^{-3}	513	1	16,129
10^{-2}	5,913	1	11,449

	number of blocks	last block size
equal	129	0
other	182	170

Table 3.7: Partitioning Results for the Two-Dimensional Markov Chain Problem.

matrix	lower bandwidth	higher bandwidth
2D	65	129
2D_gm3	16,504	16,633
2D_gm2	13,404	13,533

Table 3.8: Lower and Higher Bandwidths of the Two-Dimensional Markov Chain Test Matrices.

- a set of N_t terminals from which N_t users generate commands,
- a control 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 user generates a command at the terminal, it remains inactive until the system responds. Symbolically, this user enters the CPU queue. The system behaves in such a way that after a certain time period, called the compute time, either a page fault or an input/output (file request) occurs. In the case of a page fault, the process currently in the system enters the SM queue, otherwise, in the case of a file request, it joins the FD queue. Processes that terminate their service at the

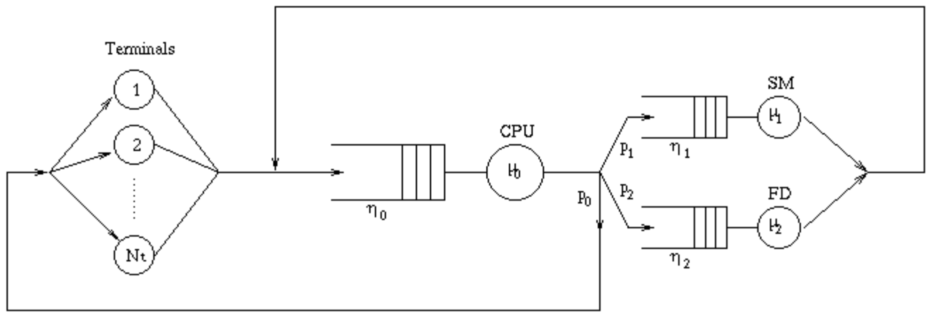

Figure 3.2: An NCD Queueing Network of the Central Server Type Model.

SM or FD queue return to the CPU queue. A command commit is symbolically represented by a departure of the process from the CPU to the terminals.

Let n_0, n_1 and n_2 respectively be the number of processes in the CPU, SM and FD queues at a certain time. Then the degree of multiprogramming at that moment is given by $\eta = n_0 + n_1 + n_2$. Let $(\mu_0(\eta))^{-1}$, $q(\eta)$ and $r(\eta)$ respectively denote the mean service time at the CPU, the mean compute time between two page faults, and the mean compute time between two i/o requests. It follows that the probabilities that a process leaving the CPU will be directed to the SM or to the FD queue are, respectively, given by $p_1(\eta) = (\mu_0(\eta)q(\eta))^{-1}$ and $p_2(\eta) = (\mu_0(\eta)r(\eta))^{-1}$. The probability that a process leaving the CPU to the terminals is given by $p_0(\eta) = (\mu_0(\eta)c(\eta))^{-1} = 1 - (p_1(\eta) + p_2(\eta))$, where $c(\eta)$ is the mean compute time of a process [24].

For experimental purposes, we assigned a specific value for each parameter. The rate at which processes move from the CPU queue to the SM device is taken to be $p_1(\eta)\mu_0(\eta) = 100(\eta/128)^{1.5}$. The mean compute time between two i/o requests $r(\eta)$ is taken as 20 ms so that $p_2(\eta)\mu_0(\eta) = 0.05$, and the mean compute time of a process $c(\eta)$ is equal to 500 ms giving $p_0(\eta)\mu_0(\eta) = 0.002$. The mean think-time of a user at a terminal is estimated to be on the order of $\lambda^{-1} = 10\ s$. The mean service time of the SM is taken as $(\mu_1(\eta))^{-1} = 5\ ms$ and that of the FD to be $(\mu_2(\eta))^{-1} = 30\ ms$. The Markov chain state space is of size $\binom{N_t+4-1}{4-1}$. The total number of users in the system (N_t) was set to 50 yielding a matrix (ncd) of order $n = 23,426$ and number of nonzero elements $nz = 156,026$.

n	nz	symmetric nz structure
23,426	256,026	yes

Table 3.9: Characteristics of the NCD Queueing Network Problem.

γ	number of blocks	smallest block size	largest block size
10^{-5}	3	1,275	20,825
10^{-4}	51	1	1,326
10^{-3}	51	1	1,326

	number of blocks	last block size
equal	154	17
other	216	206

Table 3.10: Partitioning Results for the *ncd* Test Case.

Two more ill-conditioned test cases are generated from this model. The first one (*ncd_alt1*) is obtained by setting the mean service time of the FD to $(\mu_2(\eta))^{-1} = 3,000\ s$. The second test case (*ncd_alt2*) is even more ill-conditioned than *ncd_alt1* and is generated by making the mean think-time of a user at a terminal $\lambda^{-1} = 10,000\ s$. Naturally, the three test matrices, *ncd*, *ncd_alt1*, and *ncd_alt2*, have the same order, number of nonzero elements, and nonzero structure. We also experimented with *ncd_gm4* (a permuted version of *ncd* obtained by choosing 10^{-4} as the value of γ in the ncd test of *ncd*), *ncd_alt1_gm7*, *ncd_alt1_gm5* and *ncd_alt1_gm4* (three permuted versions of *ncd_alt1* respectively by assigning 10^{-7}, 10^{-5} and 10^{-4} to γ in the ncd test of *ncd_alt1*), and *ncd_alt2_gm7* and *ncd_alt2_gm6* (two permuted versions of *ncd_alt2* obtained respectively by assigning 10^{-7} and 10^{-6} to γ in the ncd test of *ncd_alt2*). The characteristics, partitioning results, and bandwidths of all the test matrices for this model are reported in Tables 3.9, 3.10, 3.11, 3.12, and 3.13. Test matrices *ncd_gm5* and *ncd_alt2_gm8* are not interesting because of their block structures (see Tables 3.10 and 3.12).

γ	number of blocks	smallest block size	largest block size
10^{-7}	51	1	1,326
10^{-6}	51	1	1,326
10^{-5}	150	1	1,225
10^{-4}	1,326	1	51
10^{-3}	1,326	1	51

Table 3.11: Partitioning Results for the *ncd_alt1* Test Case.

γ	number of blocks	smallest block size	largest block size
10^{-8}	3	1,275	20,825
10^{-7}	25	406	3,654
10^{-6}	51	1	1,326
10^{-5}	51	1	1,326
10^{-4}	51	1	1,326
10^{-3}	51	1	1,326

Table 3.12: Partitioning Results for the *ncd_alt2* Test Case.

3.4 A Telecommunication Model

A telecommunication problem is modeled to study the effect of impatient telephone customers on a computerized telephone exchange [34]. The model is shown in Figure 3.3. In this model each customer makes a request for service. Then the customer has to wait a certain period for a reply. If the reply has not arrived at the end of that period, the customer has the right to either give up and leave the network, or wait for some period of time before trying again.

All customers have to pass by station $S2$ which is dedicated to a special processing task. These customers are processed by a single server according to a processor sharing discipline. Each customer may wait in $S2$ for a certain time which is defined as an upper bound on its service duration: whenever its patience is exhausted, the customer simply gives up processing (with a fixed probability $1 - h$).

matrix	lower bandwidth	higher bandwidth
ncd, ncd_alt1, ncd_alt2	460	460
ncd_gm4	1,326	1,326
ncd_alt1_gm7	1,375	1,375
ncd_alt1_gm5	1,374	1,374
ncd_alt1_gm4	701	701
ncd_alt2_gm7	2,980	2,980
ncd_alt2_gm6	1,325	1,325

Table 3.13: Lower and Higher Bandwidths of the NCD Queueing Network Test Matrices.

In case the customer decides to keep trying, it joins an infinite server station $S1$ where it remains for a certain period, called the thinking-time, before joining back station $S2$ for another attempt.

We are interested in studying the number of customers in $S1$ and $S2$ in the long run. Let i and j be the number of customers in $S1$ and $S2$, respectively. Then a state of the network may be described by the pair (i, j). When $j \geq 1$, the rate of

- service completions in $S2$ is μ,
- departures due to impatience is $j\tau$.

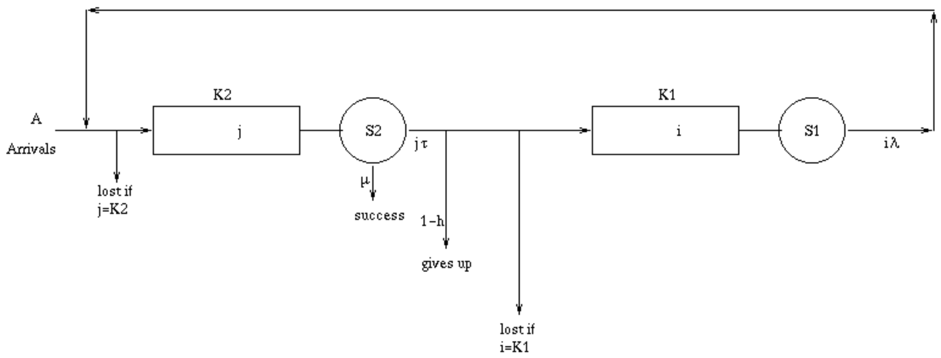

Figure 3.3: Telecommunication Model.

When $i \geq 1$, the rate of departures from $S1$ is $i\lambda$. External arrivals to $S2$ are assumed to have a Poisson distribution of rate A.

As we are interested in finite Markov chains, we let $K1$ and $K2$ be the maximum sizes of $S1$ and $S2$, respectively. Customers arriving to a full station are lost. It is important to choose large values for $K1$ and $K2$ so that the probability of saturation is negligible. In that case, the truncation of the state space will have little effect, and hence, the resulting steady-state probabilities may be taken as an accurate approximation of those of the infinite capacity network.

The following are realistic values which we took from [24] to use in our experiments:

$$A = 0.6, \quad \mu = 1.0, \quad \tau = 0.05, \quad h = 0.85, \quad \lambda = 5.0.$$

The state space of the Markov chain is of size $(K1+1)(K2+1)$. We set $K1 = 30$ and $K2 = 660$ which gave a matrix *telecom* on the order of $n = 20,491$ with $nz = 101,041$ nonzero elements. Results of ncd test, *equal* and *other* partitionings are shown in Table 3.15. Based on this ncd test, we decided to experiment with two permuted versions of the original matrix using decomposability parameters $\gamma = 10^{-2}$ to obtain the *telecom_gm2* test matrix and $\gamma = 10^{-1}$ to obtain *telecom_gm1*. It is obvious that the three test matrices; *telecom*, *telecom_gm2* and *telecom_gm1*, have the same order, number of nonzero elements, and partitioning results. Tables 3.14 and 3.16 reports the symmetric nonzero structure status and the bandwidths of the three test matrices.

n	nz	symmetric nz structure
20,491	101,041	no

Table 3.14: Characteristics of the *telecom* Problem.

γ	number of blocks	smallest block size	largest block size
10^{-2}	1,981	1	31
10^{-1}	14,389	1	28

	number of blocks	last block size
equal	144	42
other	202	190

Table 3.15: Partitioning Results for the *telecom* Problem.

matrix	lower bandwidth	higher bandwidth
telecom	31	60
telecom_gm2	18,750	18,750
telecom_gm1	9,171	9,171

Table 3.16: Lower and Higher Bandwidths of the Telecom Test Matrices.

3.5 A Queueing Network with Blocking and Priority Service Model

The model we shall discuss now is an open queueing network of three finite capacity queues and two customer classes. Class 1 customers arrive from the exterior to queue 1 according to a Poisson process λ_1. Similarly, class 2 customers arrive from outside the network to queue 2 according to a Poisson process, but this time at rate λ_2. At this stage, any customer (from either classes) is lost if upon arrival it finds the buffer full. The servers at queues 1 and 2 provide exponential service at rates μ_1 and μ_2 respectively. After being served, customers of either of these queues try to join queue 3. If queue 3 is full, class 1 customers are blocked (blocking after service) and the server at queue 1 must halt. This server cannot resume serving any other customer unless a slot becomes available in the buffer of queue 3 and the blocked customer is transferred. On the other hand, when a class 2 customer has been served at queue 2 and finds the buffer at queue 3 full, it is simply lost. Queue 3 provides exponential service at rate μ_{3_1} to class 1 customers and rate μ_{3_2} to class 2 customers. Customers departing after service at queue 3 leave the network. Figure 3.4 illustrates this model. $C_k - 1,\ k = 1, 2, 3$ denote the finite buffer capacity at queue k.

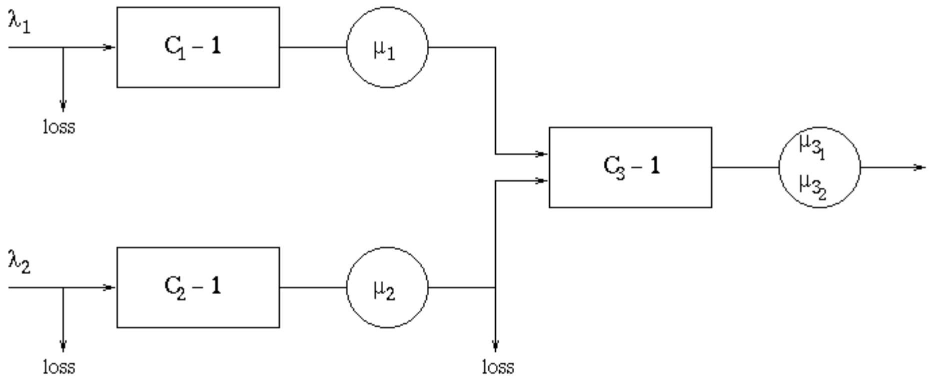

Figure 3.4: An ATM Queueing Network Model.

The states of the Markov chain underlying this model, may be represented by four-component vectors [34]. Components 1 and 2 may be used to denote the number of customers in queue 1 and 2 respectively. Components 3 and 4 may be used to represent the number of class 1 and class 2 customers, respectively, present in queue 3.

We assigned the following values to the parameters indicated on Figure 3.4:

$$\lambda_1 = 1.0, \quad \lambda_2 = 2.0, \quad \mu_1 = 3.0, \quad \mu_2 = 4.0, \quad \mu_{3_1} = 5.0, \quad \mu_{3_2} = 6.0.$$

The state space is of size $C_1 C_2 C_3 (C_3 + 1)/2$, so setting each of C_1 and C_2 to 15 and C_3 to 30 leads to a matrix (*qnatm*) of order $n = 104,625$ and number of nonzero elements $nz = 593,115$. Out of this qnatm matrix, another test matrix (*qnatm_gm1*) was formed by permuting qnatm using a decomposability parameter $\gamma = 10^{-1}$. Additional information about the two matrices are illustrated in Tables 3.17, 3.18, and 3.19.

n	nz	symmetric nz structure
104,625	593,115	no

Table 3.17: Characteristics of the ATM Queueing Network Problem.

γ	number of blocks	smallest block size	largest block size
10^{-1}	91,800	1	450

	number of blocks	last block size
equal	324	296
other	457	429

Table 3.18: Partitioning Results for the ATM Queueing Network Problem.

3.6 A Multiplexing Model of a Leaky Bucket in Tandem

One of the major problems in ATM networks is to control the congestion of intermediate buffers with fast and simple mechanisms. Several policies were proposed and evaluated with diverse probabilistic hypothesis. The simplest mechanism is the leaky bucket [34]. The problem is to determine the behavior of this mechanism under external arrivals. The external arrival stream is modeled as a Poisson process. An evaluation of this mechanism will allow it to be compared with other more complex mechanisms.

matrix	lower bandwidth	higher bandwidth
qnatm	2,728	5,385
qnatm_gm1	6,974	6,975

Table 3.19: Lower and Higher Bandwidths of the ATM Queueing Network Test Matrices.

The traffic source is of an $M/D/1/C$ type. This queue is of size C cells and has a single server with service time D (which will be taken as a unit time). The arrivals that are modeled as a Poisson process have rate λ. Then the model may be viewed as a 1-dimensional discrete time Markov chain (with time unit D) with state descriptor N_p which is the number of cells produced by the Poisson source at time t,

The leaky bucket has a finite size of K cells and a service time $T'D = TD(1-\epsilon)$, where T' is an integer. The state of the system is described by the state variable k which is the buffer occupancy (in terms of the number of cells).

The values used for the described parameters are

$$C = K = 64, \quad T = 4, \quad \lambda = 0.85, \quad \epsilon = 0.4959.$$

The *leaky* matrix we generated from this model has order $n = C \times K \times T(1-\epsilon) = 8,258$ and number of nonzero elements $nz = 197,474$. This matrix is severely ill-conditioned (i.e., NCD up to machine precision). We could not experiment with any of the permuted versions of *leaky* due to the round-off error problem mentioned in section 3.1. Some properties of the matrix are provided in Tables 3.20, 3.21, and 3.22.

n	nz	symmetric nz structure
8,258	197,474	no

Table 3.20: Characteristics of the Leaky-Bucket Problem.

3.7 Mutex—A Resource Sharing Model

In this model, M distinguishable processes share a certain resource [34]. Each of these processes alternates between a *sleeping* state and a resource *using* state. However, only P processes may concurrently use the resource, where $1 \leq P \leq M$. If a process currently in the sleeping state tries to move to the

γ	number of blocks	smallest block size	largest block size
10^{-9}	193	1	8,057
10^{-8}	225	1	8,016
10^{-7}	225	1	8,016
10^{-6}	265	1	7,976
10^{-5}	318	1	7,923
10^{-4}	400	1	7,832
10^{-3}	531	1	7,692
10^{-2}	778	1	7,427
10^{-1}	7,386	1	507

	number of blocks	last block size
equal	91	158
other	129	2

Table 3.21: Partitioning Results for the Leaky-Bucket Problem.

matrix	lower bandwidth	higher bandwidth
leaky	191	435

Table 3.22: Lower and Higher Bandwidths of the *leaky* Test Matrix.

resource using state while there are P processes already using the resource, it simply fails to access the resource and remains in the sleeping state. Notice that when $P = 1$ this model reduces to the usual mutual exclusion problem, whereas when $P = N$ all the processes are independent. Let λ_i be the rate at which process i awakes from the sleeping state wishing to access the resource and let μ_i be the rate at which this same process releases the resource when it has a possession of it. Figure 3.5 provides a graphical illustration of this model. Each process i is modelled by a two-state automaton A_i. The function f takes the value 1 when access is permitted to the resource and takes the value 0 otherwise.

To experiment with this model, we set $\lambda_i = 1/i$ and $\mu_i = i$, for $i = 1, 2, \ldots, M$. Parameters P and M were fixed to 8 and 16, respectively. These

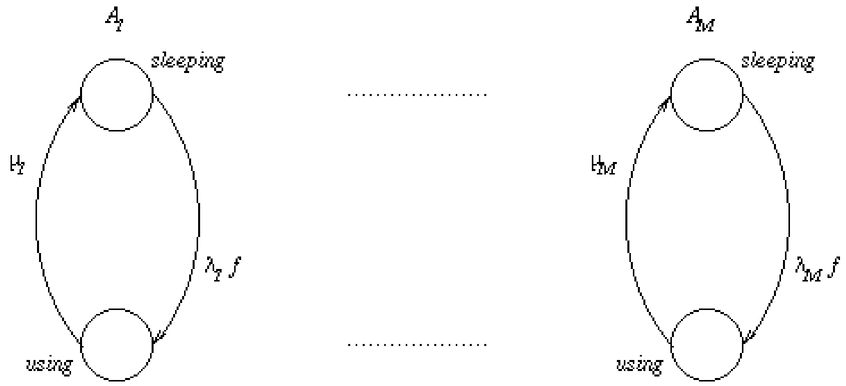

Figure 3.5: A Resource Sharing Model (Mutex).

values lead to a matrix ($mutex$) of order $n = \sum_{p=0}^{P} \binom{M}{p} = 39,203$ and number of nonzero elements $nz = 563,491$.

n	nz	symmetric nz structure
39,203	563,491	yes

Table 3.23: Characteristics of the Mutex Problem.

Two more ill-conditioned test cases are generated from this model. The first one ($mutex\_alt1$) is obtained by setting $\mu_i = 10^3\, i$. The second test case ($mutex\_alt2$) is even more ill-conditioned than the first and is generated by setting $\lambda_i = 10^{-3}/i$ and $\mu_i = 10^3\, i$. As the values of P and M are fixed, the three test cases have the same order, number of nonzero elements, and nonzero structure. The partitioning results of these matrices are shown in Tables 3.24, 3.25 and 3.26. Permuting *mutex_alt1* with decomposability parameter $\gamma = 10^{-6}$ led to an additional test matrix ($mutex\_alt1\_gm6$) which we also used in our experiments. However, we could not experiment with the permuted versions of *mutex* and *mutex_alt2* respectively with decomposability parameters $\gamma = 10^{-3}$ and $\gamma = 10^{-9}$ due to the i/o round-off error problem discussed in section 3.1. The nonzero structure symmetry information and the bandwidths of all the test matrices generated from this model are provided in Tables 3.23 and 3.27.

γ	number of blocks	smallest block size	largest block size
10^{-3}	256	1	256

	number of blocks	last block size
equal	198	394
other	280	143

Table 3.24: Partitioning Results for the *mutex* Test Matrix.

γ	number of blocks	smallest block size	largest block size
10^{-6}	256	1	256

Table 3.25: Partitioning Results for the *mutex_alt1* Test Matrix.

γ	number of blocks	smallest block size	largest block size
10^{-9}	256	1	256

Table 3.26: Partitioning Results for the *mutex_alt2* Test Matrix.

matrix	lower bandwidth	higher bandwidth
mutex,*mutex_alt1*,*mutex_alt2*	13,495	13,495
mutex_alt1_gm6	23,049	23,049

Table 3.27: Lower and Higher Bandwidths of the Mutex Problem Test Matrices.

Chapter 4

Numerical Results

In this chapter we report and interpret the numerical results of the different solvers employed on the 27 test cases generated. All the results are summarized in tables in Appendix A. We generated and stored all the test matrices in the Harwell-Boeing format using MARCA. All code is written in Fortran and compiled in double precision with $g77$ on a SUN Sparcstation running Solaris 2.5. The numerical experiments are timed using a C function that reports CPU time.

For each problem solved, the true residual and the relative backward error in the solution are computed. The true residual is computed as $\|A\hat{x}\|_\infty$, where $\hat{x}$ is the normalized approximate solution upon termination. The relative backward error [16] is computed as $\|A\hat{x}\|_\infty/(\|A\|_\infty\|\hat{x}\|_\infty)$. Table 4.1 provides the notation used in the tables of results. If a method terminates at iteration k, then the column heading $\|r\|$ denotes $\|r^{(k)}\|_2$ for GMRES and $\|r^{(k)}\|_\infty$ for the other projection methods. These norms are incurred by the respective methods and need not be computed separately. Due to this, we compare $\|r^{(k)}\|_2$ (and not $\|r^{(k)}\|_\infty$) with the stopping tolerance 10^{-10} at each (inner) iteration of GMRES. At the end of each restart the true residual is calculated explicitly from the current approximation (yet unnormalized) and then compared with the tolerance. If BCGStab converges due to the convergence test $\|s\|_\infty \leq 10^{-10}$ (see BCGStab algorithm in Chapter 2), then $\|r\|$ stands for $\|s\|_\infty$ upon termination. In this case a superscript "s" (i.e., ^{s}) is inserted in the corresponding

n	Order of the coefficient matrix
nz	Number of nonzero elements in the matrix
nzlu	Number of nonzero elements after the incomplete LU factorization
ω	Optimal relaxation parameter for point and block SOR
γ	Decomposability parameter
Time	Time (in seconds) taken by the method or the preconditioner, but not both
MFlops	Number of mega floating point operations
# it	Number of iterations performed
$\|r\|$	Infinity norm of the residual incurred by the method upon termination (exception for GMRES and BCGStab)
$\|Ax\|$	True residual upon termination
Bk. Err.	Relative backward error upon termination
$\|\Delta x\|$	Inifinity norm of the last two successive iterates (exception for point SOR)
# Bl.	Number of diagonal blocks solved iteratively (including blocks of size 1 and 2)
Partition.	Partitioning technique used

Table 4.1: Notation Used in the Tables of Results.

cell. The column heading $\|\Delta x\|$ denotes the infinity norm of the difference between the last two approximations except for point SOR in which it is reported every 10 iterations, and hence represents $\|x^{(p)} - x^{(p-1)}\|_\infty$, where p is the greatest multiple of 10 less than or equal to the number of iterations taken by point SOR upon termination. In other words, if point SOR terminates in l iterations then $p = \max\{10i, \text{ where } 10i \leq l \text{ and } i \in \mathbb{N}\}$. An asterisk (i.e., *) following the iteration number means that the method failed to converge in at most that many number of iterations. The results of QMR 1 are reported only for *mutex_alt1* test matrix as it is the only matrix for which it converged with (at least) one of the preconditioners used.

The dimension of the Krylov subspace we used for GMRES is 20 (i.e., $m = 20$). With each projection method, we used two different thresholds for the $ILUTH$ preconditioner: 10^{-2} and 10^{-3}. In $ILUK$, we allowed a maximum of 10 nonzero elements per row of the preconditioned matrix (i.e., $K = 10$). The time taken by partitioning the matrices did not exceed 1 second except for test matrices generated from the *qnatm*, *mutex*, and *leaky* problems. Partitioning *qnatm* and *qnatm_gm1* test matrices took 5.1 and 3.7 seconds respectively.

The times taken to partition *mutex_alt1* and *mutex_alt1_gm6* were 5.1 and 3.4 seconds, respectively. Partitioning *leaky* using ncd test with $\gamma = 10^{-1}$ and $\gamma = 10^{-9}$ each took 2.4 seconds.

The first thing we notice in the results of the *easy* test case is that IAD failed with all the partitionings used due to a reducible coupling matrix (see Table A.1). However, block SOR performed rather well especially with ncd test $\gamma = 10^{-3}$ and *other* partitionings. This may be explained by examining the nonzero structure of the matrix (see Appendix B). The *easy* matrix is block tridiagonal (with the exception of the first row of blocks) where diagonal blocks are tridiagonal with increasing block sizes as we go down the matrix. This enables the *other* partitioning to gather most of the nonzero elements within a block row in the diagonal block. The ncd test with $\gamma = 10^{-3}$ gives a similar partitioning to *other* (see the nonzero structure of *easy_gm3*) and hence a close performance (see Table A.3). We just wanted to show by experimenting with ncd test $\gamma = 10^{-2}$ that a partitioning technique may lead to poor performance if it does not take full advantage of the divide-and-conquer nature of block iterative methods. For this particular partitioning all the diagonal blocks had size 1 except one which was of size 2 (see block SOR $\gamma = 10^{-2}$ in Table A.1). The results of GMRES, CGS, and BCGStab are satisfactory for this test case. Each of them converge with all the preconditioners used, however they required longer time, in most of the converging cases, than block SOR due to the preconditioning time overhead for $ILUTH$ (10^{-2}), $ILUTH$ (10^{-3}), and $ILUK$ (10). It is important to notice that CGS and BCGStab with $ILUK$ converged in only 1 iteration; 2 iterations are needed for GMRES. This suggests that the preconditioned matrix formed by $ILUK$ is quite well-conditioned.

BCG converged in only five test cases. The residual infinity norm that comes as a byproduct of BCG is observed to be unstable and oscilating too much which illustrates the irregular convergence behavior of BCG especially with NCD problems.

In the *2D* matrix, BCGStab with $ILUTH$ (10^{-3}) outperformed all the other solvers by converging in 4.9 seconds (see Table A.6). In fact, this is one of the two cases where a Krylov subspace method outperformed all block iterative methods. The second test case is *mutex_alt1_gm6*. It can be seen from Table A.6 that block SOR and IAD required a relatively long time to converge with ncd partitionings $\gamma = 10^{-3}$ and $\gamma = 10^{-2}$ though they both needed 2 iterations to converge. The ncd test with $\gamma = 10^{-3}$ partitioned the matrix to 512 diagonal blocks of size 1 and one last block of size $16,129$, and similarly ncd test with $\gamma = 10^{-2}$ partitioned the matrix to $5,192$ diagonal blocks of size 1 and one last block of size $11,449$. These partitionings do not take any advantage of the divide-and-conquer nature of block iterative methods; we can easily deduce that the 2 iterations were entirely used to solve the large block in each partitioning and this may explain the relation between the low iteration number and the long time to converge. Note that *equal* and *other* partitionings converged in less time (but took larger number of iterations) as they partitioned the matrix more uniformly. The true residual calculated after 240 iterations of GMRES with $ILU0$ before normalizing the solution vector was 0.63×10^{-10} ($< 10^{-10}$) causing the algorithm to terminate. However after normalizing the solution vector, the true residual and the relative backward error turned out to be respectively 0.26×10^{-9} and 0.20×10^{-8}, which are both larger than 10^{-10}. Periodic normalization of the approximate vector, as suggested in MARCA, may alleviate this problem.

The major inconvenience of QMR (1,2, and 3) that is observed in *2D* and most of the test cases is its convergence behavior. The residual infinity norm in QMR tends to decrease in the first few iterations but stagnates thereafter. This behavior is observed in most of the test cases with all the preconditioners used. In ill-conditioned problems the residual may not be an accurate indicator of the number of correct digits in the approximate solution (see [24] p. 1168). This may explain why there were some cases (such as *2D* with QMR 2, $ILUTH$ (10^{-3})) for which although the implicit residual norm automatically computed by the method did not indicate convergence, the true residual turned out to be less than the tolerance. This suggests that an alternative stopping criterion, such as the one used in MARCA for two-stage iterative methods, might be employed for Krylov subspace methods. However, even with block iterative methods, we noticed two cases (*ncd_gm4* and *leaky*) where the method had not converged but the true residual was less than the tolerance. See Tables A.10 and A.27 for block SOR with *other* and IAD with ncd test $\gamma = 10^{-9}$

partitionings, respectively. The alternative stopping criterion seems to be a better test for convergence but still does not solve all problems.

The *equal* partitioning of the *ncd* test matrix led to a reducible coupling matrix causing IAD to fail in this case (see Table A.9). With ncd test $\gamma = 10^{-4}$ and *other* partitionings IAD outperformed block SOR though both methods followed the same strategy in solving the diagonal blocks (i.e., same blocks are solved iteratively in both methods). This shows the advantage of IAD over block SOR in solving the coupling matrix directly, in the aggregation step, when it is not too large. In this case it was of order 51 and 216 for ncd test $\gamma = 10^{-4}$ and *other* partitionings, respectively. The *ncd* test case is one of several test cases which demonstrates the superiority of BCGStab over the other projection methods.

The block SOR method with ncd test $\gamma = 10^{-1}$ and *equal* partitionings applied to the *telecom* test matrix converged when the difference between two successive iterates is still greater than the tolerance (see Table A.18). In these two particular cases, it is the alternative stopping criterion that caused the iteration to terminate. When cross validating the solution vectors obtained we noticed that for the ncd test $\gamma = 10^{-1}$ partitioning, the approximate solution had five decimal digits in common with the solution computed by IAD $\gamma = 10^{-2}$, whereas for the *equal* partitioning there were six digits in common. In this test case, IAD turned out to be far superior to block SOR for each partitioning taking into account the time to converge, the true residual, and the relative backward error. This again shows the advantage of the aggregation step in solving the coupling matrix directly when it is of reasonable order. See especially IAD results with ncd test $\gamma = 10^{-2}$, *equal*, and *other* partitionings in Table A.18.

The *qnatm* and *mutex* were the two largest problems we considered in our study. Interestingly, for *mutex* and *mutex_alt1_gm6* point SOR outperformed all other solvers in terms of computation time (see Tables A.23 and A.25). These were the only test cases where point SOR is superior to other solvers. The coupling matrix obtained from the *qnatm* problem by the ncd test $\gamma = 10^{-1}$ partitioning, and hence that of *qnatm_gm1*, was very large (of order 91,800) causing the solution time of IAD to be unreasonably long (see Tables A.21

and A.22). Nevertheless, we still could find at least one partitioning for block SOR with a satisfactory convergence time. Investigating the performance of the projection methods on these two problems we notice GMRES, CGS, and BCGStab are very satisfactory in terms of time taken by each solver (excluding preconditioning time). However, the wide bandwidth and the size of each matrix caused the preconditioning time taken by $ILUTH$ and $ILUK$, and hence the overall solution time, to be dramatically large.

We should point out that the $ILU0$ preconditioner led to better overall solution time than all the other preconditioners in 16 test cases (out of 27 with which we experimented). $ILUTH$ (10^{-3}) was the best preconditioner in 8 cases whereas each of $ILUTH$ (10^{-2}) and $ILUK$ (10) was the best in only one case. The problem with $ILUK$ is the large time overhead to form the preconditioner. The test cases in which $ILUTH$ (10^{-3}) led to an overall solution time less than $ILU0$ are those in which the test matrices are of medium order (around $20,000$), have narrow bandwidth, and are either relatively more ill-conditioned than the generator matrix coming out of the seven problems (such as *medium*, *hard*, and *ncd_alt2*) or very sparse (such as *2D* in which the average number of nonzero elements per row is roughly 4).

The *leaky* matrix is the most ill-conditioned case in our test suite. In the ncd test partitioning with $\gamma = 10^{-9}$, the matrix was partitioned to 193 diagonal blocks; the largest block was of order $8,057$ and there were 192 blocks of very small sizes. Choosing $\gamma = 10^{-1}$ led to a partitioning of $7,386$ diagonal blocks; the largest block was of order 507 and the rest of the blocks had very small sizes. As a consequence of these two unbalanced partitionings, block SOR and IAD could not benefit from the divide-and-conquer nature of block algorithms. Hence, the time taken for solving the diagonal blocks was biased towards solving the largest block in both partitionings. Going back to aggregation in IAD, these two unbalanced partitionings made it a detrimental step rather than an accelerator for convergence. This may be the reason behind the poor performance of IAD with these two partitionings in contrast to the performance of block SOR with the same partitionings (see Table A.27). *equal* and *other* provided more balanced partitioning patterns in terms of block sizes. As a result, they gave smaller solution times than ncd test partitionings in both block methods. Besides, the aggregation step caused IAD to outperform block SOR with *equal* and *other* partitionings. It is worth pointing out that IAD with the ncd test $\gamma = 10^{-1}$ partitioning did not converge in $1,000$ iterations but upon termination the true residual and the relative backward error were both less than 10^{-10}. The alternative stopping criterion used in block iterative methods seems to be more suitable for the convergence test, however it does not solve all problems. Point SOR was not as efficient as block methods in this severely ill-conditioned problem. All incomplete LU factorizations, and hence all preconditioned Krylov subspace methods failed due to extremely small pivot elements (less than machine precision).

4.1 The Effect of Ill-Conditioning

In this section, we investigate the convergence behavior of point SOR, Krylov subspace, and block iterative methods on artificially more ill-conditioned problems. As it is mentioned in Chapter 3, three problems are selected for this purpose. The *medium* and *hard* test matrices are two more ill-conditioned versions of *easy*. Similarly, *ncd_alt1*, *ncd_alt2*, *mutex_alt1*, and *mutex_alt2* are more ill-conditioned versions of *ncd* and *mutex* test matrices, respectively.

Comparing the results of *easy*, *medium*, and *hard*, we see that Krylov subspace methods performed worse as the matrix becomes more ill-conditioned (see Tables A.1, A.4 and A.5). The same behavior is observed for point SOR and block methods. When we go from *easy* to *medium* and from *medium* to *hard*, whenever the method converged the number of iterations for convergence almost always increased. Also it is interesting to notice that IAD failed due to a reducible coupling matrix in *easy* for all partitionings. The different block structure for ncd test coupled with different nonzero values of *easy* seems to be the cause. However, IAD converged for *medium* with all but one of the partitionings and especially *equal* and *other* with which it converged in a very short time. *hard* with ncd test $\gamma = 10^{-4}$, $\gamma = 10^{-3}$ and $\gamma = 10^{-2}$ did not converge possibly because point SOR did not converge either for the original matrix (i.e., *hard*).

From the *pushout threshold* example, it looks like ill-conditioning the problem affects the performance of the Krylov subspace methods, and point and block SOR adversely. Nevertheless, a close investigation of the results of *ncd*, *ncd_alt1*, and *ncd_alt2* shows that this is not always the case (see Tables A.9, A.11 and A.15). Projection methods performed better for *ncd_alt1* than for *ncd* in all but two of the converging cases. In addition to that, BCG, QMR 2, and QMR 3 with $ILUK$ (10) converged for *ncd_alt1* whereas none of them had converged for the *ncd* case. Block SOR with ncd test $\gamma = 10^{-4}$ partitioning converged in 27.2 seconds for *ncd* whereas it only needed 4.5 seconds to converge with *ncd_alt1*. Similarly, IAD converged in 21.0 seconds for *ncd* but only 5.1 seconds were required for convergence with *ncd_alt1*. Using the *other* partitioning, block SOR and IAD also performed better with *ncd_alt1* than with *ncd*. On the other hand, Krylov subspace methods showed very poor performance when applied to *ncd_alt2* (which is more ill-conditioned than *ncd_alt1*) in that only three converging cases are recorded. The only improvement we can see in block iterative methods is with the *equal* partitioning. Block SOR and IAD took less time to converge with *equal* for *ncd_alt2* than for *ncd* and *ncd_alt1*. Point SOR did not converge with any of the three matrices. It is worth emphasizing that none of the block methods converged for *ncd_alt2* with ncd test $\gamma = 10^{-8}$ partitioning. This value partitions the matrix to 3 diagonal blocks of size $1,275$, $1,326$ and $20,825$. This unbalanced partitioning and solving the largest block iteratively with tolerance of 10^{-3} are most likely to be the reasons behind the poor performance of block SOR and IAD for this test matrix.

For the *mutex_alt1* test matrix all Krylov subspace methods performed at least as good as they did for the *mutex* matrix (see Tables A.23 and A.24). In addition, we should point out the considerable improvement in solution time of QMR 1 with $ILUTH$ (10^{-2}) and QMR 3 with $ILU0$ and $ILUTH$ (10^{-2}) for *mutex_alt1*, as they had not converged for *mutex*. Interestingly, in *mutex_alt1* and *mutex_alt2* the preconditioners formed by $ILUTH$ using threshold values 10^{-2} and 10^{-3} were exactly the same and that is why we report the results of only one threshold for these two matrices. The results of Krylov subspace methods when applied to *mutex_alt2* are slightly less competitive than those for *mutex_alt1* though they performed better in few cases (see Tables A.24 and

A.26). It is quite interesting to see BCG with $ILU0$ and $ILUK$ (10) and QMR 3 with $ILU0$ converged for *mutex_alt2* when they did not converge for *mutex* and *mutex_alt1*. Furthermore, for the *mutex* problem, point SOR, block SOR, and IAD took less time to converge as the matrix became more ill-conditioned.

For the six artificially more ill-conditioned test cases, we observe different convergence behavior for different types of solvers. In some cases, more ill-conditioning helps a given solver to converge faster. Hence, more ill-conditioning does not always imply worse performance. However, in general we cannot predict how the performance of a method changes with the degree of ill-conditioning.

4.2 The Effect of Reordering

In this section we study the effect of reordering the Markov chain state space on the convergence behavior of the solvers of interest.

In some problems, we observed that permuting the coefficient matrix to a wider band form causes Krylov subspace solvers to perform worse. This behavior is clear in the *easy* (Tables A.1, A.2 and A.3), *2D* (Tables A.6, A.7 and A.8), and *mutex_alt1* (Tables A.24 and A.25) test matrices though permuting *mutex_alt1* caused BCG, QMR 2, and QMR 3 to fail in some cases. Although *easy_gm4* and *easy_gm3* are reordered versions of the same matrix *easy* and IAD fails with all the partitionings in the original ordering, it is very interesting to see that the $\gamma = 10^{-4}$ ordering with *equal* and *other* partitionings gave converging IAD iterations. Theoretically, for ncd test partitionings, the smaller the degree of coupling of the coefficient (or stochastic) matrix is, the fewer iterations IAD requires to converge. Nevertheless, this was not the case for IAD in *2D_gm2* test matrix with ncd test $\gamma = 10^{-3}$ partitioning since it took more iterations to converge than with ncd test $\gamma = 10^{-2}$ partitioning. The reason behind this is that IAD with ncd test $\gamma = 10^{-3}$ partitioning solved all diagonal blocks iteratively. We recall that a relatively large tolerance (i.e., 10^{-3}) was used for solving diagonal blocks iteratively.

For other matrices such as *ncd* (Tables A.9 and A.10), *telecom* (Tables A.18, A.19 and A.20), and *qnatm* (Tables A.21 and A.22), permuting the matrix to a wider band form caused Krylov subspace methods to perform better in terms of the number of iterations taken to converge. However, increasing the bandwidth of a matrix generally increased the preconditioning time dramatically and hence led to a longer overall solution time despite possible decrease in the number of iterations. For *ncd_alt1* and *ncd_alt2* we could not see a clear difference in the behavior of projection methods as they performed better in some cases and worse in others (see Tables A.11–A.17). Point SOR required smaller solution time when permuted versions of *qnatm* and *mutex_alt1* were used, however the solution time increased when *easy* and *2D* were permuted. For all other permuted matrices point SOR did not converge. It is quite interesting to see that permuting a matrix did not cause any nonconverging point SOR to converge. Furthermore, we observed that *equal* and *other* partitionings on the permuted coefficient matrix gave smaller solution times than with the original (nonpermuted) coefficient matrix in many test cases for both block iterative methods. See, for instance, block SOR with *equal* in Table A.7 and IAD with *other* in Table A.10.

Chapter 5

Conclusion and Future Work

In this thesis, we compare and contrast the competitivity of projection methods, block iterative methods, and point SOR in computing the stationary probability vector of finite nearly completely decomposable (NCD) Markov chains. The methods are tested on 27 test cases arising from 7 real life problems. Among these there is a severely ill-conditioned matrix (i.e., NCD up to machine precision).

The numerical experiments show that block iterative methods are in general superior to Krylov subspace methods and point SOR for the test cases used. It is noticed that the more balanced, in terms of block sizes, the partitioning is, the more these methods take advantage of their divide-and-conquer nature, and hence the shorter time they need to converge. The iterative aggregation-disaggregation (IAD) algorithm proves to be very competitive with block SOR. In case the coupling matrix is of reasonable size, IAD usually gives satisfactory performance. However, the drawback of IAD is that it may fail or require an unreasonably long time to converge if the coupling matrix is reducible or of very large size. In addition, it is shown that straightforward partitionings are very competitive with those of ncd test. Out of 27 test cases, *equal* and *other* partitionings each outperformed ncd test partitionings in 8 cases.

Among the projection methods of interest, it is clear that BCGStab performs the best. It converged for all the test matrices with at least one preconditioner. Its convergence time is always the shortest or very close to that of an outperforming projection method. GMRES and CGS come second with the former being more costly in terms of memory requirements and number of flops per iteration. BCG and QMR perform rather poorly as they converged only for very few test cases. Point SOR did not give satisfactory results either; it converged in only 12 cases.

When the coefficient (or stochastic) matrix is extremely ill-conditioned (such as *leaky*), incomplete LU factorization may fail causing preconditioned Krylov subspace methods to fail too. Moreover, it is shown that $ILU0$ may be very efficient as a preconditioner if the matrix is quite large, not very narrow banded, or not extremely sparse as *2D*.

Ill-conditioning a given problem does not always imply worse performance. It is shown that in some cases ill-conditioning the problem helps the solver to converge faster. Permuting the coefficient matrix to a wider banded form usually affects the performance of Krylov subspace methods adversely in terms of overall computation time, though the method may converge in less iterations. This is due to the longer preconditioning time required for a wider banded matrix when $ILUTH$ and $ILUK$ are used. In some cases, *equal* and *other* partitionings benefit from reordering the state space and improve the solution time significantly.

As for further research directions, it is quite important to test other preconditioners with Krylov subspace methods and study how the behavior of these methods are affected. For block iterative methods, other straightforward partitionings may be worth investigating. Since the coupling matrix in IAD is a reduced order stochastic matrix, it would be interesting to study the effect of solving the coupling matrix using the IAD method itself, especially for large NCD systems.

Appendix A

Tables of Results

easy $n = 20,301$ $nz = 140,504$

Preconditioner	nzlu	Time	MFlops
ILU0	140,504	0.5	0.2
ILUTH (10^{-2})	60,097	5.9	0.3
ILUTH (10^{-3})	110,010	6.1	0.5
ILUK (10)	201,187	18.5	9.8

Method	Preconditioner	Time	# it	$\|\|r\|\|$	$\|\|Ax\|\|$	Bk. Err.
GMRES ($m = 20$)	ILU0	2.6	7	$0.65e-12$	$0.23e-15$	$0.69e-16$
	ILUTH (10^{-2})	4.7	14	$0.60e-12$	$0.50e-12$	$0.15e-12$
	ILUTH (10^{-3})	1.8	5	$0.64e-13$	$0.40e-13$	$0.12e-13$
	ILUK (10)	1.0	2	$0.60e-12$	$0.26e-13$	$0.80e-14$
BCG	ILU0	297.9	500*	$0.12e-06$	$0.40e-09$	$0.12e-09$
	ILUTH (10^{-2})	225.7	500*	$0.57e+00$	$0.54e-02$	$0.19e-01$
	ILUTH (10^{-3})	270.6	500*	$0.46e-03$	$0.51e-04$	$0.16e-04$
	ILUK (10)	348.0	500*	$0.13e-01$	$0.19e-02$	$0.85e-03$
CGS	ILU0	2.4	4	$0.62e-10$	$0.37e-13$	$0.11e-13$
	ILUTH (10^{-2})	3.6	8	$0.20e-10$	$0.17e-10$	$0.51e-11$
	ILUTH (10^{-3})	1.7	3	$0.45e-13$	$0.26e-13$	$0.79e-14$
	ILUK (10)	0.8	1	$0.13e-10$	$0.31e-12$	$0.94e-13$
BCGStab	ILU0	2.6	4	$0.47e-12$	$0.96e-15$	$0.29e-15$
	ILUTH (10^{-2})	6.8	15	$0.69e-10$	$0.57e-10$	$0.17e-10$
	ILUTH (10^{-3})	1.5	3	$0.11e-10^{s}$	$0.64e-11$	$0.20e-11$
	ILUK (10)	0.8	1	$0.11e-12$	$0.58e-14$	$0.18e-14$
QMR 2	ILU0	359.0	500*	$0.30e-08$	$0.51e-11$	$0.15e-11$
	ILUTH (10^{-2})	286.3	500*	$0.31e-09$	$0.28e-09$	$0.86e-10$
	ILUTH (10^{-3})	331.1	500*	$0.38e-08$	$0.24e-08$	$0.74e-09$
	ILUK (10)	4.5	5	$0.46e-10$	$0.18e-10$	$0.55e-11$
QMR 3	ILU0	367.4	500*	$0.38e-06$	$0.12e-09$	$0.35e-10$
	ILUTH (10^{-2})	286.3	500*	$0.53e-03$	$0.51e-03$	$0.24e-03$
	ILUTH (10^{-3})	38.3	42	$0.80e-10$	$0.59e-10$	$0.18e-10$
	ILUK (10)	20.1	24	$0.32e-10$	$0.25e-11$	$0.75e-12$

SOR

ω	$\|\|\Delta x\|\|$	Time	# it	$\|\|Ax\|\|$	Bk. Err.
1.0	$0.64e-11$	4.8	31	$0.69e-12$	$0.21e-12$

Block SOR

Partition.	ω	$\|\|\Delta x\|\|$	Time	# it	$\|\|Ax\|\|$	Bk. Err.	# Bl.
$\gamma = 10^{-4}$	1.0	$0.61e-11$	6.1	7	$0.60e-15$	$0.18e-15$	206
$\gamma = 10^{-3}$	1.0	$0.28e-10$	2.1	7	$0.15e-14$	$0.47e-15$	3,861
$\gamma = 10^{-2}$	1.0	$0.36e-10$	8.0	24	$0.34e-12$	$0.10e-12$	20,300
equal	1.0	$0.11e-10$	3.9	7	$0.10e-14$	$0.31e-15$	0
other	1.0	$0.84e-11$	1.9	7	$0.81e-15$	$0.25e-15$	2

IAD

Partition.	ω	$\|\|\Delta x\|\|$	Time	# it	$\|\|Ax\|\|$	Bk. Err.	# Bl.
$\gamma = 10^{-4}$	failed (coupling matrix reducible)						
$\gamma = 10^{-3}$	failed (coupling matrix reducible)						
$\gamma = 10^{-2}$	failed (coupling matrix reducible)						
equal	failed (coupling matrix reducible)						
other	failed (coupling matrix reducible)						

Table A.1: Numerical Results for *easy*.

easy_gm4 $n = 20,301$ $nz = 140,504$

Preconditioner	nzlu	Time	MFlops
ILU0	140,504	0.5	0.2
ILUTH (10^{-2})	62,678	13.4	0.4
ILUTH (10^{-3})	112,427	13.9	0.8
ILUK (10)	202,541	35.3	14.8

Method	Preconditioner	Time	# it	$\|\|r\|\|$	$\|\|Ax\|\|$	Bk. Err.
GMRES ($m = 20$)	ILU0	3.1	8	$0.21e-12$	$0.11e-15$	$0.33e-16$
	ILUTH (10^{-2})	4.9	14	$0.25e-11$	$0.16e-11$	$0.50e-12$
	ILUTH (10^{-3})	2.2	6	$0.24e-13$	$0.14e-14$	$0.43e-15$
	ILUK (10)	4.0	9	$0.22e-10$	$0.18e-15$	$0.56e-16$
BCG	ILU0	300.5	500*	$0.37e+00$	$0.53e-04$	$0.17e-04$
	ILUTH (10^{-2})	226.3	500*	$0.40e+01$	$0.40e-02$	$0.26e-01$
	ILUTH (10^{-3})	273.9	500*	$0.56e-02$	$0.58e-03$	$0.19e-03$
	ILUK (10)	353.0	500*	$0.92e-03$	$0.81e-08$	$0.25e-08$
CGS	ILU0	3.1	5	$0.22e-12$	$0.45e-16$	$0.14e-16$
	ILUTH (10^{-2})	3.7	8	$0.59e-10$	$0.39e-10$	$0.12e-10$
	ILUTH (10^{-3})	1.7	3	$0.35e-11$	$0.63e-12$	$0.19e-12$
	ILUK (10)	1.5	2	$0.28e-10$	$0.49e-16$	$0.15e-16$
BCGStab	ILU0	2.8	5	$0.91e-11^{s}$	$0.95e-15$	$0.29e-15$
	ILUTH (10^{-2})	11.1	24	$0.49e-10$	$0.81e-10$	$0.25e-10$
	ILUTH (10^{-3})	1.8	3	$0.12e-11$	$0.72e-12$	$0.22e-12$
	ILUK (10)	1.5	2	$0.25e-10$	$0.48e-16$	$0.15e-16$
QMR 2	ILU0	360.1	500*	$0.14e-07$	$0.16e-10$	$0.48e-10$
	ILUTH (10^{-2})	288.0	500*	$0.18e-08$	$0.20e-08$	$0.62e-09$
	ILUTH (10^{-3})	333.9	500*	$0.58e-07$	$0.61e-09$	$0.19e-09$
	ILUK (10)	415.1	500*	$0.63e-09$	$0.15e-13$	$0.45e-14$
QMR 3	ILU0	366.6	500*	$0.29e-03$	$0.39e-06$	$0.12e-06$
	ILUTH (10^{-2})	293.5	500*	$0.32e-08$	$0.21e-08$	$0.65e-09$
	ILUTH (10^{-3})	342.2	500*	$0.26e-03$	$0.79e-04$	$0.38e-04$
	ILUK (10)	421.2	500*	$0.17e-02$	$0.38e-08$	$0.11e-08$

SOR

ω	$\|\|\Delta x\|\|$	Time	# it	$\|\|Ax\|\|$	Bk. Err.
1.0	$0.27e-07$	29.2	207	$0.55e-11$	$0.18e-11$

Block SOR

Partition.	ω	$\|\|\Delta x\|\|$	Time	# it	$\|\|Ax\|\|$	Bk. Err.	# Bl.
$\gamma = 10^{-4}$	1.0	$0.61e-11$	5.8	7	$0.59e-15$	$0.18e-15$	206
$\gamma = 10^{-3}$	1.0	$0.28e-10$	1.9	7	$0.15e-14$	$0.46e-15$	3,861
$\gamma = 10^{-2}$	1.0	$0.64e-10$	7.5	24	$0.61e-12$	$0.19e-12$	20,300
equal	1.0	$0.95e-11$	12.6	42	$0.16e-14$	$0.50e-15$	0
other	1.0	$0.23e-10$	14.1	62	$0.21e-14$	$0.65e-15$	2

IAD

Partition.	ω	$\|\|\Delta x\|\|$	Time	# it	$\|\|Ax\|\|$	Bk. Err.	# Bl.
$\gamma = 10^{-4}$	failed (coupling matrix reducible)						
$\gamma = 10^{-3}$	failed (coupling matrix reducible)						
$\gamma = 10^{-2}$	failed (coupling matrix reducible)						
equal	1.0	$0.12e-10$	8.9	9	$0.21e-14$	$0.63e-15$	0
other	1.0	$0.76e-11$	6.9	7	$0.74e-15$	$0.22e-15$	2

Table A.2: Numerical Results for *easy_gm4*.

easy_gm3 $n = 20,301$ $nz = 140,504$

Preconditioner	nzlu	Time	MFlops
ILU0	$140,504$	0.5	0.2
ILUTH (10^{-2})	$60,097$	6.0	0.3
ILUTH (10^{-3})	$108,100$	6.1	0.4
ILUK (10)	$200,614$	18.4	8.6

Method	Preconditioner	Time	# it	$\|\|r\|\|$	$\|\|Ax\|\|$	Bk. Err.
GMRES ($m = 20$)	ILU0	2.7	7	$0.11e-12$	$0.14e-15$	$0.42e-16$
	ILUTH (10^{-2})	4.7	14	$0.60e-12$	$0.50e-12$	$0.15e-12$
	ILUTH (10^{-3})	1.8	5	$0.75e-13$	$0.45e-13$	$0.14e-13$
	ILUK (10)	1.1	2	$0.83e-12$	$0.28e-13$	$0.85e-14$
BCG	ILU0	36.7	61	$0.13e-11$	$0.74e-14$	$0.23e-14$
	ILUTH (10^{-2})	226.6	500*	$0.99e-04$	$0.82e-04$	$0.25e-04$
	ILUTH (10^{-3})	271.3	500*	$0.17e-06$	$0.57e-04$	$0.18e-06$
	ILUK (10)	21.2	30	$0.92e-10$	$0.11e-10$	$0.33e-11$
CGS	ILU0	2.5	4	$0.26e-11$	$0.91e-15$	$0.28e-15$
	ILUTH (10^{-2})	3.7	8	$0.20e-10$	$0.17e-10$	$0.51e-11$
	ILUTH (10^{-3})	1.7	3	$0.32e-13$	$0.19e-13$	$0.57e-14$
	ILUK (10)	0.8	1	$0.68e-11$	$0.16e-12$	$0.49e-13$
BCGStab	ILU0	2.5	4	$0.93e-13$	$0.24e-14$	$0.73e-15$
	ILUTH (10^{-2})	6.7	15	$0.69e-10^{s}$	$0.15e-09$	$0.47e-10$
	ILUTH (10^{-3})	1.5	3	$0.94e-11^{s}$	$0.55e-11$	$0.17e-11$
	ILUK (10)	0.8	1	$0.32e-13$	$0.39e-14$	$0.12e-14$
QMR 2	ILU0	358.9	500*	$0.70e-09$	$0.20e-11$	$0.60e-12$
	ILUTH (10^{-2})	287.8	500*	$0.77e-09$	$0.10e-08$	$0.32e-09$
	ILUTH (10^{-3})	329.6	500*	$0.10e-07$	$0.17e-06$	$0.52e-07$
	ILUK (10)	12.7	15	$0.99e-10$	$0.25e-10$	$0.78e-11$
QMR 3	ILU0	367.2	500*	$0.37e-03$	$0.53e-04$	$0.17e-04$
	ILUTH (10^{-2})	292.6	500*	$0.27e-08$	$0.22e-08$	$0.67e-09$
	ILUTH (10^{-3})	337.2	500*	$0.20e-03$	$0.77e-04$	$0.52e-04$
	ILUK (10)	419.3	500*	$0.62e-09$	$0.39e-10$	$0.12e-10$

<u>SOR</u>

ω	$\|\|\Delta x\|\|$	Time	# it	$\|\|Ax\|\|$	Bk. Err.
1.0	$0.44e-10$	25.6	180	$0.61e-11$	$0.14e-11$

<u>Block SOR</u>

Partition.	ω	$\|\|\Delta x\|\|$	Time	# it	$\|\|Ax\|\|$	Bk. Err.	# Bl.
$\gamma = 10^{-4}$	1.0	$0.61e-11$	5.8	7	$0.60e-15$	$0.18e-15$	206
$\gamma = 10^{-3}$	1.0	$0.28e-10$	1.9	7	$0.15e-14$	$0.46e-15$	$3,861$
$\gamma = 10^{-2}$	1.0	$0.55e-10$	7.5	24	$0.52e-12$	$0.16e-12$	$20,300$
equal	1.0	$0.60e-11$	4.4	10	$0.20e-14$	$0.60e-15$	0
other	1.0	$0.84e-11$	1.8	7	$0.81e-15$	$0.25e-15$	2

<u>IAD</u>

Partition.	ω	$\|\|\Delta x\|\|$	Time	# it	$\|\|Ax\|\|$	Bk. Err.	# Bl.
$\gamma = 10^{-4}$	failed (coupling matrix reducible)						
$\gamma = 10^{-3}$	failed (coupling matrix reducible)						
$\gamma = 10^{-2}$	failed (coupling matrix reducible)						
equal	failed (coupling matrix reducible)						
other	failed (coupling matrix reducible)						

Table A.3: Numerical Results for *easy_gm3*.

medium $n = 20,301$ $nz = 140,504$

Preconditioner	nzlu	Time	MFlops
ILU0	140,504	0.5	0.2
ILUTH (10^{-2})	155,313	5.8	0.7
ILUTH (10^{-3})	275,253	6.4	1.5
ILUK (10)	201,189	19.8	10.8

Method	Preconditioner	Time	# it	$\|\|r\|\|$	$\|\|Ax\|\|$	Bk. Err.
GMRES ($m = 20$)	ILU0	35.0	75	$0.99e-11$	$0.26e-11$	$0.30e-11$
	ILUTH (10^{-2})	6.7	15	$0.32e-11$	$0.48e-15$	$0.53e-15$
	ILUTH (10^{-3})	10.4	19	$0.48e-10$	$0.50e-14$	$0.55e-14$
	ILUK (10)	5.4	12	$0.73e-11$	$0.12e-13$	$0.13e-13$
BCG	ILU0	298.7	500*	$0.17e+03$	$0.15e-02$	$0.12e+00$
	ILUTH (10^{-2})	309.8	500*	$0.15e-03$	$0.29e-08$	$0.32e-08$
	ILUTH (10^{-3})	396.7	500*	$0.39e-02$	$0.26e-08$	$0.28e-08$
	ILUK (10)	343.4	500*	$0.15e+00$	$0.10e-03$	$0.12e-03$
CGS	ILU0	291.6	500*	$0.16e+15$	$0.23e-02$	$0.19e+00$
	ILUTH (10^{-2})	6.8	11	$0.11e-11$	$0.33e-15$	$0.37e-15$
	ILUTH (10^{-3})	4.0	5	$0.17e-10$	$0.35e-16$	$0.38e-16$
	ILUK (10)	5.5	8	$0.11e-11$	$0.19e-14$	$0.20e-14$
BCGStab	ILU0	17.6	30	$0.35e-10^{s}$	$0.92e-11$	$0.10e-10$
	ILUTH (10^{-2})	6.6	11	$0.50e-10^{s}$	$0.10e-13$	$0.11e-13$
	ILUTH (10^{-3})	3.2	4	$0.14e-10$	$0.73e-16$	$0.80e-16$
	ILUK (10)	4.5	7	$0.34e-10^{s}$	$0.57e-13$	$0.63e-13$
QMR 2	ILU0	366.3	500*	$0.25e-05$	$0.35e-06$	$0.39e-06$
	ILUTH (10^{-2})	376.8	500*	$0.18e-07$	$0.28e-11$	$0.30e-11$
	ILUTH (10^{-3})	463.4	500*	$0.76e-06$	$0.30e-12$	$0.33e-12$
	ILUK (10)	410.2	500*	$0.13e-07$	$0.21e-10$	$0.23e-10$
QMR 3	ILU0	368.2	500*	$0.90e-04$	$0.62e-04$	$0.17e-03$
	ILUTH (10^{-2})	378.7	500*	$0.45e-04$	$0.51e-07$	$0.56e-07$
	ILUTH (10^{-3})	465.9	500*	$0.99e-07$	$0.39e-13$	$0.43e-13$
	ILUK (10)	412.4	500*	$0.91e-07$	$0.15e-09$	$0.17e-09$

SOR

ω	$\|\|\Delta x\|\|$	Time	# it	$\|\|Ax\|\|$	Bk. Err.
1.1	$0.16e-09$	51.4	352	$0.40e-10$	$0.44e-10$

Block SOR

Partition.	ω	$\|\|\Delta x\|\|$	Time	# it	$\|\|Ax\|\|$	Bk. Err.	# Bl.
$\gamma = 10^{-3}$	1.0	$0.85e-10$	219.5	13	$0.14e-10$	$0.15e-10$	3
$\gamma = 10^{-2}$	1.0	$0.93e-10$	343.0	24	$0.21e-10$	$0.23e-10$	707
$\gamma = 10^{-1}$	1.1	$0.97e-10$	19.6	57	$0.34e-11$	$0.37e-11$	20,300
equal	1.0	$0.18e-11$	4.5	2	$0.28e-15$	$0.31e-15$	0
other	1.2	$0.38e-10$	43.5	218	$0.17e-10$	$0.18e-10$	2

IAD

Partition.	ω	$\|\|\Delta x\|\|$	Time	# it	$\|\|Ax\|\|$	Bk. Err.	# Bl.
$\gamma = 10^{-3}$	1.0	$0.82e-10$	220.0	13	$0.14e-10$	$0.15e-10$	3
$\gamma = 10^{-2}$	1.1	$0.61e-10$	320.0	10	$0.13e-11$	$0.14e-11$	707
$\gamma = 10^{-1}$	1.0	requires unreasonably long time					
equal	1.0	$0.13e-14$	5.7	2	$0.59e-16$	$0.65e-16$	0
other	1.1	$0.85e-10$	4.7	10	$0.55e-11$	$0.60e-11$	2

Table A.4: Numerical Results for *medium*.

hard $n = 20,301$ $nz = 140,504$

Preconditioner	nzlu	Time	MFlops
ILU0	140,504	0.5	0.2
ILUTH (10^{-2})	237,472	6.5	1.1
ILUTH (10^{-3})	860,386	12.4	10.5
ILUK (10)	201,187	17.7	8.4

Method	Preconditioner	Time	# it	$\|\|r\|\|$	$\|\|Ax\|\|$	Bk. Err.
GMRES ($m = 20$)	ILU0	207.1	480	$0.96e-10$	$0.45e-10$	$0.52e-08$
	ILUTH (10^{-2})	184.1	360	$0.41e-10$	$0.21e-10$	$0.24e-08$
	ILUTH (10^{-3})	35.2	40	$0.13e-11$	$0.38e-12$	$0.43e-10$
	ILUK (10)	163.2	340	$0.86e-10$	$0.34e-10$	$0.38e-08$
BCG	ILU0	297.1	500*	$0.82e+06$	$0.14e-02$	$0.64e-01$
	ILUTH (10^{-2})	374.2	500*	$0.31e+03$	$0.12e-02$	$0.65e-01$
	ILUTH (10^{-3})	839.7	500*	$0.47e+00$	$0.67e-03$	$0.55e-01$
	ILUK (10)	346.0	500*	$0.74e+03$	$0.77e-03$	$0.57e-01$
CGS	ILU0	291.2	500*	$0.30e+07$	$0.34e-03$	$0.98e-01$
	ILUTH (10^{-2})	364.0	500*	$0.32e+17$	$0.15e-02$	$0.73e-01$
	ILUTH (10^{-3})	37.2	23	$0.14e-10$	$0.42e-11$	$0.48e-09$
	ILUK (10)	337.8	500*	$0.11e+15$	$0.11e-02$	$0.13e+00$
BCGStab	ILU0	43.1	73	$0.31e-10^s$	$0.15e-10$	$0.17e-08$
	ILUTH (10^{-2})	40.5	55	$0.31e-10^s$	$0.16e-10$	$0.18e-08$
	ILUTH (10^{-3})	22.7	14	$0.98e-10$	$0.29e-10$	$0.33e-08$
	ILUK (10)	40.2	59	$0.55e-10^s$	$0.26e-10$	$0.30e-08$
QMR 2	ILU0	358.6	500*	$0.27e-04$	$0.25e-04$	$0.30e-01$
	ILUTH (10^{-2})	436.8	500*	$0.27e-04$	$0.23e-04$	$0.28e-01$
	ILUTH (10^{-3})	900.1	500*	$0.78e-08$	$0.23e-08$	$0.27e-06$
	ILUK (10)	410.6	500*	$0.26e-04$	$0.21e-04$	$0.44e-01$
QMR 3	ILU0	699.2	500*	$0.45e-04$	$0.32e-04$	$0.21e-01$
	ILUTH (10^{-2})	773.0	500*	$0.26e-07$	$0.14e-07$	$0.15e-05$
	ILUTH (10^{-3})	2010.0	500*	$0.15e-08$	$0.46e-09$	$0.53e-07$
	ILUK (10)	733.7	500*	$0.59e-07$	$0.24e-07$	$0.24e-05$

SOR

ω	$\|\|\Delta x\|\|$	Time	# it	$\|\|Ax\|\|$	Bk. Err.
1.0	$0.57e-06$	147.0	1,000*	$0.97e-07$	$0.42e-04$

Block SOR

Partition.	ω	$\|\|\Delta x\|\|$	Time	# it	$\|\|Ax\|\|$	Bk. Err.	# Bl.
$\gamma = 10^{-4}$	1.0	requires unreasonably long time					
$\gamma = 10^{-3}$	1.0	requires unreasonably long time					
$\gamma = 10^{-2}$	1.0	requires unreasonably long time					
equal	1.0	$0.77e-06$	338.7	1,000*	$0.33e-07$	$0.52e-05$	0
other	1.0	$0.78e-06$	185.1	1,000*	$0.72e-07$	$0.23e-04$	2

IAD

Partition.	ω	$\|\|\Delta x\|\|$	Time	# it	$\|\|Ax\|\|$	Bk. Err.	# Bl.
$\gamma = 10^{-4}$	1.0	requires unreasonably long time					
$\gamma = 10^{-3}$	1.0	requires unreasonably long time					
$\gamma = 10^{-2}$	1.0	requires unreasonably long time					
equal	1.0	failed (coupling matrix reducible)					
other	1.4	$0.93e-10$	31.6	112	$0.14e-10$	$0.15e-08$	2

Table A.5: Numerical Results for *hard*.

2D $n = 16,641$ $nz = 66,049$

Preconditioner	nzlu	Time	MFlops
ILU0	$66,049$	0.2	0.05
ILUTH (10^{-2})	$99,997$	2.6	0.40
ILUTH (10^{-3})	$138,392$	2.7	0.60
ILUK (10)	$165,819$	9.0	4.80

Method	Preconditioner	Time	# it	$\|\|r\|\|$	$\|\|Ax\|\|$	Bk. Err.
GMRES ($m = 20$)	ILU0	67.4	240	$0.63e-10$	$0.26e-09$	$0.20e-08$
	ILUTH (10^{-2})	6.2	20	$0.32e-12$	$0.19e-12$	$0.15e-11$
	ILUTH (10^{-3})	2.9	10	$0.54e-13$	$0.79e-14$	$0.63e-13$
	ILUK (10)	4.2	13	$0.17e-12$	$0.47e-12$	$0.37e-11$
BCG	ILU0	166.6	500*	$0.77e+04$	$0.70e-02$	$0.99e-01$
	ILUTH (10^{-2})	199.0	500*	$0.73e+00$	$0.44e-03$	$0.56e-01$
	ILUTH (10^{-3})	230.5	500*	$0.36e-03$	$0.98e-04$	$0.10e-02$
	ILUK (10)	252.1	500*	$0.18e+02$	$0.11e-02$	$0.15e-01$
CGS	ILU0	164.4	500*	$0.25e+14$	$0.11e-01$	$0.11e+00$
	ILUTH (10^{-2})	4.8	12	$0.11e-10$	$0.50e-11$	$0.40e-10$
	ILUTH (10^{-3})	2.4	5	$0.42e-11$	$0.52e-12$	$0.41e-11$
	ILUK (10)	4.1	8	$0.19e-10$	$0.32e-10$	$0.26e-09$
BCGStab	ILU0	12.7	38	$0.66e-10^s$	$0.70e-10$	$0.55e-09$
	ILUTH (10^{-2})	4.6	12	$0.19e-10^s$	$0.89e-11$	$0.70e-10$
	ILUTH (10^{-3})	2.2	5	$0.36e-10^s$	$0.44e-11$	$0.35e-10$
	ILUK (10)	3.8	8	$0.65e-11^s$	$0.11e-10$	$0.90e-10$
QMR 2	ILU0	217.3	500*	$0.30e-04$	$0.26e-04$	$0.12e-02$
	ILUTH (10^{-2})	248.1	500*	$0.55e-09$	$0.51e-09$	$0.40e-08$
	ILUTH (10^{-3})	281.0	500*	$0.11e-09$	$0.26e-10$	$0.20e-09$
	ILUK (10)	304.7	500*	$0.14e-09$	$0.24e-09$	$0.19e-08$
QMR 3	ILU0	224.0	500*	$0.34e-04$	$0.30e-04$	$0.20e-02$
	ILUTH (10^{-2})	252.9	500*	$0.87e-08$	$0.41e-08$	$0.32e-07$
	ILUTH (10^{-3})	286.1	500*	$0.15e-07$	$0.18e-08$	$0.14e-07$
	ILUK (10)	308.5	500*	$0.31e-09$	$0.54e-09$	$0.43e-08$

SOR

ω	$\|\|\Delta x\|\|$	Time	# it	$\|\|Ax\|\|$	Bk. Err.
1.4	$0.64e-09$	29.6	314	$0.71e-11$	$0.57e-10$

Block SOR

Partition.	ω	$\|\|\Delta x\|\|$	Time	# it	$\|\|Ax\|\|$	Bk. Err.	# Bl.
$\gamma = 10^{-3}$	1.0	$0.26e-15$	62.3	2	$0.22e-17$	$0.17e-16$	512
$\gamma = 10^{-2}$	1.0	$0.60e-15$	35.1	2	$0.30e-17$	$0.24e-16$	$5,192$
equal	1.4	$0.91e-10$	33.5	199	$0.60e-11$	$0.48e-10$	0
other	1.2	$0.98e-10$	35.1	205	$0.79e-11$	$0.63e-10$	2

IAD

Partition.	ω	$\|\|\Delta x\|\|$	Time	# it	$\|\|Ax\|\|$	Bk. Err.	# Bl.
$\gamma = 10^{-3}$	1.2	$0.11e-13$	62.5	2	$0.16e-15$	$0.12e-14$	512
$\gamma = 10^{-2}$	1.1	$0.79e-15$	121.4	2	$0.72e-17$	$0.57e-16$	$5,192$
equal	1.3	$0.65e-10$	9.5	39	$0.22e-11$	$0.17e-10$	0
other	1.3	$0.80e-10$	10.9	39	$0.26e-11$	$0.21e-10$	2

Table A.6: Numerical Results for *2D*.

2D_gm3 $n = 16,641$ $nz = 66,049$

Preconditioner	nzlu	Time	MFlops
ILU0	66,049	0.2	0.05
ILUTH (10^{-2})	202,660	13.1	0.90
ILUTH (10^{-3})	509,417	15.0	3.40
ILUK (10)	161,144	29.2	8.40

Method	Preconditioner	Time	# it	$\|\|r\|\|$	$\|\|Ax\|\|$	Bk. Err.
GMRES ($m = 20$)	ILU0	145.6	500*	$0.12e-08$	$0.14e-08$	$0.11e-07$
	ILUTH (10^{-2})	100.1	260	$0.34e-10$	$0.37e-10$	$0.29e-09$
	ILUTH (10^{-3})	103.3	180	$0.18e-10$	$0.35e-10$	$0.28e-09$
	ILUK (10)	85.5	240	$0.12e-10$	$0.14e-10$	$0.11e-09$
BCG	ILU0	168.5	500*	$0.26e+04$	$0.53e-02$	$0.28e-01$
	ILUTH (10^{-2})	277.1	500*	$0.13e+01$	$0.47e-02$	$0.37e-01$
	ILUTH (10^{-3})	517.0	500*	$0.76e-05$	$0.14e-04$	$0.11e-03$
	ILUK (10)	246.2	500*	$0.50e+01$	$0.45e-02$	$0.33e-01$
CGS	ILU0	167.6	500*	$0.89e+15$	$0.49e-02$	$0.27e-01$
	ILUTH (10^{-2})	15.8	29	$0.67e-10$	$0.72e-10$	$0.58e-09$
	ILUTH (10^{-3})	27.6	28	$0.30e-11$	$0.57e-11$	$0.45e-10$
	ILUK (10)	20.2	41	$0.63e-10$	$0.73e-10$	$0.58e-09$
BCGStab	ILU0	30.0	88	$0.44e-10^s$	$0.51e-10$	$0.40e-09$
	ILUTH (10^{-2})	14.6	27	$0.27e-10^s$	$0.29e-10$	$0.23e-10$
	ILUTH (10^{-3})	28.3	29	$0.27e-12^s$	$0.50e-12$	$0.40e-11$
	ILUK (10)	21.3	43	$0.90e-10$	$0.10e-09$	$0.83e-09$
QMR 2	ILU0	249.8	500*	$0.12e-03$	$0.11e-03$	$0.86e-02$
	ILUTH (10^{-2})	324.0	500*	$0.30e-08$	$0.32e-08$	$0.25e-07$
	ILUTH (10^{-3})	552.5	500*	$0.32e-07$	$0.59e-07$	$0.47e-06$
	ILUK (10)	293.0	500*	$0.17e-03$	$0.20e-03$	$0.18e-02$
QMR 3	ILU0	221.5	500*	$0.16e-03$	$0.15e-03$	$0.84e-02$
	ILUTH (10^{-2})	330.1	500*	$0.44e-07$	$0.47e-07$	$0.38e-06$
	ILUTH (10^{-3})	559.6	500*	$0.12e-06$	$0.22e-06$	$0.17e-05$
	ILUK (10)	300.3	500*	$0.12e-03$	$0.13e-03$	$0.13e-02$

SOR

ω	$\|\|\Delta x\|\|$	Time	# it	$\|\|Ax\|\|$	Bk. Err.
1.0	$0.13e-06$	86.5	1,000*	$0.82e-08$	$0.65e-07$

Block SOR

Partition.	ω	$\|\|\Delta x\|\|$	Time	# it	$\|\|Ax\|\|$	Bk. Err.	# Bl.
$\gamma = 10^{-3}$	1.0	$0.58e-15$	55.5	2	$0.17e-17$	$0.14e-16$	512
$\gamma = 10^{-2}$	1.0	$0.72e-15$	31.9	2	$0.13e-17$	$0.10e-16$	5,192
equal	1.6	$0.88e-10$	23.0	122	$0.28e-11$	$0.23e-10$	0
other	1.2	$0.99e-10$	54.7	297	$0.45e-11$	$0.35e-10$	2

IAD

Partition.	ω	$\|\|\Delta x\|\|$	Time	# it	$\|\|Ax\|\|$	Bk. Err.	# Bl.
$\gamma = 10^{-3}$	1.0	$0.58e-15$	55.0	2	$0.24e-17$	$0.19e-16$	512
$\gamma = 10^{-2}$	1.0	$0.61e-15$	146.5	2	$0.28e-17$	$0.22e-16$	5,192
equal	1.2	$0.77e-10$	10.7	30	$0.12e-11$	$0.93e-11$	0
other	1.3	$0.89e-10$	11.6	27	$0.95e-12$	$0.75e-11$	2

Table A.7: Numerical Results for *2D_gm3*.

2D_gm2 $n = 16,641$ $nz = 66,049$

Preconditioner	nzlu	Time	MFlops
ILU0	66,049	0.2	0.05
ILUTH (10^{-2})	173,390	9.2	0.80
ILUTH (10^{-3})	492,850	12.0	2.80
ILUK (10)	162,834	51.1	6.50

Method	Preconditioner	Time	# it	$\|\|r\|\|$	$\|\|Ax\|\|$	Bk. Err.
GMRES ($m = 20$)	ILU0	115.4	400	$0.27e-10$	$0.40e-10$	$0.29e-09$
	ILUTH (10^{-2})	12.7	36	$0.41e-12$	$0.36e-12$	$0.29e-11$
	ILUTH (10^{-3})	11.4	20	$0.16e-10$	$0.40e-10$	$0.32e-09$
	ILUK (10)	108.9	300	$0.37e-10$	$0.49e-10$	$0.39e-09$
BCG	ILU0	169.8	500*	$0.79e+01$	$0.48e-02$	$0.43e-01$
	ILUTH (10^{-2})	253.9	500*	$0.95e-03$	$0.11e-02$	$0.56e-02$
	ILUTH (10^{-3})	490.8	500*	$0.47e+01$	$0.20e-02$	$0.27e-01$
	ILUK (10)	263.2	500*	$0.81e+02$	$0.35e-02$	$0.28e-01$
CGS	ILU0	167.6	500*	$0.18e+08$	$0.45e-02$	$0.29e-01$
	ILUTH (10^{-2})	8.5	17	$0.14e-10$	$0.13e-10$	$0.10e-09$
	ILUTH (10^{-3})	17.3	18	$0.94e-12$	$0.23e-11$	$0.18e-10$
	ILUK (10)	245.6	500*	$0.80e+09$	$0.14e-01$	$0.15e+00$
BCGStab	ILU0	25.2	74	$0.73e-10^{s}$	$0.99e-10$	$0.79e-09$
	ILUTH (10^{-2})	8.4	17	$0.11e-10^{s}$	$0.94e-11$	$0.74e-10$
	ILUTH (10^{-3})	15.5	16	$0.20e-10$	$0.48e-10$	$0.38e-09$
	ILUK (10)	18.2	36	$0.88e-10$	$0.12e-09$	$0.92e-09$
QMR 2	ILU0	216.3	500*	$0.48e-04$	$0.64e-04$	$0.55e-03$
	ILUTH (10^{-2})	301.4	500*	$0.62e-08$	$0.55e-08$	$0.44e-07$
	ILUTH (10^{-3})	545.3	500*	$0.18e-09$	$0.44e-09$	$0.35e-08$
	ILUK (10)	296.0	500*	$0.83e-06$	$0.11e-05$	$0.87e-05$
QMR 3	ILU0	221.5	500*	$0.60e-05$	$0.10e-04$	$0.81e-04$
	ILUTH (10^{-2})	306.2	500*	$0.38e-08$	$0.34e-08$	$0.27e-07$
	ILUTH (10^{-3})	544.2	500*	$0.11e-06$	$0.27e-06$	$0.21e-05$
	ILUK (10)	305.2	500*	$0.80e-06$	$0.97e-06$	$0.77e-05$

SOR

ω	$\|\|\Delta x\|\|$	Time	# it	$\|\|Ax\|\|$	Bk. Err.
1.0	$0.72e-07$	86.1	1,000*	$0.43e-08$	$0.35e-07$

Block SOR

Partition.	ω	$\|\|\Delta x\|\|$	Time	# it	$\|\|Ax\|\|$	Bk. Err.	# Bl.
$\gamma = 10^{-3}$	1.0	$0.31e-10$	245.0	15	$0.55e-11$	$0.44e-10$	513
$\gamma = 10^{-2}$	1.0	$0.10e-14$	31.3	2	$0.35e-17$	$0.28e-16$	5,192
equal	1.6	$0.96e-10$	35.7	171	$0.37e-11$	$0.30e-10$	0
other	1.4	$0.73e-09$	46.7	234	$0.41e-10$	$0.33e-09$	2

IAD

Partition.	ω	$\|\|\Delta x\|\|$	Time	# it	$\|\|Ax\|\|$	Bk. Err.	# Bl.
$\gamma = 10^{-3}$	1.1	$0.91e-10$	232.5	13	$0.50e-11$	$0.40e-10$	513
$\gamma = 10^{-2}$	1.0	$0.79e-15$	106.1	2	$0.39e-17$	$0.31e-16$	5,192
equal	1.4	$0.92e-10$	10.3	29	$0.12e-11$	$0.96e-11$	0
other	1.3	$0.65e-10$	11.0	31	$0.92e-12$	$0.73e-11$	2

Table A.8: Numerical Results for *2D_gm2*.