Abstract

JOHN COURTNEY HAWS. Preconditioning KKT Systems. (Under the direction of Professor Carl D. Meyer.)

This research presents new preconditioners for linear systems. We proceed from the most general case to the very specific problem area of sparse optimal control.

In the first most general approach, we assume only that the coefficient matrix is nonsingular. We target highly indefinite, nonsymmetric problems that cause difficulties for preconditioned iterative solvers, and where standard preconditioners, like incomplete factorizations, often fail. We experiment with nonsymmetric permutations and scalings aimed at placing large entries on the diagonal in the context of preconditioning for general sparse matrices. Our numerical experiments indicate that the reliability and performance of preconditioned iterative solvers are greatly enhanced by such preprocessing.

Secondly, we present two new preconditioners for KKT systems. KKT systems arise in areas such as quadratic programming, sparse optimal control, and mixed finite element formulations. Our preconditioners approximate a constraint preconditioner with incomplete factorizations for the normal equations. Numerical experiments compare these two preconditioners with exact constraint preconditioning and the approach described above of permuting large entries to the diagonal.

Finally, we turn to a specific problem area: sparse optimal control. Many optimal control problems are broken into several phases, and within a phase, most variables and constraints depend only on nearby variables and constraints. However, free initial and final times and time-independent parameters impact variables and constraints throughout a phase, resulting in dense factored blocks in the KKT matrix. We drop fill due to these variables to reduce density within each phase. The resulting preconditioner is tightly banded and nearly block tri-diagonal. Numerical experiments demonstrate that the preconditioners are effective, with very little fill in the factorization.

PRECONDITIONING KKT SYSTEMS

by

John Courtney Haws

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Department of Mathematics

Raleigh, North Carolina

2002

APPROVED BY:

Carl D. Meyer Chair of Advisory Committee	Ilse C. F. Ipsen
Jeffrey Scroggs	Ernest Stitzinger

Dedication

To my wife Susan,

and to my parents

Gene and Andra Haws.

Biography

John Courtney Haws was born in Jackson, Tennessee, in 1968. He is a 1987 graduate of Maryville High School, in Maryville, Tennessee. In 1991, he obtained a Bachelor of Science degree in Mathematical Sciences from Loyola University in New Orleans, Louisiana. Prior to returning to Graduate School, he was a Teach For America corp member in the Rio Grande Valley, Texas, from 1992-1994, and worked from 1995-1997 as an analyst for First Commerce Corporation, in New Orleans. He began graduate work at North Carolina State University in 1997, and obtained a Master of Science degree in 2000.

Acknowledgements

The advisement of Carl Meyer has been consistently thoughtful and wise, and I am grateful for the opportunity to work with him. I also thank Becky Meyer for her support.

The readers on my committee have been insightful, helpful, and professional, and I am grateful for their support. I also thank the Graduate School Representative Dr. Adriana Kirkman for her participation.

I am grateful to have benefitted from the guidance of Michele Benzi, who has always been enthusiastic in his mentorship and generous with his knowledge. I also thank Carol, Joyce, Carlo, and Sophia for their hospitality and friendship.

There are many fellow graduate students to thank: Jörg Gablonsky for his help and support throughout; Rob, Jim, Brian and Chris; Tracy, Patrick, Mike and Michelle, Amy, Peter, Jordan, Jason, Vicky, Xiang-Dao, Jennifer, Manfred, Bob, Peach, Brad, Zager, Yaw, and Cab; and fellow gluttons-for-punishment David, Katie, Jörg, Kristy, Todd, Kim, Neil, Matt, Lea.

Thanks to many professors, especially Erich Kaltofen, Jaime Niño, David Estes, and Steve Scariano. I thank the organizers of the student seminar for inviting me to join: Tim Kelley, Steve Campbell, Pierre Gremaud, Ilse Ipsen, and Carl Meyer.

Much of the work for Chapter 2 was performed while I was a Graduate Research Assistant at Los Alamos National Laboratory: the hospitality and support of LANL are greatly appreciated. Also, I thank Iain Duff and Jacko Koster for providing the MC64 subroutines. I am indebted to Jane Cullum, Iain Duff, Daniel Szyld and two anonymous referees for their constructive criticism of an early draft of Chapter 2.

I thank John Betts and Carsten Keller for providing many of the test matrices used in the experiments in Chapter 3, and Dr. Ipsen for her thorough reading of an early draft of the chapter.

Much of the work for Chapter 4 was completed while I was a member the Boeing Company's Mathematics and Engineering Analysis Group. The support of the entire team and the Boeing Company is greatly appreciated. I am indebted to Dan'l Pierce, John Lewis, Wei-Pai Tang, Jason Wu, and John Betts.

Finally, I thank Susan, whose love and support has been the cornerstone of this work.

This work was funded in part by NSF grants CCR-ITR0113121, DMS9714811, and CCR9731856.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This dissertation focuses on solving linear systems

$$\mathcal{A}x = b, \tag{1.1}$$

where $\mathcal{A} \in \mathbb{R}^{n\times n}$ is a non-singular matrix, $x \in \mathbb{R}^n$ is the solution vector, and $b \in \mathbb{R}^n$ is a given *right-hand side*. Linear systems lie at the heart of mathematical models from such fields as circuit theory, management science, structural analysis, computational fluid dynamics, optimization, and optimal control, and thus solving linear systems is central in scientific computing.

From a theoretical point of view, (1.1) is trivially solved: if we represent the inverse of $\mathcal{A}$ by $\mathcal{A}^{-1}$, where $\mathcal{A}\mathcal{A}^{-1} = I$ with I denoting the identity matrix, then the solution is given by $x = \mathcal{A}^{-1}b$. However, from a computational point of view, finding the solution (or a reasonable approximation) can be very difficult, or even impossible. In fact, explicitly computing the inverse to solve a large set of linear equations is almost never the most efficient approach.

For many applications, the most efficient approach is one tailored to the specific problem at hand. General approaches to solving linear systems certainly exist, but when attributes of a specific problem are used to customize the solver, the general approaches rarely outperform the customized approach. Nevertheless, specific infor-

mation about a problem is not always available or is difficult to obtain, and so there is interest in fast solution techniques applicable to large linear systems. In this dissertation, we traverse the spectrum, developing efficient approaches for solving the most general linear systems, and gradually move towards efficient approaches tailored to specific linear systems.

One well-known approach for solving linear systems is Gaussian elimination. In general, Gaussian elimination requires the storage of all n^2 entries of the coefficient matrix and approximately $2n^3/3$ arithmetic operations. Matrices that arise in practice, such as in the fields named above, are usually *sparse*, that is, they often have only a few non-zeros per row. In many applications, the key to solving efficiently large linear systems is exploiting the sparsity and thereby decreasing the required storage and operations.

Methods for solving linear systems fall into two classes: direct methods and iterative methods. Direct methods involve a fixed number of steps that require a finite number of operations, and at the end of the process provide the solution. Iterative methods begin with an initial approximation $\mathbf{x}^{(0)}$ to the solution $\mathbf{x}$, and try to construct a sequence $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots$, such that $\lim_{n\to\infty} \mathbf{x}^{(n)} = \mathbf{x}$. In practice, the iteration is stopped when a current iterate $\mathbf{x}^{(k)}$ is judged to be sufficiently close to the true solution. Often, this is determined by a measure of the residual

$$r_k \;=\; b - A\mathbf{x}^{(k)}.$$

While the algorithms of direct methods can be very complicated, most methods are based on Gaussian elimination and factor the coefficient matrix $\mathcal{A}$ into the components L and U, with L lower triangular, U upper triangular, and $\mathcal{A} = LU$. The factors are used to solve (1.1) through the solution of $Ly = b$ by forward substitution followed by the solution of $Ux = y$ via back substitution. In most cases, the rows and columns of the coefficient matrix are permuted prior to the factorization in order to improve the accuracy or sparsity of the factorization. Thus, the steps of direct methods for

solving $\mathcal{A}x = b$ fall naturally into four phases:

1. A pre-ordering phase in which the rows and/or columns of the coefficient matrix are interchanged in order to maintain sparsity in the L and U factors; the reorderings may also focus on improving the numerical stability of the factorization.
2. An analysis phase in which the matrix structure is analyzed in order to set up appropriate storage schemes.
3. A factorization phase in which the numerical factorization is performed.
4. A solve phase in which the system is solved using forward and backward substitution.

For thorough treatments of direct methods, see [28] or [38]. The subject of this dissertation tries to improve the performance of iterative methods, but many of the techniques we employ are based in direct methods. In particular, steps one and four above are of interest in our research. Forward and backward solves are important in the application of our preconditioners, and we rely heavily on reorderings in order to improve the efficiency of the solves.

To gain efficiency, sparsity in the matrix must be exploited. In the case where a matrix has only a few non-zeros per row, forming the product of that matrix with a vector requires only a few n operations; if sparsity is not exploited, the matrix-vector product requires $2n^2$ arithmetic operations. In many applications, one does not have access to the actual matrix $\mathcal{A}$, but can only access the matrix implicitly through matrix vector product $\mathcal{A}v$, for a given vector v. For these reasons, approaches to solving $\mathcal{A}x = b$ using only matrix vector products have been developed. If one can solve $\mathcal{A}x = b$ with only a few matrix-vector multiplications, then such a procedure is faster and user less storage than Gaussian elimination.

One such class of methods is *Krylov subspace methods*, which iteratively look for solutions to (1.1) in the space

$$\mathcal{K}_k \;=\; \text{span}\left\{b, \mathcal{A}b, \ldots, \mathcal{A}^{k-1}b\right\} \tag{1.2}$$

For nonsingular matrices $\mathcal{A}$, the space (1.2) is closely tied to the inverse of $\mathcal{A}$ and is therefore a good space in which to find an approximate solution [49].

1.1 GMRES

A well-known Krylov subspace method is the generalized minimal residual (GMRES) algorithm [74], which minimizes the 2-norm of the residual r_k of the Krylov space

$$r_0 + \text{span}\{\mathcal{A}r_0, \mathcal{A}^2 r_0, \ldots, \mathcal{A}^k r_0\}.$$

GMRES constructs an orthonormal basis for the Krylov space via *Arnoldi's method* (Algorithm 1.1), a modified Gram-Schmidt-like process. Define the $n \times k$ matrix

Algorithm 1.1 Arnoldi's Method

1: Given q_1 with $\|q_1\|_2 = 1$.
2: **for** $j = 1, 2, \ldots, n$ **do**
3: $\quad \tilde{q}_{j+1} = \mathcal{A} q_j$
4: $\quad$ **for** $i = 1, \ldots, j$ **do**
5: $\quad\quad h_{ij} = \tilde{q}_{j+1}^T q_i$
6: $\quad\quad \tilde{q}_{j+1} \leftarrow \tilde{q}_{j+1} - h_{ij} q_i$
7: $\quad$ **end for**
8: $\quad h_{j+1,j} = \|\tilde{q}_{j+1}\|_2$
9: $\quad q_{j+1} = \tilde{q}_{j+1} / h_{j+1,j}$
10: **end for**

$Q_k = [q_1, \ldots, q_k]$ and the $k \times k$ upper Hessenberg matrix $H_k = [h_{ij}]$, for $j = 1, \ldots, k, i = 1, \ldots, \min\{j+1, k\}$. Let e_k be the kth unit vector, and $H_{k+1,k}$ be the matrix whose top $k \times k$ block is H_k and whose last row is zero, except for the last $(k+1, k)$ element, which is $h_{k+1,k}$. Then it is convenient to write the Arnoldi process in matrix form as

$$\mathcal{A} Q_k = Q_k H_k + h_{k+1,k} q_{k+1} e_k^T = Q_{k+1} H_{k+1,k}.$$

Iterate x_k is defined to be $x_k = x_0 + Q_k y_k$, where y_k solves the least squares problem

$$\min_y \|r_0 - AQ_k y\|,$$

using a QR factorization computed with Givens rotations.

GMRES is applicable to general linear systems, and its theoretical properties are somewhat well understood, and we therefore choose GMRES for our experiments in Chapters 3 and 4. Other Krylov subspace methods such as BiCGStab [75] or QMR [37] are also suitable for general linear systems, but less understood. In Chapter 2, we choose BiCGStab because it performed best in our experiments.

1.2 Preconditioning

A good approximation may not lie in the space (1.2) for a moderate sized k, or finding the approximation may require too much work. When the coefficient matrix $\mathcal{A}$ is close to the identity, almost any reasonable iterative method will converge very rapidly. But when $\mathcal{A}$ is not close to the identity, or in general in order to obtain reasonable convergence rates, we can replace the original system $\mathcal{A}x = b$ with the *left-preconditioned* system

$$\mathcal{P}^{-1}\mathcal{A}x = \mathcal{P}^{-1}b$$

or the *right-preconditioned* system

$$\mathcal{A}\mathcal{P}^{-1}y = b, \text{ with } x = P^{-1}y.$$

The matrix $\mathcal{P}$ is called a *preconditioner*. The subject of this dissertation is finding preconditioners $\mathcal{P}$ for linear systems. A preconditioner $\mathcal{P}$ should be chosen so that

- $\mathcal{P}$ is inexpensive to construct,
- solving linear systems with $\mathcal{P}$ is inexpensive, and
- $\mathcal{P}$ approximates $\mathcal{A}$, such as $\mathcal{P}^{-1}\mathcal{A}$ is close to the identity.

While these points are somewhat vague, the bottom line is that the total cost of constructing the preconditioner, applying it, and running the iterative method on the preconditioned system, should be considerably less than the cost of running the iterative method on the original system.

In our case, we will measure cost by the number of non-zeros that need to be stored in constructing the preconditioner, floating point operations performed in the entire process, and the number of iterations needed to converge. Another suitable measure is computational time; however, many of our experiments are performed in the MATLAB scientific computing environment. In these cases, our scripts cannot compare with the compiled functions, and so we do not measure computation time. Throughout this dissertation, whenever discussing numerical experiments, we specify what costs are measured and the environment in which the experiments were run.

There exist general preconditioners that are applicable to general linear systems, but often the most effective preconditioners are constructed for specific problems from specific problem areas. In this dissertation, we look at preconditioning indefinite matrices, beginning with the most general case, with no assumptions, and gradually narrowing our focus to specific problems. In Chapter 2, we present a new approach for preconditioning general linear systems from a variety of application areas. No assumptions (other than nonsingularity) are made on the coefficient matrix. In Chapter 3, we consider KKT systems, that is systems where the coefficient matrix has the structure

$$\mathcal{H} = \begin{bmatrix} H & B^T \\ B & 0 \end{bmatrix}. \tag{1.3}$$

KKT stands for Karush–Kuhn–Tucker, in reference to the Karush–Kuhn–Tucker first order conditions for existence of a solution in optimization problems [65]. Such systems are common in application areas, and still represent a very general class of linear systems. Our approach takes advantage of the structure of the system. In Chapter 4, we focus on a specific application area, sparse optimal control, and the matrices

that are produced by a specific software package SOCS. We take full advantage of characteristics specific to these matrices in constructing effective preconditioners.

1.3 KKT Linear Systems

KKT linear systems $\mathcal{H}x = b$, where $\mathcal{H}$ is of the form 1.3, arise in applications such as computational fluid dynamics, mixed finite element formulations of PDE problems, electrical engineering, optimization, optimal control problems, image processing, and many other areas.

For example, consider the following optimization problem of minimizing a quadratic function subject to equality constraints, that is, find the vector $x \in \mathbb{R}^n$ satisfying

$$\begin{aligned} \text{minimize} \quad & f(x) = \tfrac{1}{2}x^T Ax - b^T x, \\ \text{subject to} \quad & Bx = d. \end{aligned} \tag{1.4}$$

A solution x^* to (1.4) is a stationary point of the Lagrangian

$$L(x, \lambda) = \frac{1}{2}x^T Ax - b^T x + \lambda^T (Bx - d). \tag{1.5}$$

λ^T are called *Lagrange Multipliers*, and differentiating (1.5) with respect to x and λ results in

$$\begin{aligned} Ax - B^T\lambda &= b \\ Bx &= d, \end{aligned} \tag{1.6}$$

equivalently expressed as the KKT linear system

$$\begin{bmatrix} A & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} x \\ -\lambda \end{bmatrix} = \begin{bmatrix} b \\ d \end{bmatrix}.$$

1.4 Optimal Control Problems

In Chapter 4, we consider KKT systems that arise in optimal control problems, and specifically those generated within Boeing optimal control software SOCS. The

optimal control problem minimizes or maximizes a performance index, subject to boundary conditions, path constraints, and a nonlinear system of differential equations describing the motion or behavior of some physical process. The problem is approximated in finite dimensions through direct transcription, which discretizes the time parameter.

A nonlinear programming (NLP) algorithm is used to solve the optimization problem resulting from the transcription of the optimal control problem. SOCS currently uses a direct method to solve the linear systems generated by the NLP algorithm, but high amounts of fill in the factors often necessitate secondary storage. For large problems, Boeing is looking to utilize iterative methods to solve the linear systems, but effective preconditioners have not yet been developed for sparse optimal control problems

1.5 Contributions of the Research

We present new preconditioners for general matrices, general KKT matrices, and KKT matrices that arise in sparse optimal control. First, we experiment with one-sided permutations and related scalings as a preprocessing step for constructing standard preconditioners. Our experiments show that iterative solvers can thereby be effective for for highly indefinite and nonsymmetric linear systems. We also show that this technique is applicable in a black-box manner for KKT systems. Second, we present two new preconditioners for KKT systems based on a variation of robust factorizations for the normal equations. These preconditioners approximate a *constraint preconditioner*, defined in Chapter 3. Constraint preconditioned GMRES converges fast, but the construction and application of the preconditioner can be expensive. Our approximations greatly reduce the required storage, and for many problems, the iterative method preconditioned with our new preconditioners converges nearly as fast. Finally, we develop two new preconditioners for sparse optimal control prob-

lems. The preconditioners use information about the optimal control problem to improve sparsity. Our preconditioners are extremely sparse and accurate, in that GMRES iteration counts are kept low and few floating point operations are required, when preconditioned with our preconditioners.

1.6 Organization

In this dissertation, we describe preconditioners for general linear systems, with a purely algebraic approach that assumes no structure in the coefficient matrix, we describe preconditioners in which we assume the structure (1.3), and we also describe preconditioners for the specific linear systems that arise in sparse optimal control software. The organization of this document reflects this movement from the general to the specific. Chapter 2 focuses on nonsymmetric indefinite linear systems, assuming no structure whatsoever in the coefficient matrix. Chapter 3 considers a narrow class of problems, those with a the specific KKT structure (1.3). In Chapter 4, we focus on linear systems from sparse optimal control, and specifically those solved within SOCS (Sparse Optimal Control Software).

The main chapters of this dissertation (Chapters 2, 3, and 4) also appear, in similar form, as technical reports and journal articles. Chapter 2 was first published as a technical report [5], and later appeared in [6]. Chapter 3 is also a Boeing Mathematics and Computing Technology technical report [46], and has been submitted, in abbreviated form, to the journal Numerical Linear Algebra with Applications. Chapter [47] is also a Boeing Mathematics and Computing Technology technical report, and will be submitted to a peer-reviewed journal shortly.

Chapter 2

Preconditioning Highly Indefinite and Nonsymmetric Matrices

2.1 Introduction

2.1.1 Motivation and focus

We consider the solution of sparse linear systems $Ax = b$, where A is a general sparse $n \times n$ nonsingular matrix, by preconditioned Krylov subspace methods [43], [73]. A *general* sparse matrix is a matrix that has no special properties, such as symmetry, positive definiteness, diagonal dominance, etc. In particular, we focus on matrices that are highly unstructured, nonsymmetric (structurally as well as numerically) and indefinite, i.e., the eigenvalues of A can lie anywhere in the complex plane. Such matrices arise frequently in the simulation of chemical engineering processes, in economic modeling, in management science, in the analysis of circuits and power system networks, and elsewhere. These problems are very different from the ones arising from the numerical solution of elliptic partial differential equations (PDE's), and can cause serious difficulties for standard iterative methods and preconditioners.

There have been a few attempts to use preconditioned Krylov subspace methods in

these contexts, but in general the results have been far from satisfactory. For example, in [23], various incomplete factorization (ILU) preconditioners and iterative solvers were tested on a set of standard problems from chemical engineering. The main conclusions of that study were that such linear systems are difficult to solve with iterative methods (as indicated by the large number of reported failures) and that realizing the potential of iterative solvers will require improvements in the reordering and/or preconditioning schemes. Similar conclusions were reached in [61] for the use of iterative methods in circuit simulations. The use of preconditioned Krylov subspace methods for the solution of sparse linear systems arising in economic modeling has been investigated in [67] and [40]. There the conclusion was that iterative solvers are often superior to direct ones. However, some of the problems could not be solved by iterative methods; see [67].

Preconditioned iterative methods work especially well when the coefficient matrix is, at least to some degree, diagonally dominant. Reliable methods also exist to handle matrices that are symmetric positive definite, or M-matrices. Matrices with these properties arise frequently from the discretization of second-order, elliptic PDE's. Standard preconditioners, such as those based on incomplete factorizations of the coefficient matrix, are usually reliable under these circumstances and typically converge fast. However, they are often unstable or may not even be defined when the coefficient matrix has zeros on the main diagonal and/or is highly nonsymmetric (see the discussion in section 2). Furthermore, the presence of many eigenvalues with arbitrary real part (positive, negative and zero) causes serious difficulties for many Krylov subspace solvers. Matrices of this kind are loosely referred to as *highly indefinite*, regardless of whether they are symmetric or not. Coefficient matrices from chemical engineering, circuits, economics, etc., often exhibit a large number of zero diagonal entries and poor spectral distributions, and they represent a challenge for preconditioned Krylov subspace solvers.

2.1.2 Contributions of the research

In [66], Olschowka and Neumaier introduce new permutations and scaling strategies for Gaussian elimination. The goal is to preprocess the coefficient matrix so as to obtain an equivalent system with a matrix which is more diagonally dominant. This preprocessing reduces the need for partial pivoting, thereby speeding up the solution process, and increases the accuracy of the computed solution. Although the focus in [66] is on dense systems, the sparse case and the case of incomplete factorizations are also briefly discussed. These and other heuristics have been further developed and efficiently implemented by Duff and Koster; see [30] and [31]. Some evidence of the usefulness of these preprocessings in connection with sparse direct solvers and for ILU preconditioning has been provided in [30] and [31]; see also [56]. Our contribution is to carry out a systematic experimental study of the use of these permutation and scaling algorithms in the context of preconditioned iterative methods applied to challenging linear systems. We consider a number of different preconditioners (diagonal, ILU, sparse approximate inverses) and the combined use of nonsymmetric permutations to improve numerical stability with symmetric ones aimed at reducing fill-in in the preconditioner. Our experiments indicate that this preprocessing, and particularly maximum product transversals, enables the stable computation of the preconditioners, resulting in an overall solution strategy that is both reliable and cost-effective.

While we do not claim that this approach to preconditioning general sparse matrices will always work, we hope that the results in this chapter will contribute to a reassessment of the role of iterative solvers in areas where these methods had been almost written off as unreliable, such as chemical engineering. We also hope that one-sided permutations (and related scalings) will find widespread use for preconditioned iterative solvers for highly indefinite and nonsymmetric linear systems.

The chapter is organized as follows. In section 2 we briefly discuss the precon-

ditioners used in the chapter. In section 3, which is based on [31], we recall the one-sided permutations and scalings used to preprocess the matrices. The test problems used for the numerical experiments are described in section 4, and the numerical experiments themselves in section 5. Finally, in section 6 we present our conclusions.

2.2 ILU and approximate inverse preconditioners

In this section we briefly discuss the preconditioners used in the numerical experiments. We focus our attention on ILU–type techniques and on a sparse approximate inverse preconditioner in factorized form, AINV. These are general-purpose, algebraic preconditioners that have been used successfully to solve a wide range of problems, particularly from PDE's. A detailed treatment of ILU preconditioning is given in [73]. For a recent survey of sparse approximate inverse preconditioners, see [10].

Incomplete factorization methods compute sparse approximations to the triangular factors L and U of A. The incomplete factors are obtained by dropping nonzero entries generated in the course of the factorization process (*fill-ins*) according to some rule. Different dropping rules give rise to different ILU preconditioners. The ILU(0) preconditioner [62] discards all fill-in and retains only nonzeros in positions corresponding to the nonzero entries of A. This preconditioner is easy to implement and inexpensive to compute, but it is often not good enough, particularly for the kind of challenging problems considered in this research. More powerful preconditioners can be obtained by allowing more fill-in in the incomplete factors, or by dropping fill-ins based on their value rather than position. These techniques include level-of-fills ILU, denoted ILU(k). Here $k \geq 1$ is the fill level. For example, with fill level $k = 1$, fill entries created by the original entries in the matrix are retained; with fill level $k = 2$, fill entries created by the first level of fill are retained, etc. Another technique is dual threshold ILU, denoted ILUT(tol, p). For the ILUT preconditioner, $tol \geq 0$ is a drop tolerance and $p \geq 0$ denotes the number of off-diagonal nonzeros which are retained

in each row of the incomplete factor (usually the p largest ones among those nonzeros that are greater than *tol* in absolute value).

Although fairly robust in practice, ILU preconditioners often fail on general sparse matrices because of instabilities (see below). In fact, the incomplete factors may not even exist. One way to improve their robustness is by incorporating partial (column) pivoting in the incomplete factorization. In the case of ILUT, this leads to a variant, called ILUTP [73], which is sometimes successful when ILUT fails. However, even ILUTP often fails when applied to general sparse matrices; see [22] and subsection 5.4 below.

ILU preconditioners may suffer from two types of instability. They are discussed in detail in [22]; here we give a brief discussion only. Let $\bar{L}$ and $\bar{U}$ denote the incomplete factors of A, and let

$$R := \bar{L}\bar{U} - A$$

denote the residual matrix. Also, let

$$E := I - A(\bar{L}\bar{U})^{-1}$$

denote the error matrix, assuming that preconditioning is being applied on the right. Notice that $E = R(\bar{L}\bar{U})^{-1}$. Let $\|\cdot\|_F$ denote the Frobenius matrix norm. In the symmetric positive definite case, $\|R\|_F$ is proportional to the rate of convergence of the preconditioned conjugate gradient method [32]. On the other hand, in the case of general sparse matrices, $\|R\|_F$ alone is not a reliable indicator of the quality of the preconditioner; instead, $\|R\|_F$ and $\|E\|_F$ should both be taken into account. Note that $\|R\|_F$ can be interpreted as a measure of the *accuracy* of the preconditioner, regarded as an approximation to A, while $\|E\|_F$ gauges the *stability* of the approximation. In general, a large value of $\|R\|_F$ means a poor approximation and hence a poor preconditioner. This can be caused, for instance, by very small pivots encountered in the course of the incomplete factorization, and accounts for a first kind of instability. However, even if $\|R\|_F$ is small, it can happen that $\bar{L}^{-1}$ and/or $\bar{U}^{-1}$

have very large entries. Therefore $\|E\|_F = \|R(\bar{L}\bar{U})^{-1}\|_F$ could be large, in which case the preconditioned matrix is far from the identity, and the preconditioned iteration either stagnates or diverges, leading to a second kind of instability. We stress that the instability here is not in the incomplete factorization process, but in the solve step with the incomplete factors. Ill-conditioned ILU factors occur frequently for matrices that are far from symmetric and lack diagonal dominance; see [34], [22], [8].

For general sparse matrices, it is frequently the case that both kinds of instability occur simultaneously, with a crippling effect on the quality of the preconditioner. In general, the *complete* LU factorization of A may not even be defined without pivoting, or it may be unstable. This can happen, for example, when there are zero or small entries on the main diagonal. As long as A is nonsingular it has (in exact arithmetic) an LU factorization with pivoting, but this is not true for *incomplete* factorizations [71]. On the other hand, ILU factorizations which are both accurate and stable are possible for diagonally dominant matrices.

Because of these and other limitations of ILU-type preconditioners, alternative preconditioning techniques based on sparse approximate inverses have been intensively developed in the past few years. The AINV algorithm [7], [9], based on an incomplete biconjugation process, has been shown to be one of the most effective techniques in this class. Here the preconditioner is the product of two triangular matrices, which are sparse approximations to the inverses of the L and U factors of A. Sparsity is preserved by using a drop tolerance. The approximate inverse factors are computed directly from A; no knowledge of the factors L or U is needed. Factorized approximate inverse preconditioners share one drawback with ILU-type techniques: the construction phase is guaranteed to be breakdown-free only for special classes of matrices. Sufficient conditions are that A be symmetric positive definite or an H-matrix; see [54], [7], [53], [4]. For general sparse matrices, instabilities due to very small or zero pivots can occur during the construction of the preconditioner, with disastrous effects. This is perfectly analogous to the instability of the first kind for

ILU. Notice that, in contrast to ILU, the instability of the second kind—unstable ILU solves—is not an issue here.

Like for ILU, the performance of approximate inverse preconditioners in factorized form is sensitive to the ordering of the matrix. For (almost) structurally symmetric matrices having a stable AINV preconditioner, it was shown in [21] and [12] that symmetric reorderings that reduce fill-in in the inverse factors, like minimum degree or (generalized) nested dissection, improve the performance of the preconditioner. However, these symmetric reorderings alone are of little use for general sparse matrices, because the AINV preconditioner may not even be defined. As we shall see, the stability and effectiveness of AINV can be dramatically improved by nonsymmetric permutations and scalings that place large entries on the main diagonal.

2.3 Overview of algorithms for finding maximum transversals

In this section we give a summary of algorithms for determining permutations of a matrix that place entries of large absolute value on the main diagonal. The codes used to perform the permutations were taken from MC64, a set of Fortran routines which included in a forthcoming release of the Harwell Subroutine Library. Further details on the algorithms and implementations are provided in [31].

We examine three approaches for permuting large entries to the diagonal of matrices. First, we discuss methods for permuting a matrix so that the diagonal contains a maximum number of nonzeros; this method is referred to as finding a *maximum transversal* or *maximum matching*. Second, we discuss a method that permutes a matrix so that the product of the absolute value of the entries on the diagonal is maximized. Third, we discuss a variant of the second method, where the matrix is permuted so that the absolute value of the smallest entry on the diagonal is maxi-

mized.

Finally, we include discussion on how to permute and scale the entries of a matrix so that its diagonal entries are 1 in absolute value, and its off-diagonal entries are all less than or equal to 1 in absolute value.

In our experiments we found that, in general, the best results in terms of both preconditioner fill and convergence rates were obtained when matrices were permuted so as to maximize the product of the absolute value of the entries on the diagonal, and scaled so that the diagonal entries are 1 in absolute value, and off-diagonal entries are all less than or equal to 1 in absolute value. Also note that the timing behaviors mentioned in the discussions of the algorithms below are worst-case scenarios. The implementations in MC64 are efficient and the preprocessing phase is very fast, even for relatively dense matrices.

2.3.1 Finding a maximum transversal

Let $A = (a_{ij})$ be a general $n \times n$ matrix. Let $\mathcal{M}$ denote a set of at most n ordered index pairs $(i, j), 1 \leq i, j \leq n$, in which each row index i and each column index j appears at most once. $\mathcal{M}$ is called a *transversal* or *matching*. When $\mathcal{M}$ has maximum cardinality (n for structurally nonsingular matrices), $\mathcal{M}$ is called a *maximum transversal* or *maximum matching*. Note that the magnitudes of the nonzero entries are not considered when simply finding a maximum transversal.

In the case where $|\mathcal{M}| = n$, then $\mathcal{M}$ defines an $n \times n$ permutation matrix $Q = (q_{ij})$, where

$$\begin{cases} q_{ji} = 1, & \text{for } (i, j) \in \mathcal{M} \\ q_{ji} = 0, & \text{otherwise,} \end{cases}$$

and thus AQ and QA are matrices with the transversal entries on the diagonal. Because our code is row-oriented, we limit our discussion to row permutations, i.e., permutations of the form QA.

One of the options in MC64 uses the algorithm MC21 implemented by Duff [27], [26]. MC21 is a depth-first search algorithm with look-ahead; for a sparse matrix with τ nonzero entries, the algorithm has worst-case complexity of $\mathcal{O}(n\tau)$, but in practice exhibits $\mathcal{O}(n+\tau)$ behavior.

The use of such a reordering strategy is fundamental as the first step of permuting sparse reducible matrices to block triangular form. As mentioned in the introduction, permuting to a zero-free diagonal has been examined before as a reordering strategy for preconditioning; see, for instance, [23] and [9]. Clearly, such a permutation will prove beneficial when, under the original ordering, there are zeros lying on the diagonal. However, in our experiments, we found few cases where finding a maximum transversal provided more benefit than finding a maximum product transversal.

2.3.2 Finding a maximum product transversal

In this subsection, we discuss permuting a matrix so that the product of the absolute value of the entries along the diagonal is maximized. That is, we look for a permutation σ that maximizes

$$\prod_{i=1}^{n} |a_{\sigma(i)i}|. \tag{2.1}$$

This strategy, combined with the scalings mentioned below, was introduced in [66] for pivoting in dense Gaussian elimination.

First, this maximization problem is translated into a minimization problem. Let $a_i = \max_j |a_{ij}|$ denote the maximum value in row i of the matrix A. Define the matrix $C = (c_{ij})$ by

$$c_{ij} = \begin{cases} \log a_i - \log |a_{ij}|, & \text{for } a_{ij} \neq 0, \\ \infty, & \text{otherwise.} \end{cases}$$

Then maximizing (2.1) is equivalent to minimizing

$$\sum_{i=1}^{n} c_{\sigma(i)i}. \tag{2.2}$$

Minimizing the sum (2.2) is equivalent to finding a minimum weight perfect matching. In combinatorial optimization, this is known as the bipartite weighted matching problem, or linear assignment problem. MC64 uses a sparse variant of the bipartite weighted matching algorithm introduced in [66]. Fundamental to finding a minimum weight perfect matching is finding a shortest augmenting path, which in turn relies on a sparse variant of Dijkstra's algorithm [25]. For a full $n \times n$ matrix, the assignment problem can be solved in $\mathcal{O}(n^3)$ time; for a sparse matrix with τ nonzero elements, the problem can be solved in $\mathcal{O}(n\tau \log n)$ time.

We will use the following result for applying the bipartite weighted matching algorithm to a matrix C [33]: A perfect matching $\mathcal{M}$ has minimum weight if and only if there exist vectors $u = (u_i)$ and $v = (v_i)$, each of length n, such that

$$\begin{cases} u_i + v_j = c_{ij} & \text{for } (i,j) \in \mathcal{M}, \\ u_i + v_j \leq c_{ij}, & \text{for } (i,j) \notin \mathcal{M}. \end{cases} \tag{2.3}$$

These vectors u and v are used in scaling the permuted matrix (see below).

2.3.3 Finding a bottleneck transversal

In this subsection we consider a simple variation of the method discussed in the previous subsection. Here we are interested in finding a permutation of A such that the smallest absolute value on the diagonal is maximized. That is, we look for a permutation σ that maximizes

$$\min_{1 \leq i \leq n} |a_{\sigma(i)i}|, \tag{2.4}$$

and define the matrix $C = (c_{ij})$ by

$$c_{ij} = \begin{cases} a_i - |a_{ij}|, & \text{for } a_{ij} \neq 0, \\ \infty, & \text{otherwise.} \end{cases}$$

Then maximizing (2.4) is equivalent to minimizing

$$\max_{1 \leq i \leq n} c_{\sigma(i)i}. \tag{2.5}$$

Thus the maximum product transversal algorithm can be applied here, with only minor modifications, such as to replace the sum operation (2.2) by the max operation (2.5).

Note that this method regards only the smallest entry on the diagonal of the permuted matrix. For example, as mentioned in [31], consider a matrix having a row containing only one nonzero entry whose absolute value is the smallest in the matrix. Then the bottleneck transversal algorithm may return a transversal with small values on the diagonal. MC64 takes a somewhat different approach to avoid this problem; see [30], [31] for details.

Scaling the matrix prior to finding a bottleneck transversal also alleviates this problem. Ultimately, though, we found few examples where bottleneck transversal permutations proved superior to maximum product transversal permutations.

2.3.4 Scaling

It is often beneficial to scale the matrices using the parameters u_i and v_j from (2.3) obtained in the weighted matching algorithm. To this end, define diagonal matrices D_1 and D_2 by

$$\begin{aligned} D_1 &= \mathrm{diag}(d_1^1, d_2^1, \ldots, d_n^1), \quad d_j^1 = \exp(v_j)/a_j, \\ D_2 &= \mathrm{diag}(d_1^2, d_2^2, \ldots, d_n^2), \quad d_i^2 = \exp(u_i). \end{aligned} \tag{2.6}$$

Then QD_1AD_2 is a matrix whose diagonal entries are one in absolute value, and whose off-diagonal entries are all less than or equal to one, in absolute value. Such a matrix is referred to as an I-matrix in [66]. By simply changing the sign of rows (or columns) having a diagonal entry equal to -1 we obtain a matrix with unit diagonal. The eigenvalues of such a matrix lie in discs centered at 1, with radii equal to the sum of the absolute values of the off-diagonal entries in the corresponding row. Therefore, the less the matrix deviates from a diagonally dominant matrix, the more the eigenvalues cluster around 1. In the ideal case, all the discs of such a matrix have

radii less than 1, and the matrix is strictly row-wise diagonally dominant. In turn, this guarantees that ILU and AINV preconditioners are well-defined. It is also a favorable situation for Krylov subspace methods, since all the eigenvalues have positive real part.

2.4 Description of test problems

In this section we describe the matrices that were used in the numerical experiments. Most of these matrices are available in the public domain [29], [24], [64]. They are representative of problems from a variety of applications and are difficult to solve with iterative methods. The matrices are listed in Table 2.1 below, together with some basic information. In Table 2.9 we report the estimate for the condition number returned by MATLAB (except for the three last problems, which are too large).

The first eight matrices are from chemical engineering and represent simulations of different chemical processes. They are not large, highly unstructured, and structurally nonsymmetric, with most of the diagonal entries equal to zero. In addition, they are highly indefinite and tend to be very ill-conditioned. Matrices similar to these have been used to test ILU–type preconditioners in [23]. In that paper, MC21 was used to obtain a zero-free diagonal: the results were poor.

The next five matrices come from mathematical economics and management. These matrices are also highly unstructured, nonsymmetric, indefinite with most diagonal entries equal to zero.

The next three matrices arise in circuit design. The first two problems are described in [61] and are available from the Matrix Market [64]. The original matrices WATSON4 and WATSON5 are rectangular; as in [61], we appended one row at the bottom of these matrices to make them square (and we changed the names to WATSON4a and WATSON5a). The row vector used was $e_n^T = (0, \ldots, 0, 1)$. All diagonal

Table 2.1: Test problem information.

Application Area: Chemical Engineering			
Matrix	Description	order	nonzeros
WEST0655	Sixteen stage column section	655	2854
WEST0989	Seven stage column section	989	3537
WEST1505	Eleven stage column section	1505	5445
WEST2021	Fifteen stage column section	2021	7353
LHR01	Light hydrocarbon recovery	1477	18592
LHR02	Light hydrocarbon recovery	2954	37206
BAYER09	Chemical process simulation	3083	21216
BAYER10	Chemical process simulation	13436	94926
Application Area: Economic Models and Linear Programming			
Matrix	Description	order	nonzeros
MAHINDAS	Economic model of Victoria	1258	7682
ORANI678	Economic model of Australia	2529	90158
BP200	Simplex method basis matrix	822	3802
GEMAT11	Power flow in 2400 bus system – initial simplex method basis	4929	33185
GEMAT12	Power flow in 2400 bus system – basis after 100 iterations	4929	33111
Application Area: Circuit Modeling			
Matrix	Description	order	nonzeros
WATSON4a	Jacobian at step 4, 1 row added	468	2870
WATSON5a	Jacobian at step 4, 1 row added	1854	10848
CIRCUIT3	Jacobian from nonlinear DAE system	12127	48137
Application Area: PDE problems			
Matrix	Description	order	nonzeros
SHERMAN2	Thermal simulation with steam injection	1080	23094
LNS3937	Linearized Navier-Stokes equations	3937	25407
UTM5940	Plasma physics, tokamak modeling	5940	83842
SLIDE	Solid deformation model (ALE3D)	20191	1192535
TWO-DOM	Solid deformation model (ALE3D)	22200	1188152
VENKAT25	2D unstructured Euler solver, time step=25	62424	1717792

entries are nonzero. The third circuit matrix was kindly provided by Wim Bomhof of Utrecht University; see [19]. This matrix has some zero diagonal entries. All three matrices are very sparse, and they exhibit a good deal of structural symmetry.

Finally, we included six matrices from the discretization of PDE's. SHERMAN2 does not present any difficulty for ILU preconditioning, but appears to pose a challenge for sparse approximate inverse preconditioners; see, e.g., [2], [10], [44] and [42]. Matrix LNS3937 has a zero diagonal block corresponding to the divergence constraint in the Navier-Stokes equations, and is challenging for both ILU and approximate inverse techniques; see, respectively, [22] and [2]. Zero diagonal blocks induced by constraints also occur in the unstructured finite element matrices SLIDE and TWODOM, which were kindly provided by Ivan Otero (Lawrence Livermore National Laboratory). Matrix UTM5940 is fairly difficult to solve with ILU–type methods [22] and even more so by sparse approximate inverse techniques. Matrix VENKAT25 was included because it is difficult for AINV. These last two matrices have no zero diagonal entries.

These matrices are just a selection from a larger set which was used for the tests; the chosen problems are representative of the results observed. As we will see, preprocessing makes all these problems solvable by iterative methods preconditioned with ILUT and AINV, and in many cases even with ILU(0) or ILU(1) preconditioning. We found, however, several systems cannot be solved by our techniques. In these cases, the preconditioned iteration usually converged, but to an inaccurate solution. Among these matrices are SHYY41 (also discussed in [22]), NNC666 and NNC1374, GRAHAM1, several of the FIDAP matrices, and some of the LHR0$*$c matrices from chemical engineering, all available in [24] or [64]. We tried to solve these problems by a *direct* method, namely, Gaussian elimination with partial pivoting, but again the computed solution was not accurate. The same happened when we used the *complete* factors computed with the direct method as preconditioners for Krylov subspace methods—a form of iterative refinement. Not surprisingly, these matrices are severely

ill-conditioned. One cannot blame an algorithm for being unable to produce accurate solutions to extremely ill-conditioned problems. In this chapter, we only present results for problems which could be solved with some accuracy, since these are the only results that make sense.

2.5 Numerical experiments

The numerical experiments in this section assess how the MC64 permutation and scaling routines impact the robustness and performance of preconditioned Krylov subspace methods. All algorithms were implemented in Fortran77 using double precision arithmetic. The codes were compiled by `f77` with the `-O3` optimization option. Test runs were performed on a Sun Ultra 5 workstation for all test problems except the last three, for which one 250 MHz processor of an SGI Origin 2000 computer was used.

We tried three different Krylov subspace solvers: BiCGStab [75], GMRES [74] and TFQMR [36]. While the three algorithms performed similarly on most problems, BiCGStab was somewhat better overall than the others. Therefore, we will present results for BiCGStab only. The preconditioners used are diagonal (Jacobi) scaling, ILU(0), ILU(1), ILUT and AINV. Besides these, we performed experiments also with ILUTP [73] and SPAI [44]. In all cases, right preconditioning was used. The right-hand side b was chosen so that the system $Ax = b$ has the solution $x = (x_i)$ with $x_i = i,\ 1 \leq i \leq n$. We also tested other choices of b, with similar results. The initial guess was always the zero vector, $x_0 = 0$. The iteration was stopped when the ℓ_2-norm of the initial residual was reduced by at least eight orders of magnitude, or when a maximum number of iterations $maxit = \min\{n, 2000\}$ was reached.

Some comments on the accuracy of the approximate solutions corresponding to this stopping criterion are in order. As reported in Table 2.9, many test matrices are very ill-conditioned, and a small residual does not necessarily guarantee a small error

in the solution. Nevertheless, the stopping criterion produces solutions with good relative accuracy except in a few cases (BAYER09, GEMAT12, CIRCUIT3), where some of the components of the solution were incorrect. For these cases, reducing the stopping tolerance from 10^{-8} to 10^{-12} resulted in accurate solutions, at the expense of an increase in the number of iterations of roughly 30%. However, all the results presented in the subsequent sections correspond to the stopping tolerance 10^{-8}.

2.5.1 Testing reliability

Here we discuss the impact of the preprocessing on the reliability of preconditioned BiCGStab. We only report iteration counts. In Table 2.2 we report the results of runs using diagonal (Jacobi) preconditioning for the original matrix (under "orig") and for different preprocessings: the basic maximum transversal (under "mc21"), the bottleneck transversal (under "bt"), the maximum product transversal without scalings (under "mpd") and with scalings (under "mps").

A "‡" means that the preconditioner was not defined, due to zeros on the main diagonal. A "†" means failure to converge within the maximum number of allowed iterations. A "*bd*" denotes a breakdown in the BiCGStab acceleration. Notice that mc21 leaves matrices WATSON4a, WATSON5a, SHERMAN2, UTM5940 and VENKAT25 unchanged, since these matrices have no zero diagonal entries.

The only problem that can be solved without any preprocessing is the small circuit matrix WATSON4a, which requires a number of iterations almost equal to the order of the matrix. While the maximum and bottleneck transversals lead to virtually no improvements, the maximum product transversals result in convergence in nine cases. In particular, six out of the eight chemical engineering problems and both economics problems can be solved using Jacobi preconditioning combined with the maximum product transversal and associated scalings. Notice that with mps, Jacobi preconditioning simply has the effect of changing the sign of those matrix columns

Table 2.2: Iteration count, Jacobi preconditioning.

Matrix	orig.	mc21	bt	mpd	mps
WEST0655	‡	†	†	†	†
WEST0989	‡	†	†	239	171
WEST1505	‡	†	†	536	570
WEST2021	‡	†	†	342	455
LHR01	‡	†	†	421	238
LHR02	‡	†	†	1195	1109
BAYER09	‡	†	†	†	244
BAYER10	‡	†	†	†	†
MAHINDAS	‡	†	†	348	137
ORANI678	‡	†	1133	217	196
BP200	‡	†	†	†	†
GEMAT11	‡	†	†	†	†
GEMAT12	‡	†	*bd*	†	†
WATSON4a	467	467	467	222	183
WATSON5a	†	†	†	†	†
CIRCUIT3	‡	†	†	†	†
SHERMAN2	†	†	†	466	†
LNS3937	‡	†	†	†	†
UTM5940	†	†	†	†	†
SLIDE	‡	†	†	†	†
TWO-DOM	‡	†	†	†	†
VENKAT25	†	†	†	†	†

for which the corresponding diagonal entry is -1.

The next three tables report the results obtained for various ILU–type preconditioners. These preconditioners are sensitive to the ordering of the equations and unknowns. Therefore, after applying the various preprocessings, we also reorder the matrix with a symmetric permutation. Consider for instance mps preprocessing. Denote the row and column scalings associated with the maximum product transversal with D_1 and D_2, respectively. Let Q denote the corresponding row permutation that permutes the scaled matrix to a zero-free diagonal form. Then the symmetric permutations are based on the adjacency graph of the symmetric matrix $\hat{A} + \hat{A}^T$, where $\hat{A} = QD_1AD_2$. Once the permutation matrix P has been determined, one solves the following linear system:

$$(P^TQD_1AD_2P)y = P^TQD_1b.$$

The solution of the original linear system is then $x = D_2Py$. The preconditioner calculation is carried out on the scaled and reordered matrix

$$\tilde{A} = P^TQD_1AD_2P.$$

The purpose of the scalings D_1, D_2 and of the one-sided permutation Q is to improve stability. The purpose of the symmetric permutation P is to make the preconditioner more accurate. For structurally symmetric matrices that are numerically unsymmetric and far from diagonally dominant, it was found in [8] that the performance and robustness of ILU–type preconditioners was generally improved by the reverse Cuthill–McKee ordering [28], denoted "rcm" in the tables (see column "SO"). Thus, we used rcm as the default ordering for ILU preconditioners. Although it is not always the best ordering, rcm gave often good results, in agreement with the conclusions in [8]. Whenever rcm performed poorly we switched to another symmetric ordering, like multiple minimum degree [58] (denoted "mmd") or generalized nested dissection [57] (denoted "gnd"). Occasionally, no symmetric reordering was the best

Table 2.3: Iteration count, ILU(0) preconditioning.

Matrix	SO	orig.	mc21	bt	mpd	mps
WEST0655	gnd	‡	‡	‡	157	144
WEST0989	gnd	‡	‡	‡	51	60
WEST1505	mmd	‡	‡	‡	1498	903
WEST2021	mmd	‡	‡	‡	136	162
LHR01	rcm	‡	†	†	52	49
LHR02	rcm	‡	‡	‡	210	62
BAYER09	rcm	‡	‡	‡	32	42
BAYER10	rcm	‡	‡	‡	†	†
MAHINDAS	rcm	‡	‡	167	34	32
ORANI678	rcm	‡	83	50	28	21
BP200	mmd	‡	‡	126	510	603
GEMAT11	rcm	‡	†	†	153	140
GEMAT12	rcm	‡	†	†	702	838
WATSON4a	rcm	131	131	127	60	89
WATSON5a	rcm	†	†	†	†	†
CIRCUIT3	rcm	‡	‡	†	†	†
SHERMAN2	no	8	8	†	12	11
LNS3937	rcm	‡	†	†	†	†
UTM5940	no	†	†	†	†	†
SLIDE	rcm	‡	†	628	335	335
TWO-DOM	rcm	‡	†	†	419	513
VENKAT25	rcm	90	90	†	88	117

option (denoted “no” in the tables). In these tables, a “‡” indicates a failure due to pivot breakdown—i.e., a zero or exceedingly small pivot occurred in the incomplete factorization.

The results for ILU(0) preconditioning are reported in Table 2.3. Again, using just mc21 is ineffective, with the only exception of matrix ORANI678. Results for the bottleneck transversal are not much better. However, the robustness of ILU(0) is greatly improved when the maximum product transversal is used. No pivot breakdown occurs, and all but five problems can be solved. The five failures are due to very slow convergence except for CIRCUIT3, for which the ILU(0) factors are unstable. Using different symmetric reorderings for these problems did not help.

Table 2.4: Iteration count, ILU(1) preconditioning.

Matrix	SO	orig.	mc21	bt	mpd	mps
WEST0655	gnd	‡	‡	‡	‡	†
WEST0989	rcm	‡	‡	‡	32	38
WEST1505	rcm	‡	‡	‡	37	45
WEST2021	rcm	‡	‡	‡	68	74
LHR01	rcm	‡	‡	†	64	58
LHR02	rcm	‡	‡	‡	143	104
BAYER09	rcm	‡	‡	‡	10	14
BAYER10	rcm	‡	‡	‡	1176	1707
MAHINDAS	rcm	‡	‡	108	9	16
ORANI678	rcm	‡	‡	69	14	13
BP200	mmd	‡	58	36	94	†
GEMAT11	rcm	‡	‡	107	43	46
GEMAT12	rcm	‡	‡	†	183	55
WATSON4a	rcm	126	126	92	27	26
WATSON5a	no	†	†	†	†	932
CIRCUIT3	rcm	‡	‡	†	†	†
SHERMAN2	rcm	8	8	†	4	4
LNS3937	gnd	379	*bd*	†	264	355
UTM5940	no	151	151	†	168	119
SLIDE	mmd	‡	‡	†	226	259
TWO-DOM	mmd	‡	‡	†	151	155
VENKAT25	rcm	56	56	†	57	81

We notice that there are two matrices, SHERMAN2 and VENKAT25, which can be easily solved with ILU(0) preconditioning without any preprocessing. Indeed, using mps results in a slight deterioration in the rate of convergence. Clearly, if a given combination of preconditioner and Krylov subspace solver gives good results, the use of preprocessing is not necessary and should not be used.We further observe that in several cases, mpd (without scalings) gives better results than mps. The same phenomenon occurs also for Jacobi and ILU(1) preconditioning (see Tables 2.2 and 2.4).

In Table 2.4 we report the results obtained with ILU(1) preconditioning. Again, mc21 and bt offer little benefit, whereas the use of mpd or mps allows all but

three problems to be solved. With mpd we had one failure due to pivot breakdown (WEST0655), one due to slow convergence (WATSON5a), and one due to unstable ILU factors (CIRCUIT3). For mps, unstable ILU solves were the cause of all three failures. Notice that in a few cases, ILU(1) performs worse than ILU(0). However, all PDE problems can be solved with ILU(1) when either mpd or mps is used, with mpd giving somewhat better results on average.

It is already clear from these results that the reliability of preconditioned iterative solvers is greatly enhanced by the use of one-sided permutations aimed at placing large entries on the main diagonal, even when simple preconditioners like ILU(0) and ILU(1) are used. There are, however, a few hard problems for which this approach does not work. Additional robustness can be achieved by using a drop tolerance.

In Table 2.5 we present results for the popular dual-threshold ILUT(tol, p) preconditioner. Our strategy for the experiments was the following. We used the default parameters $tol = 10^{-1}$ and $p = 5$, with rcm as the basic symmetric reordering. Whenever this combination produced poor results, we switched to a different symmetric reordering until a good one was found (in the following order: mmd, gnd, no reordering). When this didn't work we changed the ILUT parameters by decreasing tol and, if necessary, increasing p. We do not claim that this leads to an optimal, or even good preconditioning strategy: rather, the purpose of these experiments is to demonstrate that iterative solvers can be made reliable without much need for fine-tuning. We emphasize that the values $tol = 10^{-1}$ and $p = 5$ are not typical for ILUT. Usually, a much smaller drop tolerance and a much larger value of p are used, particularly for hard problems: see, e.g., [22]. In other words, we chose a very sparse preconditioner, in many cases even sparser than the original matrix. We made this choice deliberately, with the intent to show that most problems become very easy to solve once the preprocessing phase is applied. In general, better results can be obtained with a different choice of the parameters.

The results in Table 2.5 show that with mps, all problems can be solved with

Table 2.5: Iteration count, ILUT preconditioning.

Matrix	SO	tol, p	orig.	mc21	bt	mpd	mps
WEST0655	gnd	$10^{-1}, 5$	‡	‡	‡	65	37
WEST0989	rcm	$10^{-1}, 5$	‡	‡	470	21	9
WEST1505	mmd	$10^{-1}, 5$	‡	‡	†	513	53
WEST2021	rcm	$10^{-1}, 5$	‡	‡	‡	110	34
LHR01	rcm	$10^{-1}, 5$	‡	‡	‡	252	41
LHR02	rcm	$10^{-1}, 5$	‡	‡	‡	‡	78
BAYER09	rcm	$10^{-1}, 5$	‡	‡	†	†	14
BAYER10	mmd	$10^{-4}, 20$	‡	‡	‡	‡	65
MAHINDAS	rcm	$10^{-1}, 5$	‡	†	959	8	9
ORANI678	rcm	$10^{-1}, 5$	‡	†	62	12	14
BP200	mmd	$10^{-1}, 5$	‡	†	28	17	11
GEMAT11	rcm	$10^{-1}, 5$	‡	‡	†	1471	303
GEMAT12	rcm	$10^{-1}, 5$	‡	‡	†	354	306
WATSON4a	rcm	$10^{-1}, 5$	457	457	457	208	31
WATSON5a	rcm	$10^{-5}, 25$	6	6	6	6	6
CIRCUIT3	rcm	$10^{-4}, 20$	80	‡	516	36	90
SHERMAN2	rcm	$10^{-1}, 5$	37	37	†	8	10
LNS3937	rcm	$10^{-3}, 10$	‡	‡	†	776	24
UTM5940	no	$10^{-4}, 20$	164	164	†	302	141
SLIDE	rcm	$10^{-1}, 5$	‡	†	†	†	491
TWO-DOM	rcm	$10^{-1}, 5$	‡	†	†	†	494
VENKAT25	rcm	$10^{-2}, 5$	†	†	‡	†	98

Table 2.6: Iteration count, AINV preconditioning.

Matrix	SO	*tol*	orig.	mc21	bt	mpd	mps
WEST0655	no	10^{-1}	‡	‡	‡	†	176
WEST0989	mmd	10^{-1}	‡	‡	‡	141	32
WEST1505	mmd	10^{-2}	‡	‡	†	1693	37
WEST2021	mmd	10^{-2}	‡	‡	†	148	8
LHR01	mmd	10^{-1}	‡	†	‡	251	74
LHR02	mmd	10^{-1}	‡	‡	‡	385	127
BAYER09	mmd	10^{-1}	‡	‡	†	15	8
BAYER10	no	3.5×10^{-2}	‡	‡	‡	781	43
MAHINDAS	mmd	10^{-1}	‡	‡	29	15	7
ORANI678	no	10^{-1}	‡	†	251	10	9
BP200	mmd	5×10^{-2}	‡	‡	20	‡	5
GEMAT11	mmd	10^{-1}	‡	‡	†	802	230
GEMAT12	mmd	10^{-1}	‡	‡	†	†	389
WATSON4a	mmd	10^{-1}	21	21	21	14	20
WATSON5a	mmd	10^{-3}	51	51	51	51	114
CIRCUIT3	no	10^{-2}	†	‡	†	268	43
SHERMAN2	mmd	10^{-1}	†	†	†	50	14
LNS3937	gnd	10^{-1}	‡	†	†	†	112
UTM5940	gnd	10^{-1}	1268	1268	†	1388	406
SLIDE	mmd	10^{-1}	†	†	†	603	306
TWO-DOM	mmd	10^{-1}	†	†	965	491	224
VENKAT25	mmd	10^{-1}	†	†	‡	385	411

ILUT preconditioning. In most cases the basic combination of rcm reordering with $tol = 10^{-1}$ and $p = 5$ was sufficient to solve the problem. In some cases, an alternative symmetric reordering and/or additional fill-in had to be used in order to achieve convergence. The hardest problems for ILUT appear to be BAYER10, CIRCUIT3, and UTM5940. Complementing the maximum product transversal with scalings improves the reliability and performance of ILUT–preconditioned BiCGStab in nearly all tested cases. The only exception is CIRCUIT3, for which the number of iterations increases from 36 to 90.

In Table 2.6 we present results for the drop tolerance–based sparse approximate inverse AINV preconditioner. Like ILU–type preconditioners, AINV is sensitive to

the ordering. In [21] and [12] it was shown that mmd is generally a good ordering for AINV, with gnd a close second. On the other hand, rcm cannot be recommended in general. Thus, the default parameters were mmd reordering and $tol = 10^{-1}$. For a few hard problems, it was necessary to switch to gnd reordering (or no reordering) and to reduce the drop tolerance. However, no effort was made to tune the parameters for optimal performance. As for the ILU preconditioner, failures due to pivot breakdowns are denoted by "‡" slow convergence by "†".

As for ILUT, we see that mps preprocessing enables BiCGStab preconditioned with AINV to solve all our test problems. For instance, problem SHERMAN2, which is notoriously difficult for approximate inverse methods, becomes very easy to solve. We also see that in most cases, scalings improve the performance of the maximum product transversal. More importantly, scalings improve reliability, as shown by the fact that there are four problems that cannot be solved with mpd alone.

2.5.2 Timing results

The experiments illustrate that mps preprocessing dramatically increases the reliability and performance of Krylov subspace methods preconditioned with drop tolerance–based preconditioners, like ILUT and AINV. But how expensive is the preprocessing phase in practice? Because this preprocessing phase depends on the numerical values of the matrix coefficients, is not easy to amortize, except when solving a sequence of linear systems with the same coefficient matrix and different right-hand sides. If the coefficient matrix changes, the preprocessing has to be applied anew.

Timing results for ILUT and AINV are presented in Tables 2.7 and 2.8, respectively. We report the time for mps preprocessing (under NSO-time), the time for the symmetric reordering (under SO-time), the time for computing the preconditioner (under P-time), the time to perform the iterative solve phase (It-time) and, in the last column, the total solution time (under Tot-time). Timings are in seconds, and

Table 2.7: Test results for ILUT preconditioning.

Matrix	NSO-time	SO-time	P-time	ρ	Its	It-time	Tot-time
WEST0655	0.009	0.006	0.005	1.22	37	0.110	0.130
WEST0989	0.011	0.004	0.004	0.967	9	0.056	0.075
WEST1505	0.018	0.017	0.007	0.916	53	0.323	0.365
WEST2021	0.024	0.009	0.011	0.997	34	0.333	0.378
LHR01	0.091	0.019	0.013	0.349	41	0.369	0.494
LHR02	0.214	0.042	0.026	0.346	78	1.54	1.82
BAYER09	0.113	0.025	0.017	0.401	14	0.280	0.440
BAYER10	0.799	0.556	0.189	1.22	65	6.63	8.19
MAHINDAS	0.019	0.009	0.009	0.613	9	0.080	0.117
ORANI678	0.185	0.132	0.077	0.087	14	0.452	0.850
BP200	0.009	0.016	0.004	0.856	11	0.049	0.078
GEMAT11	0.079	0.038	0.038	0.632	303	8.05	8.21
GEMAT12	0.097	0.038	0.039	0.662	306	8.26	8.43
WATSON4a	0.006	0.004	0.004	0.799	31	0.063	0.078
WATSON5a	0.015	0.031	0.031	1.85	6	0.128	0.211
CIRCUIT3	0.127	0.106	0.610	2.02	90	7.08	7.88
SHERMAN2	0.053	0.021	0.020	0.228	10	0.111	0.206
LNS3937	0.131	0.030	0.129	2.59	24	0.973	1.27
UTM5940	0.223	—	1.50	3.13	141	14.5	16.2
SLIDE	0.977	0.938	0.588	0.090	491	67.1	69.6
TWO-DOM	0.976	0.956	0.595	0.100	494	69.5	72.0
VENKAT25	1.43	1.28	2.59	1.05	98	48.5	53.8

were measured with the `dtime` function. We also report the density ρ of the preconditioner, defined as the ratio between the number of nonzeros in the preconditioner over the number of nonzeros in the original coefficient matrix A. The number of iterations to converge (Its) is also included. The parameters and symmetric reorderings are the same as those used for the runs in Tables 2.5 and 2.6.

The cost of the preprocessing is usually small compared to the overall solution costs. For the larger problems the cost of preprocessing, including the time to apply the symmetric reordering, is negligible compared to the total solve time. We also see that in most cases, the time for the nonsymmetric permutation and scaling is comparable to the time required for the symmetric permutations, which are based on

Table 2.8: Test results for AINV preconditioning.

Matrix	NSO-time	SO-time	P-time	ρ	Its	It-time	Tot-time
WEST0655	0.009	—	0.122	10.2	176	1.54	1.67
WEST0989	0.011	0.010	0.018	2.36	32	0.175	0.214
WEST1505	0.018	0.017	0.094	5.92	37	0.499	0.628
WEST2021	0.024	0.023	0.203	7.87	8	0.228	0.478
LHR01	0.091	0.108	0.499	2.67	74	1.56	2.26
LHR02	0.214	0.231	1.10	3.00	127	5.94	7.49
BAYER09	0.113	0.128	0.228	1.73	8	0.270	0.739
BAYER10	0.799	—	4.67	5.96	43	11.2	16.7
MAHINDAS	0.019	0.139	0.061	1.27	7	0.082	0.301
ORANI678	0.185	—	0.776	0.146	9	0.332	1.29
BP200	0.009	0.016	0.026	2.46	5	0.046	0.097
GEMAT11	0.079	0.057	0.172	1.62	230	8.57	8.88
GEMAT12	0.097	0.058	0.224	2.04	389	15.8	16.2
WATSON4a	0.006	0.014	0.011	1.01	20	0.049	0.080
WATSON5a	0.015	0.072	0.167	1.77	114	1.34	1.59
CIRCUIT3	0.127	—	3.82	1.89	43	3.43	7.38
SHERMAN2	0.053	0.071	0.094	0.352	14	0.148	0.366
LNS3937	0.131	0.059	0.971	5.89	112	7.05	8.22
UTM5940	0.223	0.158	2.80	2.77	406	45.0	48.2
SLIDE	0.977	1.39	12.0	0.273	306	50.4	64.8
TWO-DOM	0.976	1.68	11.8	0.254	224	36.1	50.6
VENKAT25	1.43	1.40	20.1	0.838	411	184.	207.

the (unweighted) graph of the matrix only.

The performance of ILUT preconditioning is impressive: in most cases, rapid convergence is obtained with a very sparse preconditioner. For the few cases where convergence is slow (GEMAT11, GEMAT12, SLIDE and TWO-DOM), we found that much faster convergence and smaller timings result if more fill-in is allowed. For instance, using ILUT($10^{-3}, 10$) on GEMAT11 results in convergence in 40 iterations with preconditioner density $\rho = 1.46$, and the total solution time becomes 1.81 seconds.

It is our opinion that ILUT preconditioning with mps preprocessing, can become a useful tool for solving linear equations from chemical engineering applications. Because rapid convergence can be achieved with very sparse incomplete factorization preconditioners, this approach may be competitive with sparse direct methods. This approach also has considerable potential for matrices arising from economic modeling and management science, although we have less experience with such problems.

On the other hand, it is unclear whether this approach is really useful for matrices arising in circuit simulations. Sparse direct solvers suffer very little fill-in on such problems when a good ordering is adopted. If iterative solvers are to compete with direct methods, they must converge extremely fast with very sparse preconditioners. From our test runs, this seems to be difficult to achieve unless the incomplete factorization closely approaches a complete one. In [19] a combined direct/iterative method is described in which the preconditioned iteration is applied to a relatively small Schur complement matrix, which appears to be as fast as sparse direct solvers.

In terms of timings and storage requirements, AINV (Table 2.8) is generally more expensive than ILUT, but an important advantage of AINV, its parallelizability, is not captured by these one-processor experiments. For some matrices, the density of the preconditioner is rather high. Sparser preconditioners can be obtained by using a larger value of *tol*, but this may slow down convergence considerably. This is especially true for the chemical engineering problems. On the other hand, we found that for

the PDE problems faster convergence and smaller overall timings can be obtained by allowing more fill in the preconditioner. The same applies to the GEMAT∗ problems. Notice the good performance of AINV on the matrices from economics, MAHINDAS and ORANI678. As for the circuit matrices, similar remarks as for ILUT apply. The main point here is that mps preprocessing dramatically increases the reliability of both ILUT and AINV preconditioning, therefore considerably widening their applicability.

2.5.3 Further analysis of the results

In order to have a better understanding of the effect of the MC64 permutations and scalings used for preprocessing, we collected the statistics presented in Table 2.9. Under "condest" we report the condition number (estimated with the MATLAB function `condest`) for the original matrix and for the matrix scaled by the row and columns scalings associated with mps preprocessing. The condition number could not be estimated for the three largest matrices.

Under "d.d. rows" we report the number of (weakly) diagonally dominant rows in the original matrix, in the matrix permuted with the maximum product transversal (mpd), and in the matrix scaled and permuted with mps. Under "d.d. cols" similar statistics are reported relative to the number of (weakly) diagonally dominant columns.

The statistics show that the chemical engineering matrices greatly benefit from the permutations and scalings: the number of diagonally dominant rows and columns is greatly increased, and ill-conditioning is drastically reduced. Analogous remarks apply to problems MAHINDAS, ORANI678, BP200 and GEMAT11. Hence, it is not surprising that preconditioned iterative methods perform well on these problems when mps preprocessing is used.

For problems GEMAT12 and CIRCUIT3, the scalings have the effect of increasing the condition number. However, for GEMAT12 the number of weakly diagonally

Table 2.9: Conditioning and diagonal dominance statistics.

	condest		d.d. rows			d.d. cols		
Matrix	orig.	scaled	orig.	mpd	mps	orig.	mpd	mps
WEST0655	1.6E+12	2.0E+04	4	321	355	3	260	340
WEST0989	5.7E+12	1.9E+04	2	638	666	0	412	641
WEST1505	9.0E+12	2.5E+04	2	957	1004	0	619	969
WEST2021	7.5E+12	2.4E+04	2	1256	1340	0	808	1309
LHR01	5.4E+06	4.4E+04	20	323	438	0	766	710
LHR02	8.2E+06	5.5E+04	40	626	856	0	1532	1409
BAYER09	2.3E+21	1.1E+04	1	1610	1733	1	1737	2168
BAYER10	3.8E+15	1.5E+05	2	4653	7048	0	7081	7501
MAHINDAS	1.0E+13	5.0E+03	2	910	896	0	635	762
ORANI678	1.0E+07	1.1E+06	72	1866	1826	0	1653	1692
BP200	8.9E+08	3.8E+03	0	317	317	1	368	480
GEMAT11	3.7E+08	6.8E+06	2	1508	1536	2	1409	1238
GEMAT12	3.7E+08	1.2E+13	1	1465	1213	1	1398	1298
WATSON4a	8.8E+10	8.9E+07	161	374	270	377	161	261
WATSON5a	5.5E+07	3.2E+06	187	1264	1099	1268	187	332
CIRCUIT3	3.6E+07	3.0E+08	7865	7503	8245	7354	7650	8227
SHERMAN2	1.4E+12	3.3E+03	634	204	292	74	560	460
LNS3937	1.0E+17	1.9E+04	509	1283	689	307	892	701
UTM5940	1.9E+09	7.6E+08	762	915	882	925	766	926
SLIDE	n/a	n/a	1330	1203	1885	1201	1277	1274
TWO-DOM	n/a	n/a	2917	4627	4727	2917	4627	4700
VENKAT25	n/a	n/a	0	0	0	0	0	0

dominant rows and columns is greatly increased by the mpd preprocessing. The increase is somewhat smaller when scalings are used, but it is still a significant improvement. This makes the preconditioner computation stable, and therefore the net effect is positive. Matrices WATSON4a and WATSON5a are better conditioned after scaling and have a greater fraction of diagonally dominant rows after mpd. However, the scalings have the effect of reducing the number of diagonally dominant rows as compared to using mpd alone. Perhaps this explains why using mps often gives worse results than using mpd alone for these matrices (see Tables 2.3 and 2.6).

For the matrices arising from PDE problems, the permutations and scalings are generally beneficial. For VENKAT25, it is not clear from the results reported in Table 2.9 that the preprocessing does any good. However, we know that ILUT and AINV benefit from the preprocessing. Although no row or column became diagonally dominant after the permutations and scalings, we found that the Gerschgorin bounds on the real and imaginary parts of the eigenvalues were one order of magnitude smaller after mps preprocessing, suggesting that the scaled and permuted matrix is less far from diagonally dominant and has a more clustered spectrum than the original matrix.

2.5.4 Experiments with ILUTP and SPAI

We also experimented with the ILUTP preconditioner (ILUT with partial pivoting) [73], which is generally more reliable than ILUT. However, we found that ILUTP preconditioning applied to the original matrices, or even to those permuted with mc21, is hardly better than ILUT, unless very large amounts of fill are allowed. Moreover, we found several problems that could not be solved by ILUTP even with very large amounts of fill ($tol = 0$, $p = 30$). These are WEST0655, LHR01, LHR02, BAYER09, BAYER10, BP200, and LNS3937. The failure of ILUTP was due to pivot breakdowns in all cases, except for LNS3937, where it appeared to be due to unsta-

ble ILU factors. It is mentioned in [22] that ILUTP with row pivoting (rather than column pivoting) is able to solve LHR01, but it took 134 iterations of GMRES(50) and a rather dense preconditioner. Using ILUT with mps, we can solve LHR01 in 41 iterations with a very sparse preconditioner (see Table 2.5). On the other hand, none of the techniques used in [22] succeeded in solving problem LNS3937, which is easily solved by ILUT with mps preprocessing. Hence, based on our experiments, using standard ILUT with mps preprocessing is more reliable than using ILUTP alone.

A combination of ILUTP and mps gives fast convergence, sometimes better than those obtained with ILUT for the same choice of the parameters. However, column pivoting makes ILUTP slightly more expensive than ILUT, and it is unclear whether ILUTP is worth using with mps preprocessing.

Additional experiments were performed with the sparse approximate inverse preconditioner SPAI [44], based on adaptive Frobenius norm minimization. Because the SPAI preconditioner is not factored, it is insensitive to the ordering of the equations and unknowns. However, it is sensitive to scalings. We tested SPAI on the original matrices and with mps preprocessing. Scalings appear to improve the reliability and performance of SPAI. This was especially true for the chemical engineering matrices; none of these matrices could be solved with SPAI, even with generous amounts of fill, but all of them could be solved after applying mps preprocessing. The same happened with MAHINDAS, BP200, SHERMAN2, LNS3937 and UTM5940.

In [2, p. 112], the authors give results for LNS3937 using a parallel implementation of BiCGStab preconditioned with SPAI. This took 1942 iterations. The approximate inverse contained 1588045 nonzeros, and had a density $\rho = 61.3$. The total solution time was 567.3 seconds using 16 processors of a Cray T3E. Using SPAI with mps preprocessing, we can solve LNS3937 in 660 iterations. The preconditioner contains 150270 nonzeros, and has a density $\rho = 5.9$, for a total solution time of 88.5 seconds on a Sun Ultra 5 workstation. Nevertheless, SPAI is outperformed by ILU(1), ILUT and AINV used in combination with mps. Furthermore, SPAI failed on GEMAT12

and CIRCUIT3 (with or without mps). Finally, we found that mps had a detrimental effect on the convergence rate obtained with SPAI applied to matrices SLIDE, TWO-DOM, and VENKAT25.

2.6 Conclusions

The experiments in this chapter illustrate that the reliability and performance of Krylov subspace methods preconditioned with standard incomplete factorizations can be dramatically enhanced by means of nonsymmetric permutations and scalings that place large entries on the main diagonal. This is also true for other preconditioning techniques, such as factorized sparse approximate inverses. The preprocessing phase is inexpensive, both in absolute terms and when compared to the total solution costs.

Of the heuristics considered in this research, the maximum product transversal algorithm gave the best results. With this preprocessing, many of the diagonal entries are large relative to the off-diagonal ones, and the matrix is closer to being diagonally dominant. This may have a stabilizing effect on the computation of the preconditioner, and also can speed up convergence rates. In combination with scalings, which often improve the conditioning of the problem, preconditioners based on drop tolerances (like ILUT and AINV) become reliable.

Much work remains to be done before preconditioned iterative methods can become a viable alternative to direct methods in areas such as chemical engineering, economics and management, circuit design, etc. Nevertheless, it is now at least conceivable to use iterative methods in such areas, and fair comparisons with direct solvers can be established. Our experiments suggest that nonsymmetric permutations and scalings can improve the performance of iterative solvers even in areas where these are already widely used, such as PDE problems.

Chapter 3

Preconditioning KKT Systems

3.1 Introduction

In this chapter, we solve real symmetric indefinite linear systems $\mathcal{H}x = b$, where

$$\mathcal{H} = \begin{bmatrix} H & B^T \\ B & 0 \end{bmatrix}, \tag{3.1}$$

via preconditioned Krylov subspace methods [43, 73]. Matrices of the form (3.1) are called KKT matrices, in reference to the Karush-Kuhn-Tucker first-order necessary optimality conditions for the solution of general nonlinear programming problems. KKT matrices arise in equality constrained nonlinear programming [39, 65], sparse optimal control [13], and mixed finite element discretization of partial differential equations [35, 69]. We assume $\mathcal{H}$ is nonsingular, $H \in \mathbb{R}^{n\times n}$ is nonsingular, symmetric and possibly indefinite, and $B \in \mathbb{R}^{m\times n}$, with $m \leq n$. Note that when $\mathcal{H}$ is nonsingular, B has full rank.

This paper describes three new preconditioners for KKT systems. The first two preconditioners exploit the structure of the KKT matrix; we approximate the con-

straint preconditioner

$$\mathcal{P} = \begin{bmatrix} I & B^T \\ B & 0 \end{bmatrix} \tag{3.2}$$

applied via a factorization of its inverse

$$\mathcal{P}^{-1} = \begin{bmatrix} I & -B^T \\ 0 & I \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & -(BB^T)^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -B & I \end{bmatrix}. \tag{3.3}$$

Our preconditioners approximate the solve with BB^T. We use a modification of the SAINV algorithm of Benzi, Cullum, and Tuma [4], to construct factors $\tilde{Z}$ and $\tilde{D}$ and approximate

$$\tilde{Z}\tilde{D}^{-1}\tilde{Z} \approx (BB^T)^{-1},$$

and we use a modification of the RIF algorithm of Benzi and Tuma [11], to construct factors $\tilde{L}$ and $\tilde{D}$ and approximate

$$\tilde{L}\tilde{D}\tilde{L}^T \approx BB^T.$$

The SAINV and RIF algorithms are robust, in that the factorizations are guaranteed to exist (in exact arithmetic) when the matrix (3.1) is nonsingular. We describe these approximations in Section 3.2, below. The RIF and SAINV algorithms are closely related, and they are do not break down in exact arithmetic for symmetric positive definite matrices. We modify SAINV and RIF for BB^T; the algorithms require only sparse matrix-vector products with the matrix B^T.

The third approach arises out of efforts to improve robustness for preconditioning general nonsymmetric, highly indefinite matrices, like the ones we examined in Chapter 2. When applied to KKT systems, with a large zero (2,2) block, standard preconditioning techniques such as LU factorizations and factorized approximate inverses can breakdown due to zero pivots. Dynamic pivoting strategies to avoid breakdown are examined in [71, 77], but still fail for many matrices [22]. In Chapter 2, permutations and scalings that place entries of large magnitude on the diagonal of a matrix

improve the robustness and effectiveness of standard preconditioning techniques. In this chapter, we combine these permutations with ILU preconditioning [72] for KKT systems. Our experiments show that this is an effective technique, albeit one that destroys the structure of the original system.

The chapter is organized as follows. In Section 3.2 we describe constraint preconditioners and their theoretical properties, and we describe how to approximate constraint preconditioners with factorizations of $(BB^T)^{-1}$ and BB^T. In Section 3.3 we describe algorithms for maximizing the product of entries on the diagonal of a matrix, as a preprocessing step for improving ILU preconditioning. In Section 3.4, we describe the test matrices used in our experiments. In Section 3.5, we compare exact constraint preconditioning to our approximate constraint preconditioners described in Section 3.2 and to standard ILU preconditioning combined with the permutations of Section 3.3.

3.2 Approximate Constraint Preconditioner

In this section, we describe the constraint preconditioners, and the permutations and scalings, applied to the KKT matrix. We apply an approximation of the constraint preconditioner (3.2) to a permuted and diagonally scaled KKT system $P^T D\mathcal{H}DPy = P^T Db$, with $x = DPy$. Our preconditioners are based upon factored approximations to $(BB^T)^{-1}$ and BB^T.

The factorization (3.3) shows that the application of the exact constraint preconditioner involves a matrix-vector product with a lower-triangular matrix, a solve with BB^T, and matrix-vector product with an upper-triangular matrix. The solve with BB^T is the most computationally expensive aspect of applying the preconditioner. For many of our problems, BB^T is denser than B (see Table 3.2), and m is approximately equal to n, that is B is nearly square and BB^T has dimensions approximately equal to B. For these reasons, we approximate BB^T and $(BB^T)^{-1}$ to speed up the

computation.

The approximation of $(BB^T)^{-1}$ is based on the algorithm SAINV [7, 4], which is formulated for general symmetric positive definite matrices. Our first preconditioner is

$$\mathcal{P}_{\text{SAINV}}^{-1} = \begin{bmatrix} I & -B^T \\ 0 & I \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & -\tilde{Z}\tilde{D}^{-1}\tilde{Z}^T \end{bmatrix} \begin{bmatrix} I & 0 \\ -B & I \end{bmatrix}, \tag{3.4}$$

where $\tilde{Z}D^{-1}\tilde{Z}^T \approx (BB^T)^{-1}$. $\tilde{Z}$ and $\tilde{D}$ are computed with the SAINV algorithm, and approximate the inverse of the exact L and D, respectively, in

$$LDL^T \;=\; BB^T \tag{3.5}$$

The approximation of BB^T is based on the related algorithm RIF [11], also formulated for general symmetric positive definite matrices. Our second preconditioner is

$$\mathcal{P}_{\text{RIF}} = \begin{bmatrix} I & \\ B & I \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & -\tilde{L}\tilde{D}\tilde{L}^T \end{bmatrix} \begin{bmatrix} I & B^T \\ & I \end{bmatrix}, \tag{3.6}$$

where RIF computes $\tilde{L}$ and $\tilde{D}$ to approximate L and D, respectively, in (3.5), so that $\tilde{L}\tilde{D}\tilde{L}^T \approx BB^T$. We describe our versions of the SAINV and RIF algorithms in Subsection 3.2.2, below.

3.2.1 Exact Constraint Preconditioning

The exact constraint preconditioner (3.2) can be effective for KKT systems [52, 59, 41, 69, 68, 70, 48]. We describe exact constraint preconditioning and give some theorems that justify using the preconditioner (3.2): constraint preconditioned GMRES converges in a small number of iterations. The convergence rate depends on a simple relation between the dimensions of the constraint matrix B. Exact constraint preconditioning has a serious drawback: as the size of the linear systems grows, the application of (3.2) can become costly [69]. In this section, we provide examples that demonstrate that this cost increases compared to the size of the matrix.

Convergence

Let $\mathcal{H} \in \mathbb{R}^{(n+m)\times(n+m)}$ be a symmetric indefinite, nonsingular matrix of the form

$$\mathcal{H} = \begin{bmatrix} H & B^T \\ B & 0 \end{bmatrix},$$

with $H \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{m\times n}$, and $m \leq n$. Recall that B then necessarily has full rank. Let $\mathcal{P} \in \mathbb{R}^{(n+m)\times(n+m)}$ be a symmetric indefinite, nonsingular preconditioner of the form

$$\mathcal{P} = \begin{bmatrix} I & B^T \\ B & 0 \end{bmatrix}.$$

Lemma 1 *Let $P = B^\dagger B$ be an orthogonal projector. Then*

$$\mathcal{P}^{-1}\mathcal{H} = \begin{bmatrix} (I - B^\dagger B)H + B^\dagger B & 0 \\ (B^\dagger)^T & I \end{bmatrix}.$$

Proof. The proof follows from

$$\mathcal{P}^{-1} = \begin{bmatrix} I - B^\dagger B & B^\dagger \\ (B^\dagger)^T & -(BB^T)^{-1} \end{bmatrix}.$$

□

The following theorem describes the clustering of eigenvalues due to constraint preconditioning, and is analogous to [52, Theorem 2.1].

Theorem 2 *$\mathcal{P}^{-1}\mathcal{H}$ has at least $2m$ eigenvalues equal to 1.*

Proof. Lemma 1 implies that the eigenvalues of $\mathcal{P}^{-1}\mathcal{H}$ are those of $(I - P)H + P$, where $P = B^\dagger B$ is an orthogonal projector. P has eigenvalue 1 with multiplicity m [63, p. 430]. We show that $(I - P)H + P$ also has eigenvalue 1 with multiplicity greater than or equal to m.

Let $(I-P)HPy = 0$, and hence $(I-P)HPPy = 0$. Since $\text{rank}(P) = m$, there are m linearly independent null vectors Py of $(I-P)H$. Hence, $Py = (I-P)HPy+Py =$

$[(I-P)H+P]Py$. That is, there are m linearly independent vectors Py that are eigenvectors of $(I-P)H+P$ associated with eigenvalue 1. Then $(I-P)H+P$ has eigenvalue 1 with multiplicity $\geq m$. □

Theorem 3 *If $\mathcal{P}^{-1}\mathcal{H}$ is diagonalizable, then any Krylov space of the linear system $\mathcal{P}^{-1}\mathcal{H}x = \mathcal{P}^{-1}b$ has dimension at most $n-m+1$.*

Proof. If a diagonalizable matrix $\mathcal{A}$ had d distinct eigenvalues, then any Krylov space for $\mathcal{A}x = b$ has dimension $\leq d$ [49].

Since $\mathcal{P}^{-1}\mathcal{H}$ is of order $n+m$ with at least $2m$ eigenvalues equal to 1, it has at most $n-m+1$ distinct eigenvalues. Hence, a Krylov subspace for $\mathcal{P}^{-1}\mathcal{H}x = \mathcal{P}^{-1}b$ has dimension at most $n-m+1$. □

While we do not know if the matrices in this research are diagonalizable, when they are, constraint preconditioned GMRES will converge in at most $n-m+1$ iterations. In Appendix A, we include theorems from [52] for the case where $\mathcal{H}$ is not assumed to be diagonalizable, in which case constraint preconditioned GMRES will converge in at most $n-m+2$ iterations.

Complexity

As problem size grows, the factorization and application of the preconditioner (3.2) becomes expensive. This was observed experimentally in [69]. Here, we examine the growth of fill for constraint preconditioners for a matrix from a standard sparse optimal control problem. The problem **brn201** (two-burn orbit transfer) describes the path of a spacecraft transferring from a low earth orbit to a mission orbit via two separate propulsion burns. To examine the reduction in sparsity in B and BB^T, we make the time discretization finer and finer, thereby increasing the size of the matrix (see Chapter 4). Figure 3.1 illustrates the increase in nonzeros for B and BB^T compared to the dimension of the matrix. The horizontal axis represents the row-dimension of B (which is nearly square). The vertical axis represents nonzeros.

The solid line represents a constant growth in nonzeros for brn201r1. The dotted line shows the nonzeros for the constraint portion B for each matrix. The dashed line shows the nonzeros for BB^T.

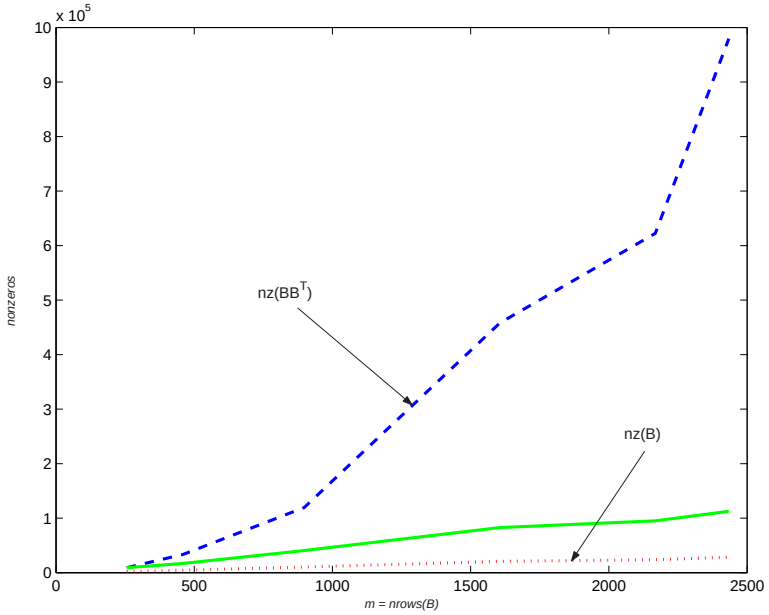


Figure 3.1: Increase for nonzeros for constraint matrix B and BB^T for two-burn orbit transfer problem **brn201**.

For the 6 matrices, Table 3.1 shows the number of nonzeros for the constraint matrix B compared to the number of nonzeros in the product BB^T. For the smallest matrix brn201r1, BB^T has four times as many nonzeros as B. For the largest matrix, brn201r6, BB^T has almost 40 times as many nonzeros as B.

The increase in nonzeros illustrated by Figure 3.1 and Table 3.1 translates directly to the Cholesky factors of BB^T and motivates our search for an effective incomplete factorization. In [69], the authors approximate the preconditioner with an incomplete QR factorization. However, unlike our factorizations, incomplete QR factorizations may break down [50, Appendix].

Table 3.1: Complete LU factorization fill for for two-burn orbit transfer problem **brn201**.

matrix	n	m	nz(B)	nz(BB^T)	ratio
brn201r1	278	256	2309	9184	3.98
brn201r2	484	460	4261	33450	7.85
brn201r3	927	898	10208	119500	11.71
brn201r4	1670	1606	20776	458056	22.05
brn201r5	2206	2168	23871	622260	26.07
brn201r6	2482	2434	28321	979576	38.59

3.2.2 Approximation of the Constraint Preconditioner

We approximate $(BB^T)^{-1}$ and BB^T by applying incomplete A-conjugation algorithms to BB^T, as described below. These factorizations are based on the algorithms SAINV [7, 4] and RIF [11], which were originally proposed for symmetric positive definite matrices. Our modifications construct approximations for BB^T, but require matrix-vector products with B^T only. If the KKT matrix (3.1) is nonsingular, the factorizations are guaranteed to exist in exact arithmetic.

We describe a variant of the SAINV algorithm for an efficient breakdown-free factored approximation of $(BB^T)^{-1}$. A robust factorized approximate inverse for general symmetric positive definite matrices is described in [4]; see also [54, 7, 9, 53].

SAINV for BB^T

The SAINV (Stabilized Approximate Inverse) algorithm [4] is an incomplete A-conjugation process. When applied to a real matrix BB^T, with $B \in \mathbb{R}^{m\times n}, m \leq n$, and B full rank, the *complete* A-conjugation algorithm produces a LDL^T factorization of $(BB^T)^{-1}$,

$$(BB^T)^{-1} = ZD^{-1}Z^T,$$

where Z unit is upper triangular and D diagonal with positive diagonal elements. The A-conjugation process is described in Algorithm 3.1 below, with e_i denoting the

ith unit basis vector.

Algorithm 3.1 Complete A-conjugation applied to BB^T

1: $z_i = e_i, \quad i = 1, \ldots, m.$
2: **for** $i = 1, \ldots, m$ **do**
3: **for** $j = i, \ldots, m$ **do**
4: $v_j = B^T z_j$
5: $p_j = v_i^T v_j$
6: **end for**
7: **for** $j = i + 1, \ldots, m$ **do**
8: $z_j = z_j - \frac{p_j}{p_i} z_i$
9: **end for**
10: $Z = [z_1, z_2, \ldots, z_m]$
11: $D = \mathrm{diag}(p_1, p_2, \ldots, p_m)$
12: **end for**

If $\mathcal{H}$ is nonsingular, then B has full rank, BB^T is symmetric positive definite, and

$$p_i = z_i^T BB^T z_i > 0 \quad \text{for} \quad z_i \neq 0. \tag{3.7}$$

Hence, there is no division by zero in exact arithmetic, and the resulting factorization is positive definite, provided the z_i's are non-zero. Indeed, element i of z_i is always 1:

Proposition 4 *Let $B \in \mathbb{R}^{m \times n}, m \leq n$ be full rank. Let Z, D be given by Algorithm 3.1. Then Z is unit upper triangular, and D is positive definite.*

Proof. The proof is elementary, and proceeds by induction.

Let $z_k^{(i)}$ denote the vector z_k at iteration i. At iteration i, the vectors $z_1^{(i)}$ through $z_i^{(i)}$ are already determined. The remaining vectors $z_j^{(i)}, j = i + 1, \ldots m$ are given by

$$z_j^{(i)} = z_j^{(i-1)} - \frac{p_j}{p_i} z_i^{(i-1)}. \tag{3.8}$$

We show that the ith element of z_i is always 1, and that elements $i+1$ through m of z_i are zero.

For $i = 1$, $z_1^{(1)} = e_1$, and $z_0^{(i-1)} = e_j, j = 2, \ldots, m$. In the update (3.8), only the first element of each of the z_j^0's is modified, and therefore element j of $z_j^{(1)}$ is equal to 1, and elements $j+1$ through m remain untouched and equal to zero.

We assume that the $(i-1)$th element of $z_j^{(i-1)}$ is always 1, and that elements $j+1$ through m elements of $z_j^{(i-1)}$ are zero.

At iteration i, $z_j^{(i)}, j = i+1, \ldots m$ are given by (3.8). By assumption, the jth element of $z_j^{(i-1)}$ is 1, and elements $i+1$ through m are zero. Therefore, only elements 1 through i of $z_j^{(i-1)}$ are modified. Since $j > i$ for all $j = i+1, \ldots m$, element j of $z_j^{(i)}$ is zero, and elements $j+1$ through m remain zero.

By induction, for all $i = 1, \ldots, m$, the ith element of z_i is always 1, and elements $i+1$ through m of z_i are zero. By the identity (3.7), $p_1, \ldots, p_m$ are positive. Therefore $Z = [z_1, \ldots, z_m]$ is unit upper-triangular, and $D = \text{diag}\,[p_1, \ldots, p_m]$ is positive definite. □

An incomplete A-conjugation process constructs a sparse $\tilde{Z}$ by dropping elements in Z. In the SAINV algorithm, small *off-diagonal* elements of z_j are dropped after the update in step 7 of Algorithm 3.1, with no a-priori sparsity pattern determined. When many entries in L^{-1} are small, the incomplete $\tilde{Z}^T$ is a good approximation to L^{-1}. Neither diagonal elements of Z nor pivots are dropped.

Algorithm 3.1 requires only matrix–vector products with B^T; one need not explicitly form BB^T. In step 5, for example,

$$\begin{aligned} p_j &= z_i^T (BB^T) z_j \\ &= (B^T z_i)^T B^T z_j. \end{aligned}$$

RIF for BB^T

We obtain an incomplete factorization

$$\tilde{L}\tilde{D}\tilde{L}^T \approx (BB^T)^{-1}$$

by applying the same incomplete A-conjugation process with only slight modifications. Incomplete factorizations exist for certain classes of matrices. In [62], an incomplete Cholesky factorization with arbitrary sparsity pattern was shown to exist for M-matrices, and in [60] the result was extended to H-matrices. However, for general symmetric positive definite matrices, general incomplete Cholesky factorizations can break down by encountering zero or negative pivots [50, Appendix]. Various breakdown-free incomplete factorizations of symmetric positive definite matrices have been proposed. For example, to avoid breakdowns, corrections to the diagonal entries of the original matrix can be made to ensure a positive definite factorization. Alternatively, incomplete orthogonalization can be implemented so as to lessen the occurrence of breakdown; for example, see [1, 51, 73, 76], or the summary given in [18]. We modify the RIF (Robust Incomplete Factorization) algorithm first proposed in [11], which is a process nearly identical to the SAINV algorithm.

The key to the relation between the RIF and SAINV algorithms lies in how the pivots are calculated, and in the fact that the Z matrices are upper-triangular. Recall that in the exact A-orthogonalization process (without dropping), the factors Z and D satisfy

$$ZD^{-1}Z^T = (BB^T)^{-1}.$$

In the factorization $LDL^T = BB^T$, we have $L^T = Z^{-1}$. Therefore,

$$D^{-1}Z^T BB^T = L^T$$

or

$$L = BB^T ZD^{-1}. \tag{3.9}$$

The matrix D is a diagonal matrix of the pivots $p_i, i = 1, \ldots, m$, in (3.7). Let b_i^T denote the ith row of B; then $b_i^T B^T$ is the ith row of BB^T. Because Z is unit upper triangular and $BB^T Z$ is lower triangular we can rewrite the pivots $p_i = z_i^T BB^T z_i$ as

$$p_i \;=\; b_i^T B^T z_i$$

where z_i denotes the ith column of Z. Likewise, we can write the intermediate p_j's

$$p_j \;=\; b_i^T B^T z_j, \quad j \geq i. \tag{3.10}$$

Let l_{ij} denote entry (i, j) of L. We can equate the (i, j) element of each side in (3.9)

$$l_{ij} \;=\; p_j^{-1} p_i \tag{3.11}$$

where $i \geq j$. The elements of L are calculated column-wise as the multipliers $l_{ji} = \frac{p_j}{p_i}$ in step 8 of Algorithm 3.1, at no extra cost.

There are now two choices for maintaining sparsity in the L factors. We can drop in the Z factors after updating, as before. But we can also implement a *post-filtration* strategy, dropping small elements in L after updating the vectors z_j in step 8 of Algorithm 3.1. This is post filtration, because once the multipliers that form the column of L are computed, they are not used again in the process. It has been observed in practice that an accurate and sparse factorization can be obtained using only the former dropping tolerance [11, 3]. We found in our experiments that a single drop tolerance is effective and simpler to manage, and thus we drop only in the z vectors, with no post-filtration.

3.2.3 Permutation and Diagonal Scaling of the KKT Matrix

The structure of BB^T is unaffected by reordering the columns of B: for any permutation matrix P, $BPP^T B^T = BB^T$. Also, the number of nonzeros in BB^T is unaffected by reordering the rows of B; however, we can reorder the rows of B

to produce a *symmetric* reordering $P^T BB^T P$. While the sparsity of $(BB^T)^{-1}$ is independent of the ordering of the rows and columns of BB^T, the sparsity of L^{-1} is highly dependent on the ordering. In practical implementations of SAINV, sparsity is preserved by combining the dropping of small elements with a sparse reordering such as multiple minimum degree [58], reverse Cuthill-McKee [28], or generalized nested dissection [57]. An ordering that limits sparsity in the inverse factors also limits the loss of information due to dropping in an incomplete conjugation process. Reorderings reduce construction and storage requirements for SAINV and improve the effectiveness of the preconditioner [12, 21]; in particular, minimum degree orderings are effective for SAINV. We apply a reordering P^T to the rows of B so that $P^T BB^T P$ has minimum degree ordering.

When the (1,1) block H of the KKT matrix is positive definite, then it has only positive diagonal elements (since $e_i^T He_i > 0, i = 1, \ldots, n$) and the identity matrix in the (1,1) block of the constraint preconditioner (3.2) may be replaced with $G = \text{diag}(H)$. This preconditioner is at least as good as the unscaled version with identity matrix in the (1,1) block [52]. For matrices H with positive diagonal elements, we describe an approach in which we diagonally scale H so that the scaled matrix has a unit diagonal in the (1,1) block. Note that diagonal scaling changes the B blocks also.

We also scale the (2,1) and (1,2) blocks of the KKT matrix. In [4], diagonal scaling improves the quality of the SAINV factorization for general symmetric positive definite matrices. We employ a similar scaling and guard against the impact on the B matrices of the scaling from the (1,1) block. We scale $\mathcal{H}$ so that BB^T also has unit diagonal, accounting for any scaling of the (1,1) block. We conducted our experiments both with and without this scaling, and found that the scaling produced preconditioners that converged faster than without the scaling.

To describe the scalings in detail, let b_i denote the ith row of the matrix B. Then

$$(BB^T)_{ii} = \|b_i\|_2^2, \qquad i = 1, \ldots, m.$$

The first block row and column of $\mathcal{H}$ are scaled by the $n \times n$ diagonal matrix D_H^{-1}, where

$$D_H = \begin{cases} \sqrt{\text{diag}(H)} & \text{if diag}(H) > 0, \\ I & \text{otherwise.} \end{cases}$$

The second block row and column of $\mathcal{H}$ are scaled by the $m \times m$ diagonal matrix D_B. Let $\bar{B}^T = D_H B^T = [\bar{b}_1, \ldots, \bar{b}_m]$, and define the $m \times m$ diagonal matrix D_B by

$$[D_B]_{ii} = \|\bar{b}_i\|_2^{-1}.$$

The scaling matrix for $\mathcal{H}$ is

$$D = \begin{bmatrix} D_H & 0 \\ 0 & D_B \end{bmatrix}.$$

The scaling $D\mathcal{H}D$ is equivalent to including $G = \text{diag}(H)$ in the (1,1) block of (3.2), and also ensures BB^T has a unit diagonal. That is, the scaled matrix $\bar{\mathcal{H}} = D\mathcal{H}D$ is

$$\begin{bmatrix} H_s & B_s^T \\ B_s & 0 \end{bmatrix} = \begin{bmatrix} D_H & 0 \\ 0 & D_B \end{bmatrix} \begin{bmatrix} H & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} D_H & 0 \\ 0 & D_B \end{bmatrix}, \tag{3.12}$$

where $\text{diag}(H_s) = I$ and $\text{diag}(B_s B_s^T) = I$.

3.3 Permutations for ILU Preconditioners

In this section, we describe permutations and scalings that maximize the product of the diagonal entries of a matrix. This technique deviates from the preconditioners in the previous sections because it ignores the structure of $\mathcal{H}$, and it applies to general non-symmetric indefinite matrices.

ILU preconditioners can breakdown, unless the matrix in question has special properties, such as when it is an H-matrix. In particular, ILU preconditioners are

guaranteed to exist for diagonally dominant matrices in exact arithmetic. Very small or zero pivots can lead to instabilities during the construction of ILU preconditioners; instabilities associated with ILU preconditioners are described in detail in [22]. Because KKT matrices have a large zero block on the diagonal, such instabilities are likely to occur when constructing ILU preconditioners for KKT matrices.

Chapter 2 illustrates that, for general nonsymmetric and highly indefinite systems, nonsymmetric permutations that place large entries on the diagonal of a matrix

- allow breakdown-free construction of standard ILU preconditioners, and
- improve the convergence of preconditioned iterative solvers without significantly increasing the cost of constructing the preconditioner.

The Maximum Product with Scalings (MPS) permutes and scales the matrix so that the diagonal contains elements of absolute value one and the off–diagonal elements are less than one in absolute value. This algorithm is part of the FORTRAN subroutines MC64 [30, 31] from the Harwell Subroutine Library. The strategy was introduced in [66] for pivoting in dense Gaussian elimination. Permuting and scaling a matrix with MPS improves the convergence of ILU preconditioners for almost every matrix examined in Chapter 2. The other one-sided permutations also improve the convergence of the iterative solvers, but are not as effective as the MPS permutations. Chapter 2 also demonstrates that the MPS orderings are effective in combination with sparsity-preserving symmetric reorderings. In this chapter, we apply ILU preconditioners to *KKT systems*, where $\mathcal{H}$ is permuted and scaled with MPS, and also permuted symmetrically to reduce fill. That is, we apply ILU preconditioners to permuted KKT systems

$$(P^T Q D_1 \mathcal{H} D_2 P) y = P^T Q D_1 b, \tag{3.13}$$

where the original system is $\mathcal{H}x = b$; P and Q represent symmetric and one-sided permutation matrices, respectively; D_1 and D_2 are diagonal scaling matrices; and the

solution is given by $x = D_2 P y$. The preconditioner is an incomplete LU factorization of $P^T Q D_1 \mathcal{H} D_2 P$.

3.4 Description of Test Problems

In this section, we describe the matrices used in our numerical experiments. The matrices listed in Table 3.2 have the KKT structure

$$\mathcal{H} = \begin{bmatrix} H & B^T \\ B & 0 \end{bmatrix}.$$

The matrices come from three sources. The first six matrices are *stiffness matrices* and *mass matrices* from mixed finite element approximations of variational problems [52]. The next three matrices come from *convex quadratic programming problems* from the constrained and unconstrained testing environment (CUTE) test suite [20]. These nine matrices were also examined in [52], and the (1,1) block in these cases is positive definite.

Our third set of matrices comes from *sparse optimal control* [13]. The matrices were produced with SOCS (Sparse Optimal Control Software), commercial software developed by the Boeing Company. The matrices were generated at iteration 2 of a sequential quadratic programming method, applied to the nonlinear programming problem derived from the sparse optimal control problem. For these problems, the (1,1) block H represents an approximate Hessian of the Lagrangian for a nonlinear optimization problem and is not guaranteed to be definite; the B blocks correspond to constraints.

SOCS currently solves the linear systems with a highly–tuned multifrontal method. Dynamic pivoting improves the stability of the algorithm. For many matrices, the factors incur large amounts of fill, necessitating the use of secondary storage. This motivates the inclusion of sparse optimal control problems in this study.

Table 3.2: Description of KKT Test Problems

Application Area: Mixed Finite Element							
Matrix	**N**	**m**	**n**	**nz**($\mathcal{H}$)	**nz**(H)	**nz**(B)	**nz**(BB^T)
stiff1ns	610	32	578	3168	2316	426	238
stiff2ns	2306	128	2178	13480	9740	1870	1166
stiff3ns	8686	236	8450	47146	39948	3599	1024
mass01ns	610	32	578	3980	3128	426	238
mass02ns	2306	128	2178	17140	13400	1870	1166
mass03ns	8686	236	8450	62646	55448	3599	1024
Application Area: Quadratic Programming							
Matrix	**N**	**m**	**n**	**nz**($\mathcal{H}$)	**nz**(H)	**nz**(B)	**nz**(BB^T)
bloweyqp	1504	502	1002	8010	3004	2503	2508
cvxqp1qp	200	100	100	1262	672	295	1328
mosqp2qp	960	30	930	1316	1020	148	144
Application Area: Sparse Optimal Control							
Matrix	**N**	**m**	**n**	**nz**($\mathcal{H}$)	**nz**(H)	**nz**(B)	**nz**(BB^T)
brn203r1	514	256	258	6982	2436	2273	9380
brn201r1	534	256	278	7044	2426	2309	9184
brn201r2	944	460	484	12772	4250	4261	33450
brn201r3	1825	898	927	28435	8019	10208	119500
brn201r4	3276	1606	1670	56768	15216	20776	458056
brn201r5	4374	2168	2206	66270	18528	23871	622260
brn201r6	4916	2434	2482	78100	21458	28321	979576
traj01r1	401	185	216	4502	2024	1239	6053
traj03r1	388	185	203	4603	1457	1573	6415
traj05r1	403	185	218	7260	3698	1781	6435
mirv01r1	2397	1151	1246	31520	9826	10847	240469
capt09r1	2063	1019	1044	22942	7064	7939	73907
gsoc01r1	1376	602	774	20144	7016	6564	48912
putt01r1	114	56	58	924	290	317	1552
lwbr01r1	2109	1014	1095	22545	6843	7851	81748

3.5 Numerical Experiments

In this section, we compare three preconditioning techniques for KKT systems. All tests were performed on a Sun Ultra-4 SPARCstation with 2048 MB physical RAM running Sun OS 5.6. Algorithms were implemented using MATLAB version 5.3. We measure iteration counts, and we also count additional nonzeros that must be stored in the incomplete factors. Due to the occasional use of built-in MATLAB commands, and we do not measure CPU time, as to avoid comparing compiled built-in MATLAB codes to interpreted MATLAB scripts. Instead, we further measure computational costs by counting floating point operations using the MATLAB command `flops`.

Constraint preconditioning (**CP**) represents our benchmark, where the preconditioner (3.2) is applied via the factorization

$$\mathcal{P} = \begin{bmatrix} I & \\ B & I \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & -BB^T \end{bmatrix} \begin{bmatrix} I & B^T \\ & I \end{bmatrix}. \tag{3.14}$$

The application of (3.14) requires multiplication by two triangular systems, since

$$\begin{bmatrix} I & B^T \\ & I \end{bmatrix}^{-1} = \begin{bmatrix} I & -B^T \\ & I \end{bmatrix},$$

and a solve with BB^T. To perform the solve, we compute a Cholesky factor R for BB^T via the built-in MATLAB command `R = qr(B')`. The rows of B are reordered so that BB^T has MMD ordering.

When H is nonsingular and $\mathcal{H}$ is preconditioned by a matrix $\mathcal{P}$ of the form (3.2), GMRES converges (in exact arithmetic) in at most $n - m + 2$ iterations, where $H \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{m\times n}$ [52, Theorem 3.5].

CP—SAINV refers to our approximation (3.4) of the exact constraint preconditioner, where we approximate $(BB^T)^{-1}$ by the SAINV factorization, with a drop tolerance of 0.1. The rows of B are reordered so that BB^T has MMD ordering.

CP—RIF refers to our approximation (3.6) of the exact constraint preconditioner, where we approximate BB^T by a RIF factorization. We use a drop tolerance

of 0.1 for the Z factors and no a posteriori dropping in the L factors. Again, the rows of B are reordered so that BB^T has MMD ordering.

For CP, CP—SAINV, and CP—RIF, we scale the matrix as in (3.12). That is, for the mixed finite element and quadratic programming matrices, we diagonally scale the coefficient matrix so that H and BB^T have unit diagonal. For the sparse optimal control problems, H does not necessarily have positive diagonal, and we only scale BB^T to have a unit diagonal.

MC64—ILU (permuting large entries to the diagonal) represents a more general approach that disregards any structure in the coefficient matrix. We apply the permutations described in Section 3.3 and then construct a standard ILU factorization with the built-in MATLAB command `luinc()`. We use a drop tolerance of 10^{-2}, no diagonal modifications, and no pivoting. To avoid breakdown, we replace zero pivots (diagonal U entries) by the local drop tolerance. Breakdown occurs in 3 cases, which we discuss below. Since the reverse Cuthill-McKee (RCM) ordering can increase the accuracy of ILU preconditioners [8], we use this symmetric ordering after applying the one-sided permutations and scalings to improve stability.

The iterative method is full GMRES with left preconditioning in order to compare the preconditioning approaches under a norm-minimizing method. The initial guess is the zero vector. Right-hand sides are constructed from the solution $[1, 2, \ldots, N]^T$, where $N = n + m$ represents the dimension of the matrix $\mathcal{H}$. Iterations are terminated when the 2-norm of the relative residual was reduced by at least six orders of magnitude, or when a maximum of 200 iterations was completed.

Tables 3.3 and 3.4 report our experiments. For each preconditioner, we list GMRES iteration counts to convergence; "nc" indicates no convergence within 200 iterations. "bd" indicates breakdown in preconditioner construction. We report the following nonzero ratios:

- For **CP**, $\rho = \text{nnz}(R)/\text{nnz}(B)$ from the Cholesky decomposition $RR^T = BB^T$.

In general, R is much denser than B. $\rho_1 \approx 1.0$ represents an extremely sparse factorization.

- For **CP—SAINV** $\rho = \text{nnz}(\tilde{Z})/\text{nnz}(B)$ where $\tilde{Z}\tilde{D}\tilde{Z}^T \approx (BB^T)^{-1}$, and $\tilde{Z}, \tilde{D}$ are computed with SAINV.

- For **CP—RIF** $\rho = \text{nnz}(\tilde{L})/\text{nnz}(B)$ where $\tilde{L}\tilde{D}\tilde{L}^T \approx BB^T$, and $\tilde{L}, \tilde{D}$ are computed with RIF.

- For **MC64–ILU**, $\rho = \text{nnz}(\bar{L} + \bar{U})/\text{nnz}(\mathcal{H})$, where let $\bar{L}$ and $\bar{U}$ denote the LU factors for **MC64–ILU**. That is, ρ is the ratio of the total number of nonzeros in the incomplete factors of the scaled and permuted matrix $P^T Q D_1 \mathcal{H} D_2 P$ to the number of nonzeros in the original matrix $\mathcal{H}$.

We also report total floating point operations for the preconditioner construction and GMRES solve phases, as returned by the MATLAB command `flops`. The MC64 permutations and scalings were calculated via calls to MATLAB Fortran MEX files, and the routines contain no facility for counting floating point operations. However, as shown in Chapter 2, the computation of the permutations and scalings are cheap compared to preconditioner construction and solve costs.

In Table 3.3, we report our experiments with the mixed finite element and quadratic programming matrices. For each preconditioner, GMRES converges within 200 iterations.

CP has the lowest iteration count among the constraint preconditioners (as expected); the iteration count also falls within the theoretical prediction of $n - m + 2$. The Cholesky factor R for BB^T is reasonably sparse for all problems. CP produces the lowest flop counts for the mass matrices and the matrices bloweyqp and cvxqp1qp.

CP—SAINV produces equal or higher iteration counts, but for four problems (stiff1ns, stiff3ns, mass03ns, and mosqp2qp), CP—SAINV produces the sparsest factors while requiring less than 25% additional iterations compared to CP. Flop counts

for these matrices are also low.

CP—RIF performes almost as well as CP for most problems, often with sparser factors. For three problems (stiff1ns, stiff3ns, and mass03ns), CP—RIF produces sparser factors and requires less than 25% additional iterations compared to CP.

MC64—ILU has the lowest iteration count for seven of the nine problems. However, except for cvxqp1qp, the factors are more dense. Often, the factors are more than twice as dense. For the stiffness matrices, the iteration counts are significantly lower than for the other preconditioners. Likewise, for the stiffness matrices and mosqp2qp, MC64—ILU is the fastest in terms of floating point operations.

Table 3.3: Comparison of preconditioners for mixed finite element and quadratic programming problems.

	CP			**CP—SAINV**			**CP—RIF**			**MC64—ILU**		
matrix	its	ρ	flops	its	ρ	flops	its	ρ	flops	its	ρ	flops
stiff1ns	40	0.39	4.2e+06	47	0.35	5.6e+06	42	0.38	4.6e+06	8	3.35	8.4e+05
stiff2ns	72	0.68	4.5e+07	77	1.29	7.7e+07	73	0.68	4.9e+07	16	4.59	9.3e+06
stiff3ns	184	0.44	9.6e+08	184	0.24	9.7e+08	184	0.33	9.7e+08	24	2.83	3.5e+07
mass01ns	10	0.39	5.6e+05	12	0.65	9.4e+05	11	0.39	7.2e+05	5	2.22	5.7e+05
mass02ns	10	0.68	2.4e+06	16	1.29	1.3e+07	12	0.68	6.1e+06	7	2.56	4.2e+06
mass03ns	8	0.44	6.2e+06	10	0.24	1.8e+07	8	0.33	1.0e+07	4	1.79	6.6e+06
bloweyqp	4	0.80	5.4e+05	6	25.37	3.0e+08	4	0.80	8.4e+07	7	16.25	8.5e+07
cvxqp1qp	2	4.31	8.1e+04	38	5.28	6.0e+06	23	4.32	1.2e+06	3	2.35	8.5e+04
mosqp2qp	6	0.59	3.1e+05	6	0.40	3.0e+05	6	0.59	3.2e+05	3	1.82	1.6e+05

In Table 3.4, we report our experiments preconditioning sparse optimal control problems. Exact constraint preconditioning **CP** solves every problem. However, the factors are often quite dense; for four matrices (mirv01r1, brn201r4, brn201r5, brn201r6), the R factor for BB^T is over ten times as dense as B. This is due to dense blocks in BB^T, caused by coupling between phases and dense columns in B. See Subsection 3.5.1 and Chapter 4, where we examine these matrices further.

CP-SAINV fails to converge for most matrices, and when convergence occurs, the iteration count is high. **CP-RIF** fares somewhat better, in that more problems are solved; however, the incomplete L factor for BB^T is very dense, as the large ratio's

ρ show. The factors are as dense as those in CP. However, the high iteration counts indicate that the loss of information due to the dropping in the Z factor is severe.

MC64-ILU converges for most matrices. For five matrices, (brn201r4, traj03r1, traj05r1, mirv01r1, and gsoc01r1), MC64-ILU produces the lowest iteration counts. For ten of the 15 problems, flop counts are lowest. In contrast to the problems in Table 3.3, for every problem that MC64-ILU solves, the LU factors are very sparse, usually only slightly more dense than the original matrix $\mathcal{H}$. The factorization breaks down for capt09r1. Without the use of zero replacement in the factorization, the factorization also breaks down for the trajectory matrices.

Table 3.4: Comparison of preconditioners for sparse optimal control problems.

	CP			**CP—SAINV**			**CP—RIF**			**MC64—ILU**		
matrix	its	ρ	flops	its	ρ	flops	its	ρ	flops	its	ρ	flops
brn201r1	10	2.17	1.2e+06	74	3.62	1.8e+08	44	2.17	1.3e+07	16	0.98	1.5e+06
brn201r2	7	4.14	2.4e+06	181	3.11	1.0e+09	66	4.13	4.4e+07	13	1.00	2.1e+06
brn201r3	9	6.47	1.1e+07	152	2.90	1.8e+09	82	6.29	1.8e+08	20	1.88	1.2e+07
brn201r4	62	13.55	1.5e+08	nc	—	—	nc	—	—	24	1.23	2.7e+07
brn201r5	10	13.84	5.3e+07	nc	—	—	nc	—	—	21	1.42	2.8e+07
brn201r6	11	17.99	7.2e+07	nc	—	—	nc	—	—	24	1.67	4.8e+07
brn203r1	4	2.25	5.9e+05	75	1.05	3.8e+07	43	2.24	9.1e+06	6	0.79	5.1e+05
traj01r1	10	2.62	7.5e+05	84	4.26	1.0e+08	30	2.63	6.1e+06	13	1.26	8.5e+05
traj03r1	15	2.19	1.1e+06	75	2.92	7.7e+07	34	2.19	7.2e+06	9	1.25	5.8e+05
traj05r1	11	1.94	1.0e+06	70	2.60	7.5e+07	29	1.94	7.2e+06	10	0.84	8.2e+05
mirv01r1	24	11.16	3.5e+07	nc	—	—	nc	—	—	17	1.94	1.0e+07
capt09r1	4	4.77	3.1e+06	nc	—	—	151	4.76	3.4e+08	bd	—	—
gsoc01r1	21	3.83	8.5e+06	37	2.97	3.3e+08	28	3.83	5.9e+07	12	3.01	8.5e+06
putt01r1	4	3.53	8.7e+04	86	2.02	7.2e+06	26	2.84	6.8e+05	5	1.41	7.9e+04
lwbr01r1	9	6.58	9.5e+06	nc	—	—	108	6.54	3.5e+08	22	3.95	1.8e+07

3.5.1 Further Analysis of Results

In this subsection, we examine two matrices to understand the differences in performance. The approximate constraint preconditioners work well for the mixed finite element and quadratic programming matrices, but fail or work poorly for the sparse

optimal control problems.

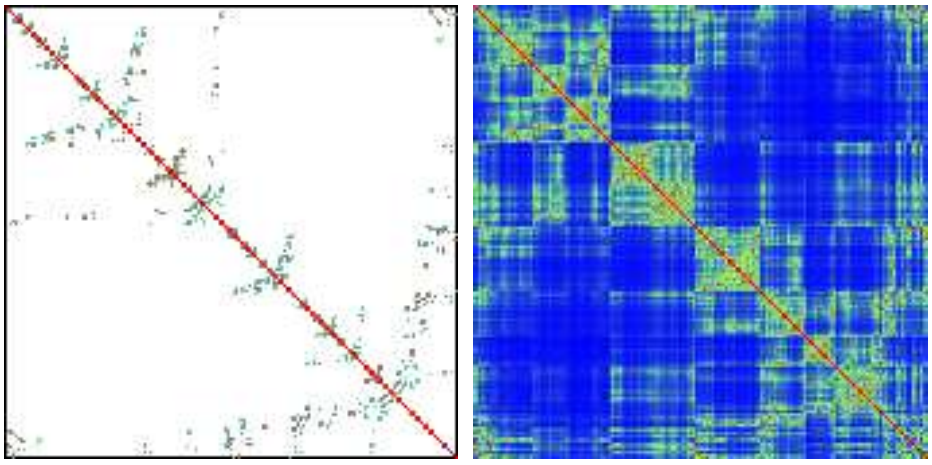
When many of the entries of an inverse are small in absolute value, such as when the magnitude of the entries decays away from the diagonal, a sparse approximate inverse can be accurate. However, if there is little decay, and large entries are scattered throughout the inverse, then sparse approximate inverses are not as accurate [10].

We explicitly formed BB^T and $(BB^T)^{-1}$ for two matrices in our test set. The nonzeros of the matrices are shown in Figures 3.2 and 3.3; zero entries are white. Viewed in color, the magnitude of the entries is represented over a spectrum, with red denoting entries large in absolute value, and blue denoting entries near zero. In a greyscale, lighter shades represent large magnitudes, while darker shades represent entries near zero; white again represents exact zeros.

Figure 3.2, shows the mixed finite element matrix **stiff3ns**, for which the approximate-inverse based algorithms converge fast. The left portion of the figure is BB^T, where the rows of the constraint matrix B are ordered so that BB^T has minimum degree ordering. The right portion is the inverse $(BB^T)^{-1}$, with large entries on or clustered near the diagonal. Most of the remaining entries in the matrix are small, and hence the SAINV and RIF factorizations perform well.

Figure 3.3 shows the sparse optimal control matrix **traj01r1**, for which the approximate-inverse based algorithms did not performed well. Again, the left portion of the figure is BB^T, where the rows of the constraint matrix B are ordered so that BB^T has minimum degree ordering. The inverse $(BB^T)^{-1}$ displayed on the right has an even distribution of relatively large values. The inverse exhibits no decay, and as expected, the approximate inverse-based techniques perform poorly.

The large dense blocks of BB^T in Figure 3.3 occur frequently in sparse optimal control problems, and arise from dense columns in B and coupling between phases (see Chapter 4). A complete factorization produces a lot of fill, since the portions of the factors corresponding to these blocks are dense. We discuss this further in Chapter 4.



BB^T $(BB^T)^{-1}$

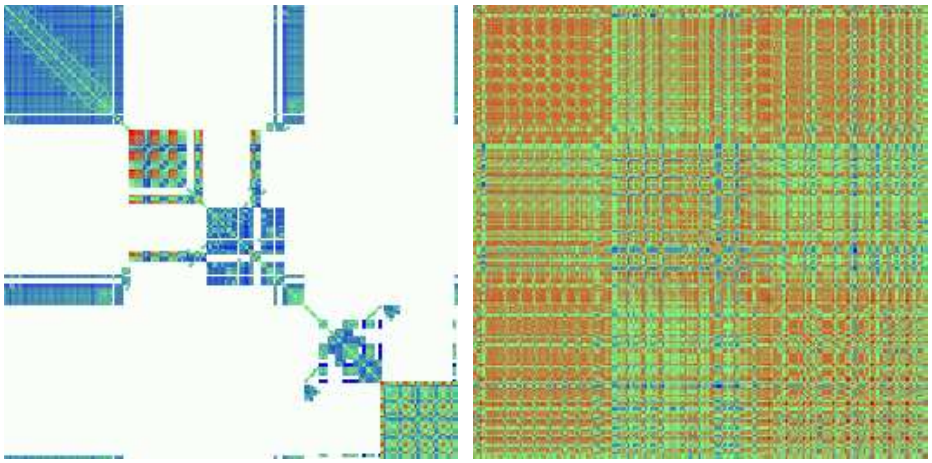
Figure 3.2: Decay away from the diagonal for mixed finite element matrix **stiff3ns**.

3.6 Conclusions

We compare three approaches for preconditioning KKT systems $\mathcal{H}x = b$ to exact constraint preconditioning. Two approximate constraint preconditioners in factored form approximate the products $(BB^T)^{-1}$ and BB^T, and scale the matrices so that BB^T has unit diagonal. A third approach applies nonsymmetric permutations and scalings to place large entries on the diagonal of the matrix, and constructs standard ILU preconditioners.

Constraint preconditioners are effective for KKT systems. Our experiments show that approximate constraint preconditioners with factorized approximate inverses or incomplete factorizations, can outperform exact constraint preconditioning, because they reduce the number of nonzeros in the preconditioner factors, without significantly increasing iteration counts. However, the preconditioning often fails for sparse optimal control problems.

We also experiment with permutations and scalings that maximize the product of the entries along the diagonal of the matrix, in combination with a standard ILU



BB^T $(BB^T)^{-1}$

Figure 3.3: Lack of decay for sparse optimal control matrix **traj01r1**.

preconditioner. While this approach ignores the structure of the original system, it results in fast convergence. For most matrices, we are able to construct an incomplete LU factorization and solve the systems with very sparse factors. However, sometimes the factorizations break down, despite the permutations and scalings.

For the sparse optimal control problems, approximate constraint preconditioning converges slowly, and the preconditioner factors for the exact constraint preconditioner are very dense. In Chapter 4, we examine these problems more closely and construct approximate constraint preconditioners using knowledge of problem dependencies and discretization strategies.

Chapter 4

Preconditioning Linear Systems from Sparse Optimal Control

4.1 Introduction

Constrained optimal control problems involve the minimization of an objective function related to a dynamical system, subject to constraints on the variables. The dynamical system is described by a system of ordinary differential equations (ODE's). The solution may be additionally constrained by simple bounds on variables and path constraints. Sparse optimal control problems represent an extension of nonlinear programming (NLP) problems to an infinite number of variables. In order to solve optimal control problems numerically, the ODE's are discretized, resulting in a finite dimensional problem.

The finite–dimensional optimization problem can be solved by nonlinear programming methods. The Boeing Company has developed SOCS (Sparse Optimal Control Software), a collection of software tools that solve optimal control problems in this manner. SOCS transforms the optimal control problem to a large, sparse nonlinear program, which is solved via a sequential quadratic programming method or an in-

terior point (barrier) method. The solution of the nonlinear program requires the solution of a KKT linear system

$$\begin{bmatrix} H & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} -p \\ \bar{\lambda} \end{bmatrix} = \begin{bmatrix} g \\ c \end{bmatrix}. \tag{4.1}$$

SOCS currently solves these systems with a direct multifrontal solver. However, when the matrices are large, a large amount of fill is created in the factorization of the coefficient matrix, requiring secondary storage. In order to solve linear systems with million of unknowns, Boeing would like to use iterative methods, which can require less storage. For large problems, iterative methods can be faster than direct methods, provided effective preconditioners are available.

At this point, few preconditioners for sparse optimal control problems have been developed. This chapter contributes two new preconditioners for the linear systems (4.1) solved in SOCS. The preconditioner uses knowledge of the underlying problem and the numerical formulation. We discuss solution of the linear systems when the nonlinear program is solved with sequential quadratic programming. However, the linear systems solved when the barrier method is used have similar structure, and the fundamental ideas of our preconditioners can be applied to those problems.

This chapter is organized as follows. Section 4.2 introduces the sparse optimal control problem. We describe our preconditioner for sparse optimal control problems in Section 4.3. We give general descriptions of the optimal control problems used in our experiments in Section 4.4. In Section 4.5, we present numerical experiments with several sparse optimal control problems.

4.2 The Sparse Optimal Control Problem

In this section, we describe the optimal control problem. For a detailed description, see [13]. The problem is formulated as a collection of phases, where within each

phase, the dynamics of a system are described by the dynamic state variables $y(t)$ and dynamic control variables $u(t)$. $t_I \leq t \leq t_F$ is an independent variable, which we assume represents time, and we assume that the phases are sequential. t_I and t_F are initial and final times, respectively, which may themselves be allowed to vary. The phases are linked by boundary conditions ψ that relate the values of the dynamic variables at the beginning and end of each phase.

The dynamics are explicitly defined by a set of ordinary differential equations (ODE's), the *state equations*

$$\dot{y} = f(y(t), u(t), p, t), \tag{4.2}$$

which depend on the state and control variables, t, and time-independent parameters p. A solution is required to satisfy *path constraints* and simple bounds on state and control variables.

A simple optimal control problem can be stated as follows: find the control vectors u and parameters p that minimize an objective function ϕ subject to the state equations, path constraints, and simple bounds. The objective function ϕ depends on the values computed at each phase.

4.2.1 Transcription

The infinite-dimensional problem can be converted to a finite-dimensional problem by a process called *transcription*, which divides each phase into intervals

$$t_I = t_1 < t_2 < \ldots < t_M = t_F$$

The M t_i's are referred to as grid or mesh points within a particular phase.

To treat the values of the state and control variables as a set of NLP variables, we replace the differential state equations by a set of defect constraints, by discretizing the state equations. Further constraints are imposed by enforcing boundary conditions

directly at the grid points. A finite difference scheme or some other method can then be used to approximate the Jacobian and Hessian.

Solving the sparse optimal control problem thus involves transcribing the continuous optimal control problem into a finite–dimensional optimization problem, and solving the optimization problem. SOCS estimates the accuracy of the finite-dimensional solution with respect to the continuous problem. If the error for the discrete method is too large relative to the order of the discretization scheme, then SOCS constructs a new refined mesh. The process for computing the error and refining the mesh is very involved; for a details, see [17, 15, 13]. When the mesh is refined, the number of variables grows and the matrices become larger.

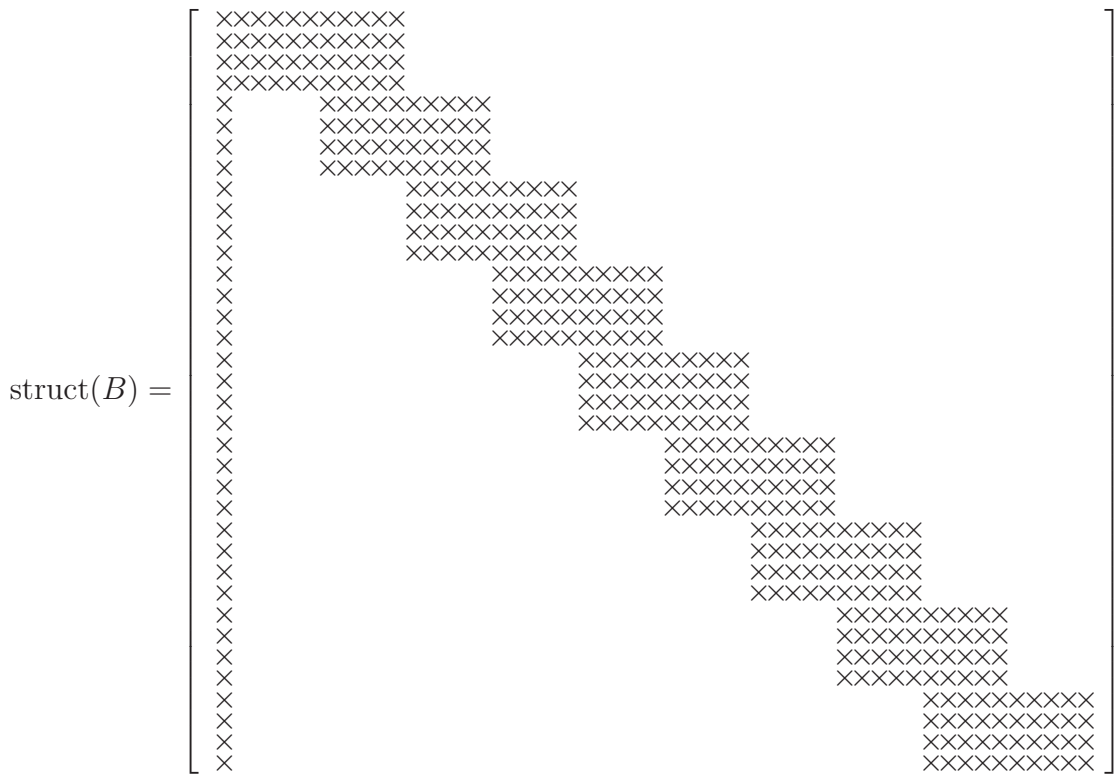


Figure 4.1: Typical Jacobian structure for a single phase.

Figure 4.1 shows a typical structure for the Jacobian of the constraint matrix

for the KKT system that must be solved, and the structure is common to most of the KKT systems from optimal control. The problem that generated this example involved a variable final time, resulting in the dense first column. In general, when initial or final times are allowed to vary, or when time-independent parameters are present, all variables and constraints in the problem will depend on these time and parameter variables. However, the remaining columns in the Jacobian each have only a few nonzeros in consecutive rows. Because the Hessian is approximated by a finite difference scheme, it will also have a similar structure. We describe in Section 4.3 how we take advantage of the structure present in the Hessian and Jacobian in constructing sparse accurate preconditioners.

In general, the structure of the Hessian is determined by dependencies among and between differential variables on algebraic variables, and the structure of the Jacobian is governed by the dependence of "differential" and "algebraic" constraints (that is, the constraint equations derived from differential and algebraic variables) on variables at other grid points.

With few exceptions, nearly all the variables and equations within a phase are "locally dependent," that is, they depend only on variables at nearby grid points. The sparsity of the resulting matrix reflects this with a banded structure. For the constraint portion of the matrix, this is most often exhibited as block bi-diagonal, since most discretization schemes produce constraints that, at node k, depend only on information at node k and $k+1$. Furthermore, because the phases are sequential, linkage boundary conditions depend principally on final time variables from the previous phase and initial time variables in the current phase.

As noted above, the exception to this local dependence is on the time independent parameters and free initial or free final times. Each row of the matrix represents an equation or variable at a *time* grid point, but since the parameters are time-independent, each variable and equation must account for every parameter. Likewise, when the initial or final times are allowed to be free, every mesh point then becomes

dependent upon the free time variables, and dense columns and rows result. We refer to time independent parameters and free initial and final times as *global variables*. When there are no global variables, we say the matrix is *locally dependent*.

We are ultimately interested in how these dependencies affect the nonzero structure of the resulting KKT matrix. Figure 4.2 presents a detailed sparsity pattern for the Jacobian from a problem constructing an optimal trajectory for an aircraft. The problem is broken into five phases. The dense columns at the beginning of each

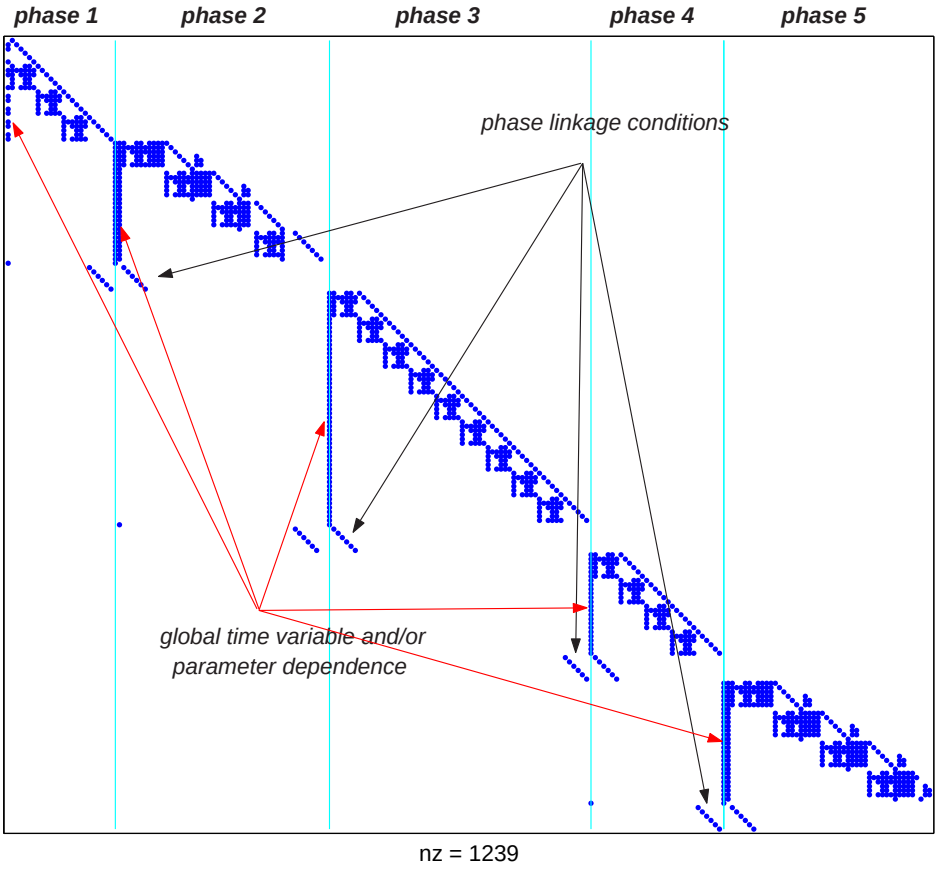


Figure 4.2: Constraint matrix detailing local and global dependence.

phase show the global dependence on time variables and parameters; at the end of each phase, phase linkage conditions couple consecutive phases. We discuss in the

next section how to handle these dense columns and phase couplings.

4.3 A Preconditioner for Sparse Optimal Control

In this section, we describe two preconditioners for KKT matrices

$$\mathcal{H} = \begin{bmatrix} H & B^T \\ B & 0 \end{bmatrix},$$

where $H \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{m\times n}, m \leq n$.

The motivation for our preconditioners is the *exact* constraint preconditioner described in Chapter 3,

$$\mathcal{P} = \begin{bmatrix} I & B^T \\ B & 0 \end{bmatrix}, \tag{4.3}$$

which is applied via the factorization

$$\begin{bmatrix} I & B^T \\ B & 0 \end{bmatrix} = \begin{bmatrix} I & \\ B & I \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & -BB^T \end{bmatrix} \begin{bmatrix} I & B^T \\ & I \end{bmatrix}. \tag{4.4}$$

As noted in Chapter 3, the principal cost in applying (4.3) is the solve with BB^T. Consider the typical B block shown in Figure 4.1 for single-phase sparse optimal control problems. A close examination of the structure reveals that BB^T is dense. For the $m \times n$ constraint matrix $B = [B_{ij}]$, the (i, j) element of BB^T is given by

$$\begin{aligned} \left[BB^T\right]_{ij} &= \sum_{k=1}^{n} B_{ik}B_{kj} \\ &= B_{i1}B_{1j} + \sum_{k=2}^{n} B_{ik}B_{kj}. \end{aligned}$$

The first column of B is dense, and hence without cancellation all elements in BB^T will be nonzero. For a multiphase problem, the matrix contains dense blocks along the diagonal. The sparsity of BB^T is independent of the ordering of the rows or columns of B.

As described in the previous section, the presence of global variables, that is independent variables upon which every equation and variable in a phase depend, result in dense columns. For example, if a phase is modeled with a free final time and s independent parameters, the corresponding blocks of the Hessian and Jacobian will have $s + 1$ dense columns.

4.3.1 Locally Dependent Preconditioners

Our preconditioners

- remove global dependencies from the KKT matrix, and
- reorder the equations in the Jacobian portion of the matrix so that local dependence is preserved.

Our first preconditioner is

$$\mathcal{P}_1 = \begin{bmatrix} I & \tilde{B}^T \\ \tilde{B} & 0 \end{bmatrix}, \tag{4.5}$$

where $\tilde{B} \approx B$ preserves sparsity in the factorization focusing in particular on $\tilde{B}\tilde{B}^T$. We will refer to the preconditioner (4.5) as a *Local Constraint Preconditioner* (**Local–CP**).

To see how we construct $\tilde{B}$, consider the example presented in Figure 4.2. Denote the Jacobian matrix as B. The left half of Figure 4.3, shows the sparsity structure of BB^T. The right half shows the sparsity structure of R from a Cholesky factorization $RR^T = BB^T$. The dense columns of B result in dense diagonal blocks in BB^T for each phase. As the figure illustrates, the Cholesky factors are at least as dense. The experiments on sparse optimal control problems in Chapter 3 illustrate this phenomenon. The density of the factorized constraint preconditioner results in high computational costs, both in the construction and application of the factorization

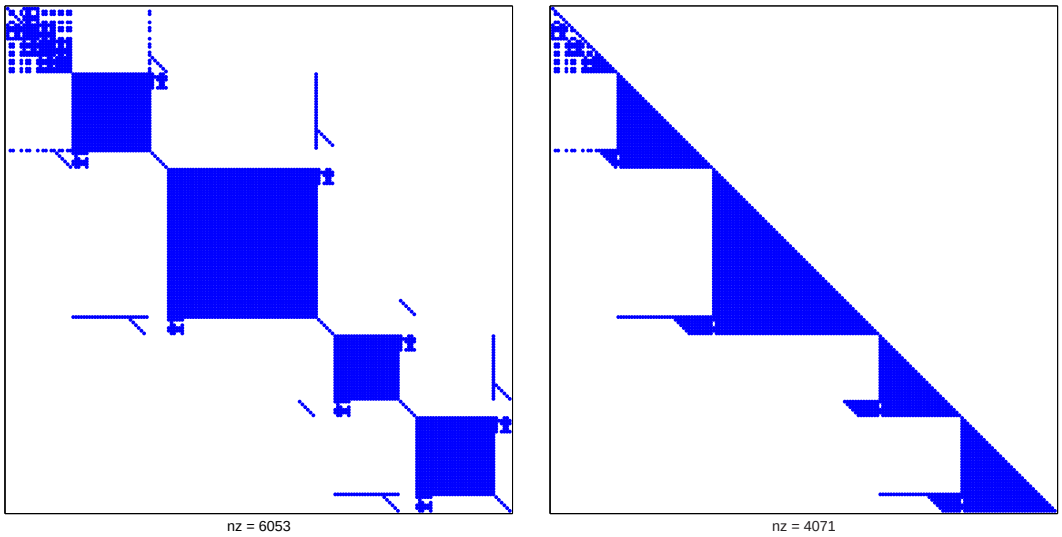


Figure 4.3: Large dense diagonal blocks in a typical multiphase sparse optimal control problem.

(4.4). We emphasize that the density of BB^T (and hence of its Cholesky factors) is independent of the ordering of the rows and columns of B.

We first remove this global dependence by replacing the dense rows and columns corresponding to time variables and parameters with appropriate multiples of unit vectors. To explain this, we return to the example Jacobian B shown in Figure 4.1. We replace the first column of B by $[B_{1,1}, 0, \ldots, 0]$, and denote the new matrix $\tilde{B}$. The structure of the resulting $\tilde{B}\tilde{B}^T$ is shown in Figure 4.4. In the multiphase case, and in cases when the matrix has more multiple dense columns, we replace the dense columns in a similar manner. This can be thought of as a dropping strategy, in which we discard fill from off-diagonal entries for the global variables. That is, we compute an approximation $\tilde{R}$ to the exact Cholesky factor R for BB^T. Entries of R attributable to global variables (included the off-diagonal global entries themselves) are dropped.

Applying this dropping strategy to the Jacobian B from Figure 4.2 gives the resulting sparsity pattern of $\tilde{B}\tilde{B}^T$ and its Cholesky factor $\tilde{R}$ shown in Figure 4.5.

$$\text{struct}(\tilde{B}\tilde{B}^T) = \left[\begin{array}{cccccccc}
\times\times\times\times\times\times\times\times\times & & & & & & & \\
\times\times\times\times\times\times\times\times\times & & & & & & & \\
\times\times\times\times\times\times\times\times\times & & & & & & & \\
\times\times\times\times\times\times\times\times\times & & & & & & & \\
\times\times\times\times\times\times\times\times\times & & & & & & & \\
\times\times\times\times\times\times\times\times\times\times\times\times\times & & & & & & & \\
\times\times\times\times\times\times\times\times\times\times\times\times\times & & & & & & & \\
\times\times\times\times\times\times\times\times\times\times\times\times\times & & & & & & & \\
\times\times\times\times\times\times\times\times\times\times\times\times\times & & & & & & & \\
 & \times\times\times\times\times\times\times\times\times\times\times\times & & & & & & \\
 & \times\times\times\times\times\times\times\times\times\times\times\times & & & & & & \\
 & \times\times\times\times\times\times\times\times\times\times\times\times & & & & & & \\
 & \times\times\times\times\times\times\times\times\times\times\times\times & & & & & & \\
 & & \times\times\times\times\times\times\times\times\times\times\times\times & & & & & \\
 & & \times\times\times\times\times\times\times\times\times\times\times\times & & & & & \\
 & & \times\times\times\times\times\times\times\times\times\times\times\times & & & & & \\
 & & \times\times\times\times\times\times\times\times\times\times\times\times & & & & & \\
 & & & \times\times\times\times\times\times\times\times\times\times\times\times & & & & \\
 & & & \times\times\times\times\times\times\times\times\times\times\times\times & & & & \\
 & & & \times\times\times\times\times\times\times\times\times\times\times\times & & & & \\
 & & & \times\times\times\times\times\times\times\times\times\times\times\times & & & & \\
 & & & & \times\times\times\times\times\times\times\times\times\times\times\times\times & & & \\
 & & & & \times\times\times\times\times\times\times\times\times\times\times\times & & & \\
 & & & & \times\times\times\times\times\times\times\times\times\times\times\times & & & \\
 & & & & \times\times\times\times\times\times\times\times\times\times\times\times & & & \\
 & & & & & \times\times\times\times\times\times\times\times & & \\
 & & & & & \times\times\times\times\times\times\times\times & & \\
 & & & & & \times\times\times\times\times\times\times\times & & \\
 & & & & & \times\times\times\times\times\times\times\times & &
\end{array}\right].$$

Figure 4.4: Structure of $\tilde{B}\tilde{B}^T$, where $\tilde{B}$ approximates the Jacobian B

The large dense diagonal blocks are no longer present.

The linkage boundary conditions appear at the end of each phase but depend on data from the end of the *previous* phase and information at the *beginning* of the current phase. We reorder the constraint equations so that the linkage conditions appear at the beginning of the phase. This ordering leads to much more sparse Cholesky factors and an efficient factorization, since little fill occurs outside the blocks along the diagonal. as illustrated in Figure 4.6.

Of course, by removing the global dependence, we lose considerable information. Because the global dependence within phases is also present in the Hessian portion of the KKT matrix, we construct a second preconditioner which includes Hessian information but applies an analogous dropping strategy.

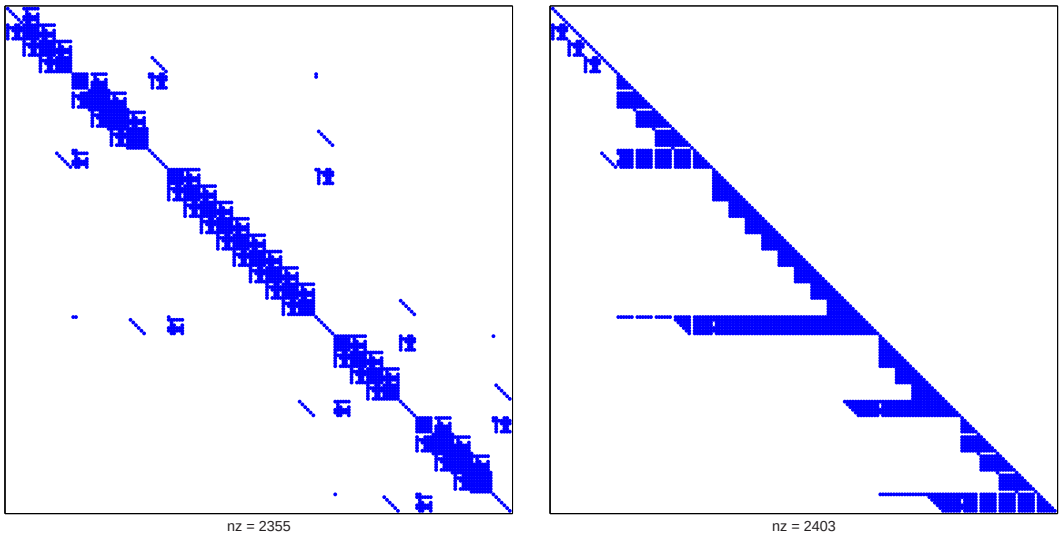


Figure 4.5: Locally dependent approximate Jacobian matrix and Cholesky factor.

The second preconditioner is

$$\mathcal{P}_2 = \begin{bmatrix} \tilde{H} & \tilde{B}^T \\ \tilde{B} & 0 \end{bmatrix}, \tag{4.6}$$

where $\tilde{B}$ is defined as before, and $\tilde{H} \approx H$.

The preconditioner (4.6) is applied via the factorization

$$\begin{bmatrix} \tilde{H} & \tilde{B}^T \\ \tilde{B} & 0 \end{bmatrix} = \begin{bmatrix} \tilde{H} & \\ \tilde{B} & -S \end{bmatrix} \begin{bmatrix} I & \tilde{H}^{-1}\tilde{B}^T \\ & I \end{bmatrix}, \tag{4.7}$$

where S is the Schur complement

$$S = \tilde{B}\tilde{H}^{-1}\tilde{B}^T. \tag{4.8}$$

We refer to the preconditioner (4.6) as a *Local Schur Complement Preconditioner* (**Local-SC**).

Figure 4.7 shows the Hessian for the problem in Figures 4.2—4.6. The left half of Figure 4.7 shows the sparsity of the Hessian H from the original KKT matrix; the right portion shows the factor L from an LU factorization of H.

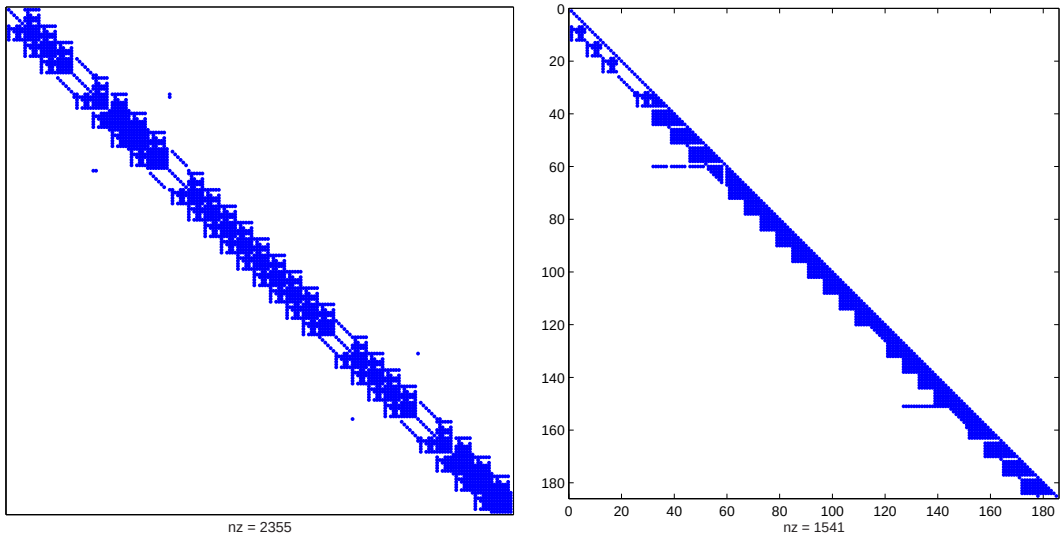


Figure 4.6: Reordered locally dependent approximate Jacobian matrix and Cholesky factor.

Figure 4.8 shows $\tilde{H}$, the approximate Hessian with global dependencies removed, and the corresponding $\tilde{L}$ factor, where fill from the global variables is dropped. Removing the global dependence for each phase dramatically improves the sparsity of the Hessian and provides a sparse factorization. Our experiments below demonstrate that this is an effective preconditioner.

4.4 Description of Test Problems

In this section, we describe in more detail the sparse optimal control problems for the matrices used in the numerical experiments. Most of the problems are industry problems and not only of academic interest. The are part of a Boeing test suite of challenging optimal control problems. Many of the problems are described in [13]; others are described in [16]. Some problems are not formally documented, and our description comes from [14]. The matrices used in this chapter were also used in Chapter 3; for information on dimension and nonzeros, see Table 3.2. All the

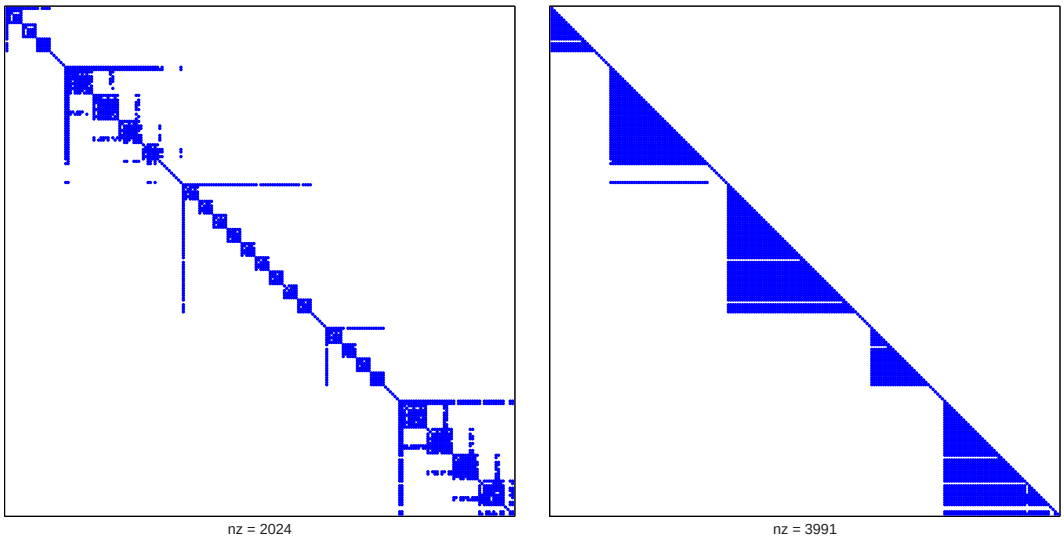


Figure 4.7: Original Hessian matrix and corresponding L factor.

matrices come from multi-phase problems. The last digit of the matrix name refers to the ODE mesh (time discretization) at which the matrices were generated.

4.4.1 Two-Burn Orbit Transfer

The matrices **brn203r1** and **brn201r1**—**brn201r6** are from a two-burn orbit transfer problem [13, page 152]. In this problem, a first propulsion "burn" moves a vehicle from an initial orbit to a transfer orbit. After coasting for an unspecified time, a second burn places the vehicle in a geosynchronous orbit. The problem is to construct the optimal steering during the thrusts such that fuel usage is minimized. There are four phases: two coast arcs, and two burn arcs. The location, duration, and steering of the burns must be chosen to maximize the weight in the final orbit, thereby minimizing the fuel used. Initial and final times for all intermediate phases are free, as are the final time for the initial phase and initial time for the final phase.

The matrices **traj01r1**,**traj03r1**, and **traj05r1** also are based on maximizing the final weight of a two-burn orbit transfer problem. In this case, there are five phases.

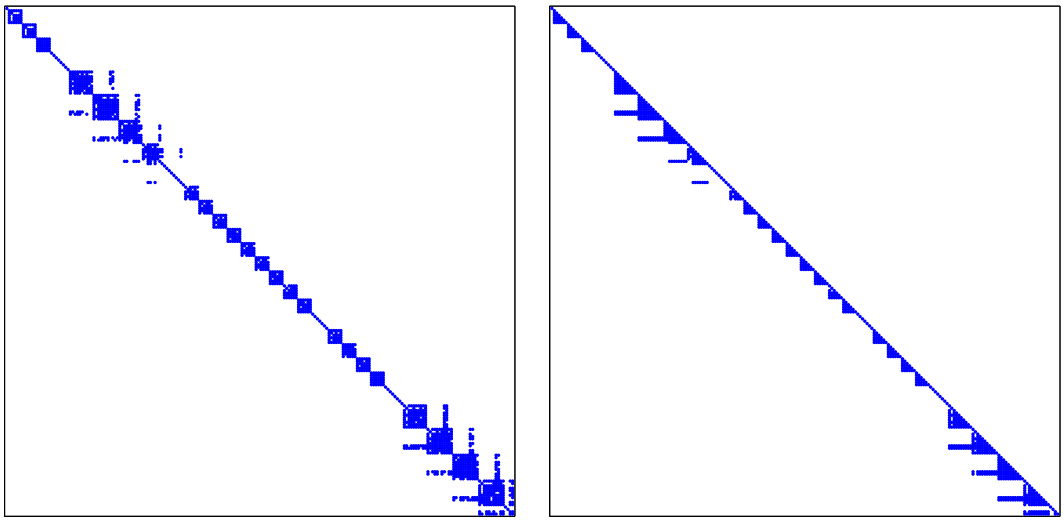


Figure 4.8: Locally dependent approximate Hessian matrix and corresponding $\tilde{L}$ factor.

The problems are described in detail in [16].

4.4.2 Other Optimal Control Problems

The problem **mirv01r1** is based on a *maneuverable independent reentry vehicle* trajectory, which models the trajectory of a ballistic missile from launch point to target. The objective is to maximize a deviation from a direct trajectory in order to avoid anti-missile defenses. The trajectory is broken into five phases.

The problem **capt09r1** determines the fuel-minimizing trajectory of a commercial airliner. The trajectory is broken down into fifteen phases describing the various stages of climbing, cruising, and descending at different altitudes and speeds. Global parameters include initial and final times for each phase and mission final time. Phases are linked by boundary conditions specifying altitude, velocity, state continuity, etc. The problem is described in detail in [16].

The matrix **gsoc01r1** comes from an eight-phase problem based on a high-performance

aircraft mission of an aircraft flying among various waypoints, where at some intermediate point, the aircraft must launch a glide weapon.

The matrix **putt01r1** is from a golf-putter problem, based on the trajectory a golf ball follows on a putting green; see [13, pages 72–75]. The problem is divided into two phases, representing the path of the ball on the green, and then once it falls into the hole.

The problem **lwbr01r1** is based on a kinetic model of a chemical reactor system, described in [55]. The problem is formulated as an optimal control problem with stiff DAE's, three phases, inequality path constraints, equality and inequality boundary conditions, and a variable terminal time.

4.5 Numerical Experiments

In this section, we present experiments with the preconditioners described in Section 4.3, applied to sparse optimal control problems. All tests were performed on a Sun Ultra-4 SPARCstation with 2048 MB physical RAM running Sun OS 5.6. Algorithms were implemented using MATLAB version 5.3. As in Chapter 3, we count GMRES iterations, fill for the preconditioners, and floating point operations. We do not measure CPU time, since our implementations of the algorithms rely on both built-in MATLAB functions and our own MATLAB scripts.

For every problem, the coefficient matrices are of the form

$$\mathcal{H} = \begin{bmatrix} H & B^T \\ B & 0 \end{bmatrix},$$

where $B \in \mathbb{R}^{m \times n}, m \leq n$ has full rank. The matrices were generated in SOCS at iteration 2 of a sequential quadratic programming method associated with a sparse optimal control problem.

The iterative method is full GMRES with left preconditioning. The initial guess is the zero vector. Right-hand sides are constructed with the solution $[1, 2, \ldots, N]^T$,

where $N = n + m$ represents the dimension of the coefficient matrix $\mathcal{H}$. In all cases, iterations were terminated when the 2-norm of the relative residual was reduced by at least six orders of magnitude, or when a maximum of 200 iterations was completed.

We compare three different preconditioners to exact constraint preconditioning **CP**, described in Chapter 3.

Local–CP reorders the phase linkage constraints to the beginning of each phase and approximates the constraint preconditioner with dropping in the Jacobian B. **Local–SC** discards fill in both the Hessian H and Jacobian B. For both preconditioners, the dropping strategy is as described in 4.3.

We also include results with the **MC64–ILU** approach described in Chapter 2, in which scalings and permutations are applied to the KKT matrix, to improve the stability and accuracy of ILU factorizations. The experiments in Chapter 3 showed **MC64–ILU** preconditioning applied to optimal control matrices is effective provided a preconditioner can be constructed. However, the preconditioner breaks down in construction for one matrix.

Table 4.1 contains the data from our experiments. Each row of the table represents one problem. The first column contains the matrix name. Subsequent columns contain iteration counts and sparsity information for each preconditioner. **CP** is exact constraint preconditioning; **Local–CP** is the local constraint preconditioner (4.5); and **Local–SC** is the local Schur Complement preconditioner (4.6).

We include sparsity information for each preconditioner, compared to the original KKT matrix. We report the following nonzero ratios:

- For **CP**, $\rho = \text{nnz}(R)/\text{nnz}(B)$ from the Cholesky decomposition $RR^T = BB^T$. We compute R using the built-in MATLAB `qr()` command. In general, R is much denser than B. $\rho_1 \approx 1.0$ represents an extremely sparse factorization.

- For **Local-CP** $\rho = \text{nnz}(\tilde{R})/\text{nnz}(B)$ from the Cholesky decomposition $\tilde{R}\tilde{R}^T = \tilde{B}\tilde{B}^T$. Again we compute $\tilde{R}$ using the built-in MATLAB `qr()` command.

- For **Local–SC**, $\rho = \text{nnz}(\tilde{L}+\tilde{U})/\text{nnz}(\mathcal{H})$, where $\tilde{L}, \tilde{U}$ represent the factors from (4.7) and $\mathcal{H}$ denotes the original KKT matrix. For a general LU factorization of K, ρ is much larger than 1; $\rho \leq 2$ represents a very sparse factorization.
- For **MC64–ILU**, $\rho = \text{nnz}(\bar{L}+\bar{U})/\text{nnz}(\mathcal{H})$, where let $\bar{L}$ and $\bar{U}$ denote the LU factors for **MC64–ILU**.

Table 4.1: Comparison of preconditioners for sparse optimal control problems.

	CP			**Local—CP**			**Local—SC**			**MC64—ILU**		
matrix	its	ρ	flops	its	ρ	flops	its	ρ	flops	its	ρ	flops
brn201r1	10	2.17	1.2+e06	18	0.95	1.7+e06	9	1.55	1.3+e06	16	0.98	1.5+e06
brn201r2	7	4.14	2.4+e06	16	0.98	2.6+e06	9	1.55	2.3+e06	13	1.00	2.1+e06
brn201r3	9	6.47	1.1+e07	26	1.55	1.1+e08	30	2.02	3.1+e08	20	1.88	1.2+e07
brn201r4	62	13.55	1.5+e08	134	1.47	4.2+e08	12	1.95	5.6+e08	24	1.23	2.7+e07
brn201r5	10	13.84	5.3+e07	30	1.72	6.5+e08	30	2.05	7.3+e08	21	1.42	2.8+e07
brn201r6	11	17.99	7.2+e07	29	1.65	7.4+e08	46	2.03	8.2+e08	24	1.67	4.8+e07
brn203r1	4	2.25	5.9+e05	17	0.97	1.5+e06	14	1.44	1.8+e06	6	0.79	5.1+e05
traj01r1	10	2.62	7.5+e05	19	1.24	1.3+e06	11	2.51	1.7+e06	13	1.26	8.5+e05
traj03r1	15	2.19	1.1+e06	24	1.16	1.8+e06	11	1.76	9.8+e05	9	1.25	5.8+e05
traj05r1	11	1.94	1.0+e06	20	1.04	1.7+e06	11	2.96	3.5+e06	10	0.84	8.2+e05
mirv01r1	24	11.16	3.5+e07	31	1.04	1.6+e07	9	1.65	6.0+e06	17	1.94	1.0+e07
capt09r1	4	4.77	3.1+e06	34	1.06	1.5+e07	29	1.52	1.4+e07	bd	—	—
gsoc01r1	21	3.83	8.5+e06	33	1.18	1.2+e07	17	1.56	6.4+e06	12	3.01	8.5+e06
putt01r1	4	3.53	8.7+e04	8	1.23	1.1+e05	5	1.66	9.3+e04	5	1.41	7.9+e04
lwbr01r1	9	6.58	9.5+e06	20	1.92	9.7+e06	8	2.42	5.7+e06	22	3.95	1.8+e07

As noted in the previous chapter, the sparsity of the exact constraint preconditioner (**CP**) suffers from the dense blocks caused by the global variables. For four problems (brn201r4, brn201r5,brn201r6, and mirv01r1), the number of nonzeros in the Cholesky factor of BB^T is an order of magnitude larger than the number of nonzeros in B. However, iteration counts are consistently low, except for the matrix brn201r4, which also caused problems for **Local–CP**. Thus, while **CP** converges according to the theoretical predictions of Chapter 3, the storage requirements for the preconditioner are high.

Local–CP is the sparsest preconditioner, with density in the Cholesky factor for

$\tilde{B}\tilde{B}^T$ close to that of the original Jacobian B. Iteration counts are higher in general than for exact constraint preconditioning (as expected, since **Local–CP** approximates **CP**). However, iteration counts are generally low, especially for the larger problems. The preconditioner is very sparse. While considerable information may be discarded in constructing the preconditioner, experiments suggest little information is lost, and the extremely sparse preconditioner is effective. The high flop counts are attributable to the increased iteration counts.

The experiments in Table 4.1 suggest that the inclusion of the incomplete Hessian in the preconditioner **Local–SC** provides a good balance between sparsity and fast convergence. For every problem, the preconditioner is roughly one or two times as sparse as the original matrix. For six of the 15 problems, the iteration counts are also the lowest among the preconditioners tested. In all cases, iteration counts are low. The worst cases for the **Local–SC** preconditioner are brn201r5 and brn201r6, for which GMRES required 30 and 46 iterations, respectively, to converge. In fact, **Local–SC** is worse than **Local–CP** for these two matrices. However, in general, **Local–SC** is the most effective preconditioner in terms of sparsity and convergence rate.

The application of **Local–SC** is more expensive than **Local–CP**. In particular, the explicit formulation and solve with the Schur complement factor (4.8) can be expensive. However, for the matrices considered here, $\tilde{H}$ is nearly block-tridiagonal, and $\tilde{B}$ is tightly banded, so the Schur complement is nearly block-tridiagonal. For most problems, including the Hessian information is worth in increased flop counts.

4.6 Conclusion

In this chapter, we have narrowed our focus from developing preconditioners for general KKT systems to constructing preconditioners for special KKT systems. We develop two preconditioners for matrices arising in sparse optimal control problems,

when the transcribed problem is solved with sequential quadratic programming. Our preconditioners use information about the problem and the numerical formulation in order to construct sparse approximations of the original KKT matrix.

The first preconditioner **Local–CP** is based on exact constraint preconditioning described in Chapter 3. The variables are reordered, and we apply a dropping strategy to the Jacobian portion of the KKT matrix, which eliminates dense blocks in the factorization of the preconditioner. Experiments demonstrate that the preconditioner is extremely sparse. Local–CP preconditioned GMRES requires more iterations than with exact constraint preconditioning, but iteration counts are still low.

The second preconditioner applies the same dropping strategy to the Hessian portion of the KKT matrix, in addition to the reordering and dropping applied to the Jacobian. The application uses a factorization of an explicit Schur complement, which increases the cost of the preconditioner. However, the preconditioner is very sparse, and the preconditioned iteration converges fast, outperforming exact constraint preconditioning for most of our problems. For many matrices, including the approximation to the Hessian block is worth the added cost.

The problems we experiment on are industrial problems generated with Boeing's sparse optimal control software SOCS. The linear systems that arise are currently solved with an efficient direct method, but the high amounts of fill in the direct factorizations motivate this investigation into finding effective preconditioners for iterative methods. As shown by our experiments, iterative methods are a realistic option when combined with preconditioners such as those presented in this research.

Chapter 5

Conclusion

This dissertation presents several new preconditioners for linear systems. First, we present an innovative approach to preconditioning general nonsingular matrices, for which no structure is assumed. The approach involves nonsymmetric permutations and scalings which place large entries on the diagonal. We choose matrices for which iterative methods have so far been unsuccessful, due to failures in constructing preconditioners. Experiments show that preprocessing the matrices with these permutations and scalings allows stable construction of preconditioners. In particular, permutations and scalings which maximize the product of the entries of the diagonal, combined with fill-reducing symmetric reorderings, result in reliable preconditioners based on drop tolerances (like ILUT and AINV).

We also present two new preconditioners for KKT systems. The preconditioners exploit the KKT the structure of the coefficient matrix and approximate exact constraint preconditioners by applying A-conjugate algorithms based on AINV and RIF to the constraint portion of the KKT matrix. Experiments show that the approximate constraint preconditioners are much sparser than the exact constraint preconditioner, and for many problems are just as effective as the exact constraint preconditioner in reducing GMRES iterations. However, for sparse optimal control problems, the

approximate constraint preconditioners are ineffective. We also apply the maximum-diagonal permutations and scalings to the KKT matrices, and use a standard ILU preconditioner. The ILU preconditioner constructed from the scaled and permuted matrix is sparse and effective, but construction breaks down for several matrices. We apply no fine-tuning for the ILU preconditioner; with some adjustments, an effective preconditioner could likely be constructed.

Finally, we describe preconditioners for sparse optimal control problems; the preconditioners above are less effective for these matrices, and in fact few effective preconditioners exist for sparse optimal control problems. The matrices in our experiments are generated by a sequential quadratic programming method applied to the transcribed optimal control problem, and our preconditioners used information about the problem formulation and discretization. This information is used to control sparsity via reorderings and dropping in two new approximate constraint preconditioners. One preconditioner applies the dropping strategy and a reordering only to the constraint portion of the KKT matrix, resulting in an extremely sparse preconditioner. GMRES iteration counts are low for this preconditioner. The second preconditioner for optimal control problems includes the Hessian information, using the same reorderings and dropping for the constraints, and also applies the dropping strategy to the Hessian block of the KKT matrix. This preconditioner is most effective for the sparse optimal control problems, offering a good balance between GMRES iteration counts and sparsity.

Bibliography

[1] M. A. Ajiz and A. Jennings. A robust incomplete Choleski-conjugate gradient algorithm. *Internat. J. Numer. Methods Engrg.*, 20(5):949–966, 1984.

[2] S. T. Barnard, L. M. Bernardo, and S. H. D. An mpi implementation of the spai preconditioner on the t3e. *Int. J. High. Perf. Comput. Applic.*, 13:107–123, 1999.

[3] M. Benzi. personal communication, January 2002.

[4] M. Benzi, J. K. Cullum, and M. Tůma. Robust approximate inverse preconditioning for the conjugate gradient method. *SIAM J. Sci. Comput.*, 22(4):1318–1332 (electronic), 2000.

[5] M. Benzi, J. C. Haws, and M. Tůma. Preconditioning highly indefinite and nonsymmetric matrices. Technical Report LA-UR-99-4857, Los Alamos National Laboratory, Los Alamos, New Mexico, August 1999.

[6] M. Benzi, J. C. Haws, and M. Tůma. Preconditioning highly indefinite and nonsymmetric matrices. *SIAM J. Sci. Comput.*, 22(4):1333–1353 (electronic), 2000.

[7] M. Benzi, C. D. Meyer, and M. Tůma. A sparse approximate inverse preconditioner for the conjugate gradient method. *SIAM J. Sci. Comput.*, 17(5):1135–1149, 1996.

[8] M. Benzi, D. B. Szyld, and A. van Duin. Orderings for incomplete factorization preconditioning of nonsymmetric problems. *SIAM J. Sci. Comput.*, 20(5):1652–1670 (electronic), 1999.

[9] M. Benzi and M. Tůma. A sparse approximate inverse preconditioner for nonsymmetric linear systems. *SIAM J. Sci. Comput.*, 19(3):968–994 (electronic), 1998.

[10] M. Benzi and M. Tůma. A comparative study of sparse approximate inverse preconditioners. *Appl. Numer. Math.*, 30(2-3):305–340, 1999. Iterative methods and preconditioners (Berlin, 1997).

[11] M. Benzi and M. Tůma. A robust factorization preconditioner for positive definite matrices. submitted to *Numerical Linear Algebra with Applications*, 2001.

[12] M. Benzi and M. Tuma. Orderings for factorized sparse approximate inverse preconditioners. *SIAM J. Sci. Comput.*, 21(5):1851–1868, Sept. 2000.

[13] J. T. Betts. *Practical methods for optimal control using nonlinear programming.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2001.

[14] J. T. Betts. personal communication, February 2002.

[15] J. T. Betts, N. Biehn, S. L. Campbell, and W. P. Huffman. Compensating for order variation in mesh refinement for direct transcription methods. *J. Comput. Appl. Math.*, 125(1-2):147–158, 2000. Numerical analysis 2000, Vol. VI, Ordinary differential equations and integral equations.

[16] J. T. Betts, S. K. Eldersveld, and W. P. Huffman. Sparse nonlinear programming test problems (release 1.0). Technical Report BCSTECH-93-047, Mathematics and Computing Technology (formerly Boeing Computer Services), P.O. Box 3707 MC 7L-21, Seattle, WA 98124-2207, 1993.

[17] J. T. Betts and W. P. Huffman. Mesh refinement in direct transcription methods for optimal control. *Optimal Control Appl. Methods*, 19(1):1–21, 1998.

[18] Å. Björck. *Numerical Methods for Least Squares Problems*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1996.

[19] C. W. Bomhof and H. A. van der Vorst. A parallel linear system solver for circuit simulation problems. *Numer. Linear Algebra Appl.*, 7(7-8):649–665, 2000. Preconditioning techniques for large sparse matrix problems in industrial applications (Minneapolis, MN, 1999).

[20] I. Bongartz, A. Conn, N. Gould, and P. Toint. Cute: Constrained and unconstrained testing environment. *ACM Trans. Math. Software*, 21:123–160, 1995.

[21] R. Bridson and W.-P. Tang. Ordering, anisotropy, and factored sparse approximate inverses. *SIAM J. Sci. Comput.*, 21(3):867–882, 1999.

[22] E. Chow and Y. Saad. Experimental study of ILU preconditioners for indefinite matrices. *J. Comput. Appl. Math.*, 86(2):387–414, 1997.

[23] H. N. Cofer and M. A. Stadtherr. Reliability of iterative linear equations solvers in chemical process simulation. *Computers Chem. Engng.*, 20:1123–1132, 1996.

[24] T. Davis. University of Florida sparse matrix collection. available online at http://www.cise.ufl.edu/research/sparse/matrices.

[25] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1:269–271, 1959.

[26] I. S. Duff. Algorithm 575: Permutations for a zero-free diagonal. *ACM Trans. Math. Software*, 7:387–390, 1981.

[27] I. S. Duff. On algorithms for obtaining a maximum transversal. *ACM Trans. Math. Software*, 7:315–330, 1981.

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

[29] I. S. Duff, R. G. Grimes, and J. G. Lewis. The Rutherford–Boeing sparse matrix collection. Technical Report RAL-TR-97-031, Rutherford Appleton Laboratory, 1997.

[30] I. S. Duff and J. Koster. The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. *SIAM J. Matrix Anal. Appl.*, 20(4):889–901 (electronic), 1999. Sparse and structured matrices and their applications (Coeur d'Alene, ID, 1996).

[31] I. S. Duff and J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM J. Matrix Anal. Appl.*, 22(4):973–996 (electronic), 2001.

[32] I. S. Duff and G. A. Meurant. The effect of ordering on preconditioned conjugate gradients. *BIT*, 29(4):635–657, 1989.

[33] J. Edmonds. Maximum matching and a polyhedron with $0, 1$-vertices. *J. Res. Nat. Bur. Standards Sect. B*, 69B:125–130, 1965.

[34] H. C. Elman. A stability analysis of incomplete LU factorizations. *Math. Comp.*, 47(175):191–217, 1986.

[35] H. C. Elman, D. J. Silvester, and A. J. Wathen. Iterative methods for problems in computational fluid dynamics. In *Iterative methods in scientific computing (Hong Kong, 1995)*, pages 271–327. Springer, Singapore, 1997.

[36] R. W. Freund. A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems. *SIAM J. Sci. Comput.*, 14(2):470–482, 1993.

[37] R. W. Freund and N. M. Nachtigal. QMR: a quasi-minimal residual method for non-Hermitian linear systems. *Numer. Math.*, 60(3):315–339, 1991.

[38] A. George, J. W. H. Liu, and E. Ng. Computer solution of large sparse positive definite systems. Revision of George and Liu (1981), to be published., March 2001.

[39] P. E. Gill, W. Murray, and M. H. Wright. *Numerical linear algebra and optimization. Vol. 1.* Addison-Wesley Publishing Company Advanced Book Program, Redwood City, CA, 1991.

[40] M. Gilli and G. Pauletto. Krylov methods for solving models with forward-looking variables. *J. Econom. Dynam. Control*, 22(8-9):1275–1289, 1998. Algorithms and economic dynamics (Geneva, 1996).

[41] N. I. M. Gould, M. E. Hribar, and J. Nocedal. On the solution of equality constrained quadratic programming problems arising in optimization. *SIAM J. Sci. Comput.*, 23(4):1375–1394, 2001.

[42] N. I. M. Gould and J. A. Scott. Sparse approximate-inverse preconditioners using norm-minimization techniques. *SIAM J. Sci. Comput.*, 19(2):605–625 (electronic), 1998.

[43] A. Greenbaum. *Iterative methods for solving linear systems*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997.

[44] M. J. Grote and T. Huckle. Parallel preconditioning with sparse approximate inverses. *SIAM J. Sci. Comput.*, 18(3):838–853, 1997.

[45] I. Gustafsson. A class of first order factorization methods. *BIT*, 18(2):142–156, 1978.

[46] J. C. Haws, A. J. Booker, J. G. Lewis, and J. Wu. Evaluating tunable preconditioners with design explorer. Technical Report M&CT-TECH-01-020, Mathematics and Computing Technology, The Boeing Company, P.O. Box 3707 MC 7L-21, Seattle, WA 98124-2207, December 2001.

[47] J. C. Haws and C. D. Meyer. Preconditioning KKT systems. Technical Report M&CT-TECH-01-021, Mathematics and Computing Technology, The Boeing Company, P.O. Box 3707 MC 7L-21, Seattle, WA 98124-2207, December 2001.

[48] L. Hemmingsson-Frändén and A. Wathen. A nearly optimal preconditioner for the Navier-Stokes equations. *Numer. Linear Algebra Appl.*, 8(4):229–243, 2001.

[49] I. C. F. Ipsen and C. D. Meyer. The idea behind Krylov methods. *Amer. Math. Monthly*, 105(10):889–899, 1998.

[50] D. James. *Conjugate Gradient Methods for Constrained Least Squares Problems.* PhD thesis, North Carolina State University, Department of Mathematics, 1990.

[51] A. Jennings and M. A. Ajiz. Incomplete methods for solving $A^T Ax = b$. *SIAM J. Sci. Statist. Comput.*, 5(4):978–987, 1984.

[52] C. Keller, N. I. M. Gould, and A. J. Wathen. Constraint preconditioning for indefinite linear systems. *SIAM J. Matrix Anal. Appl.*, 21(4):1300–1317 (electronic), 2000.

[53] S. A. Kharchenko, L. Y. Kolotilina, A. A. Nikishin, and A. Y. Yeremin. A robust AINV-type method for constructing sparse approximate inverse preconditioners in factored form. *Numer. Linear Algebra Appl.*, 8(3):165–179, 2001.

[54] L. Y. Kolotilina and A. Y. Yeremin. Factorized sparse approximate inverse preconditionings. I. Theory. *SIAM J. Matrix Anal. Appl.*, 14(1):45–58, 1993.

[55] D. Leineweber. *Efficient reduced methods for the optimization of chemical processes described by large sparse DAE models.* PhD thesis, Interdisziplinäres Zentrum fr Wissenschaftliches Rechnen, Universität Heidelberg, Heidelberg, Germany, 1998.

[56] X. S. Li and J. W. Demmel. Making sparse Gaussian elimination scalable by static pivoting. In *Proceedings of SuperComputing 98 Conference, Association for Computing Machinery, 1998 [CD-ROM]*, 1998.

[57] R. J. Lipton, D. J. Rose, and R. E. Tarjan. Generalized nested dissection. *SIAM J. Numer. Anal.*, 16(2):346–358, 1979.

[58] J. W. H. Liu. Modification of the minimum-degree algorithm by multiple elimination. *ACM Trans. Math. Software*, 11(2):141–153, 1985.

[59] L. Lukšan and J. Vlček. Indefinitely preconditioned inexact Newton method for large sparse equality constrained non-linear programming problems. *Numer. Linear Algebra Appl.*, 5(3):219–247, 1998.

[60] T. A. Manteuffel. An incomplete factorization technique for positive definite linear systems. *Math. Comp.*, 34(150):473–497, 1980.

[61] W. D. McQuain, C. J. Ribbens, L. T. Watson, and R. C. Melville. Preconditioned iterative methods for sparse linear algebra problems arising in circuit simulation. *Comput. Math. Appl.*, 27(8):25–45, 1994.

[62] J. A. Meijerink and H. A. van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Math. Comp.*, 31(137):148–162, 1977.

[63] C. D. Meyer. *Matrix analysis and applied linear algebra*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000. With 1 CD-ROM (Windows, Macintosh and UNIX) and a solutions manual (iv+171 pp.).

[64] National Institute of Standards. Matrix market. Available online at http://math.nist.gov/MatrixMarket.

[65] J. Nocedal and S. J. Wright. *Numerical optimization*. Springer-Verlag, New York, 1999.

[66] M. Olschowka and A. Neumaier. A new pivoting strategy for Gaussian elimination. *Linear Algebra Appl.*, 240:131–151, 1996.

[67] G. Pauletto. *Solution and Simulation of Macroeconometric Models*. PhD thesis, University of Geneva, Switzerland, Department of Econometrics, 1995.

[68] I. Perugia and V. Simoncini. An optimal indefinite preconditioner for a mixed finite element method. Technical Report 1098, IAN-CNR, November 1998.

[69] I. Perugia and V. Simoncini. Block-diagonal and indefinite symmetric preconditioners for mixed finite element formulations. *Numer. Linear Algebra Appl.*, 7(7-8):585–616, 2000. Preconditioning techniques for large sparse matrix problems in industrial applications (Minneapolis, MN, 1999).

[70] M. Rozložník and V. Simoncini. Short–term recurrences for indefinite preconditioning of saddle point problems. Technical Report 1181, IAN-CNR, February 2001.

[71] Y. Saad. Preconditioning techniques for nonsymmetric and indefinite linear systems. *J. Comput. Appl. Math.*, 24(1-2):89–105, 1988. Iterative methods for the solution of linear systems.

[72] Y. Saad. ILUT: a dual threshold incomplete *LU* factorization. *Numer. Linear Algebra Appl.*, 1(4):387–402, 1994.

[73] Y. Saad. *Iterative methods for sparse linear systems*. PWS, Boston, MA, 1996.

[74] Y. Saad and M. H. Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 7(3):856–869, 1986.

[75] H. A. van der Vorst. Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 13(2):631–644, 1992.

[76] X. Wang, K. A. Gallivan, and R. Bramley. CIMGS: an incomplete orthogonal factorization preconditioner. *SIAM J. Sci. Comput.*, 18(2):516–536, 1997.

[77] J. Zhang. A multilevel dual reordering strategy for robust incomplete LU factorization of indefinite matrices. *SIAM J. Matrix Anal. Appl.*, 22(3):925–947 (electronic), 2000.

Appendix A

Appendix

We recall theorems from [52] which describe properties of the constraint preconditioned system. In [52], the authors consider a constraint preconditioner with a nonsingular (1,1) block. When that block is assumed to be the identity matrix, many of the results simplify.

We will prove certain properties about the eigenvalues of the preconditioned system by looking at similar matrices. Recall that similar matrices have the same spectrum [63, p. 508].

The following theorem describes the clustering of eigenvalues due to constraint preconditioning.

Theorem 5 (Theorem 2.1 in [52]) *Let $\mathcal{H} \in \mathbb{R}^{(n+m)\times(n+m)}$ be a symmetric indefinite, nonsingular matrix of the form*

$$\mathcal{H} = \begin{bmatrix} H & B^T \\ B & 0 \end{bmatrix},$$

with $H \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{m\times n}$, and $m \leq n$. Let $\mathcal{P} \in \mathbb{R}^{(n+m)\times(n+m)}$ be a symmetric indefinite, nonsingular preconditioner of the form

$$\mathcal{P} = \begin{bmatrix} I & B^T \\ B & 0 \end{bmatrix}.$$

Let $Q_2 \in \mathbb{R}^{n\times(n-m)}$ be a matrix whose columns span the nullspace of B. Then the preconditioned matrix $\mathcal{P}^{-1}\mathcal{H}$ has $2m$ eigenvalues with value 1, and $(n-m)$ eigenvalues given by the eigenvalues of $Q_2^T H Q_2$.

Proof. Denote the QR factorization of B^T by $B^T = \mathcal{Q}R$, where $\mathcal{Q} \in \mathbb{R}^{n\times n}, R \in \mathbb{R}^{n\times m}$ and

$$\mathcal{Q} = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}.$$

The columns of $Q_1 \in \mathbb{R}^{n\times m}$ form an orthonormal basis for the range of B^T, and the rows of $Q_2 \in \mathbb{R}^{n\times(n-m)}$ form an orthonormal basis for the nullspace of B; therefore,

$$\begin{aligned} BQ_2 &= 0_{m\times(n-m)}, \text{ and} \\ Q_2^T Q_2 &= I_{(n-m)\times(n-m)} \end{aligned}$$

and since B^T is the orthogonal complement to Q_2,

$$I - Q_2(Q_2^T Q_2)^{-1} Q_2^T = B^T(BB^T)^{-1}B. \tag{A.1}$$

Define the $n \times m$ (right-hand) pseudo-inverse of B by $B^\dagger = B^T(BB^T)^{-1}$ [63], and note that

$$\begin{aligned} BB^\dagger &= I_{m\times m}, \text{ and} \\ (B^\dagger)^T Q_2 &= 0. \end{aligned}$$

Let $M_1 \in \mathbb{R}^{n\times n}$ be given by

$$M_1 = \begin{bmatrix} B^\dagger & Q_2 \end{bmatrix},$$

and note

$$M_1^{-1} = \begin{bmatrix} B \\ Q_2^T \end{bmatrix}.$$

It is clear that $M_1^{-1}M_1 = I$. The reverse product is not so clear:

$$M_1 M_1^{-1} = \begin{bmatrix} B^\dagger & Q_2 \end{bmatrix} \begin{bmatrix} B \\ Q_2^T \end{bmatrix}$$

$$
\begin{aligned}
&= B^T(BB^T)^{-1}B + Q_2Q_2^T \\
&= I - Q_2(Q_2^TQ_2)^{-1}Q_2^T + Q_2Q_2^T \quad \text{(from A.1)} \\
&= I - Q_2Q_2^T + Q_2Q_2^T \\
&= I.
\end{aligned}
$$

Define the $(m+n)\times(m+n)$ matrix

$$
\mathcal{M} = \begin{bmatrix} M_1 & \\ & I_{m\times m} \end{bmatrix}.
$$

Then

$$
M^T\mathcal{H}M = \begin{bmatrix} (B^\dagger)^T & \\ Q_2^T & \\ & I \end{bmatrix} \begin{bmatrix} H & B^T \\ B & \end{bmatrix} \begin{bmatrix} B^\dagger & Q_2 & \\ & & I \end{bmatrix} \tag{A.2}
$$

$$
= \begin{bmatrix} (B^\dagger)^THB^\dagger & (B^\dagger)^THQ_2 & I \\ Q_2^THB^\dagger & Q_2^THQ_2 & 0 \\ I & 0 & 0 \end{bmatrix}. \tag{A.3}
$$

Likewise,

$$
M^T\mathcal{P}M = \begin{bmatrix} (BB^T)^{-1} & 0 & I \\ 0 & I & 0 \\ I & 0 & 0 \end{bmatrix}. \tag{A.4}
$$

The inverse of (A.4) is easy to compute; it is

$$
(M^T\mathcal{P}M)^{-1} = \begin{bmatrix} 0 & 0 & I \\ 0 & I & 0 \\ I & 0 & -(BB^T)^{-1} \end{bmatrix}. \tag{A.5}
$$

and note that $(M^T\mathcal{P}M)^{-1} = M^{-1}\mathcal{P}^{-1}M^{-T}$. Multiplying (A.5) and (A.3),

$$
(M^T\mathcal{P}M)^{-1}M^T\mathcal{H}M = \begin{bmatrix} 0 & 0 & I \\ 0 & I & 0 \\ I & 0 & -(BB^T)^{-1} \end{bmatrix} \begin{bmatrix} (B^\dagger)^THB^\dagger & (B^\dagger)^THQ_2 & I \\ Q_2^THB^\dagger & Q_2^THQ_2 & 0 \\ I & 0 & 0 \end{bmatrix}
$$

$$
\begin{aligned}
&= \begin{bmatrix} I & 0 & 0 \\ Q_2^T H B^\dagger & Q_2^T H Q_2 & 0 \\ (B^\dagger)^T H B^\dagger - (BB^T)^{-1} & (B^\dagger)^T H Q_2 & I \end{bmatrix} \\
&= M^{-1}\mathcal{P}^{-1}\mathcal{H}M.
\end{aligned}
$$

Thus the matrix $M^{-1}\mathcal{P}^{-1}\mathcal{H}M$ has $2m$ eigenvalues with value 1, and $n-m$ eigenvalues given by $Q_2^T H Q_2$, where the columns of Q_2 form a basis for the nullspace of B. The similar matrix $\mathcal{P}^{-1}\mathcal{H}$ has the same eigenvalues. □

Hence, the eigenvalues of interest for the constraint-preconditioned system are given by $Q_2^T H Q_2$, where the columns of Q_2 form a basis for the nullspace of B. It is not immediately clear that this leads to larger, or tighter, clusters of eigenvalues. To prove that constraint preconditioning does lead to a tighter clustering of eigenvalues, we need Cauchy's Interlace Theorem.

Theorem 6 *Let $\mathcal{T} \in \mathbb{R}^{N\times N}$ by a symmetric matrix with leading principal submatrix $T \in \mathbb{R}^{n\times n}$. Denote the eigenvalues and eigenvectors by*

$$
\begin{aligned}
\mathcal{T} z_i = \alpha_i z_i, \quad i = 1, \ldots, N, \quad \alpha_1 \le \alpha_2 \le \ldots \le \alpha_N \\
T y_i = \lambda y_i, \quad i = 1, \ldots, n, \quad \lambda_1 \le \lambda_2 \le \ldots \le \lambda_n.
\end{aligned}
$$

Then

$$
\alpha_k \le \lambda_k \le \alpha_{k+(N-n)}, \qquad k = 1, \ldots, n.
$$

Proof. See [63, p. 552]. □

Thus, the eigenvalues of T are bound by those of $\mathcal{T}$. The following proposition shows that, through a similar relationship, constraint preconditioning can lead to more tightly clustered eigenvalues.

Proposition 7 *Let $\mathcal{H} \in \mathbb{R}^{(n+m)\times(n+m)}$ be a symmetric indefinite, nonsingular matrix of the form*

$$
\mathcal{H} = \begin{bmatrix} H & B^T \\ B & 0 \end{bmatrix},
$$

with $H \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{m\times n}$, and $m \leq n$. Let $Q_2 \in \mathbb{R}^{n\times(n-m)}$ be a matrix whose columns span the nullspace of B. Then the eigenvalues of H bound the eigenvalues of $Q_2^T H Q_2$.

Proof. Let $\lambda \in \sigma(H)$, and let u be the associated eigenvector, i.e. $Hu = \lambda u$. Denote the QR factorization of B^T by $B^T = \mathcal{Q}R$, where

$$\mathcal{Q} = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}.$$

The the rows of $Q_2 \in \mathbb{R}^{n\times(n-m)}$ form an orthonormal basis for the nullspace of B. Define the permutation matrix $P = \begin{bmatrix} 0 & I \\ I & 0 \end{bmatrix}$. Since $\mathcal{Q}$ is orthonormal

$$P^T\mathcal{Q}^T H \mathcal{Q} P u = \lambda P^T \mathcal{Q}^T \mathcal{Q} P u = \lambda u. \tag{A.6}$$

In block form, the matrix on the left of A.6 looks like

$$P^T\mathcal{Q}^T H \mathcal{Q} P = \begin{bmatrix} Q_2^T H Q_2 & Q_2^T H Q_1 \\ Q_1^T H Q_2 & Q_1^T H Q_1 \end{bmatrix}.$$

By Theorem 6, the eigenvalues of $P^T\mathcal{Q}^T H \mathcal{Q} P$ bound those of $Q_2^T H Q_2$. Since similar matrices have the same eigenvalues, the eigenvalues of H bound those of $Q_2^T H Q_2$. □

Finally, constraint preconditioning guarantees that, in exact arithmetic, GMRES will converge in at most $n + m - 2$ iterations for KKT matrices of the form (3.1).

Theorem 8 (Theorem 3.5 from [52]) *Let $\mathcal{H} \in \mathbb{R}^{(n+m)\times(n+m)}$ be a symmetric indefinite nonsingular matrix of the form*

$$\mathcal{H} = \begin{bmatrix} H & B^T \\ B & 0 \end{bmatrix},$$

where $H \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{m\times n}$, with $m \leq n$. Let $\mathcal{P}$ be a preconditioner given by

$$\mathcal{P} = \begin{bmatrix} I & B^T \\ B & 0 \end{bmatrix}.$$

Then the dimension of the Krylov subspace associated with the preconditioned system $\mathcal{P}^{-1}\mathcal{H}x = \mathcal{P}^{-1}b$ is at most $n - m + 2$.

Proof. See [52]. □

Thus, exact constraint preconditioning is an effective approach to preconditioning linear systems, in that GMRES is guaranteed to converge in $n - m + 2$ iterations.