INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
RICE UNIVERSITY

A New Class of Preconditioners for Large-Scale Linear Systems from Interior Point Methods for Linear Programming

by

Aurelio Ribeiro Leite de Oliveira

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy

Approved. Thesis Committee:

Danny C. Sorensen. Chairman
Professor of Computational and Applied Mathematics

Robert E. Bixby
Professor of Computational and Applied Mathematics

Keith D. Cooper
Associate Professor of Computer Science

Richard A. Tapia
Noah Harding Professor of Computational and Applied Mathematics

Houston, Texas
April. 1997
Abstract

A New Class of Preconditioners for Large-Scale Linear Systems from Interior Point Methods for Linear Programming

by

Aurelio Ribeiro Leite de Oliveira

A new class of preconditioners for the iterative solution of the linear systems arising from interior point methods is proposed. For many of these methods, the linear systems come from applying Newton's method on the perturbed Karush-Kuhn-Tucker optimality conditions for the linear programming problem. This leads to a symmetric indefinite linear system called the augmented system. This system can be reduced to the Schur complement system which is positive definite. After the reduction, the solution for the linear system is usually computed via the Cholesky factorization. This factorization can be dense for some classes of problems. Therefore, the solution of these systems by iterative methods must be considered. Since these systems are very ill-conditioned near a solution of the linear programming problem, it is crucial to develop efficient preconditioners. We show that from the point of view of designing preconditioners, it is better to work with the augmented system. We show that all preconditioners for the Schur complement system have an equivalent for the augmented system while the opposite is not true. The theoretical properties of the new preconditioners are discussed. This class works better near a solution of the linear
programming problem when the linear systems are highly ill-conditioned. Another important feature is the option to reduce the preconditioned indefinite system to a positive definite one of the size of the Schur complement. This class of preconditioners relies on the computation of an $LU$ factorization of a subset of columns of the matrix of constraints. The techniques developed for a competitive implementation are rather sophisticated since the subset of columns is not known a priori. The new preconditioner applied to the conjugate gradient method compares favorably with the Cholesky factorization approach. The new approach performs better on large scale problems whose Cholesky factorization contains a large number of nonzero entries. Numerical experiments on several representative classes of linear programming problems are presented to indicate the promise of this new approach.
Acknowledgments

This thesis would not be possible without the support of my graduate advisor Professor Danny Sorensen. I would to thank for his help and patience throughout all my stay at Rice University. I would also like to thank all the other members of my committee, Professors Richard Tapia, Robert Bixby and Keith Cooper for their attention and suggestions during the course of this work.

I would like to thank all my friends at the department and beyond for their support and kindness. Including but not limited to Ana, Beatriz, Cristina, Hector, Hector, Leticia, Lucio, Mayita, Miguel, Nila, Raquel, Sandra, Sergio and Shireen. Special thanks to Zeferino, Eva, Edmar and Marielba for many helpful discussions and comments about this work. Many thanks to Mike Pearlman, Linda Neyra and Fran Mailian who always provided valuable help and by keeping the department running so smoothly. Thanks to my family and friends in Brazil for the help and encouragement I received during my stay in Houston. I would like to thank Walkiria specially for having such sensibility and for being at the right place and time.

I am grateful to the Brazilian Council for the Development of Science and Technology CNPq for financial support during part of my graduate career at Rice University.
Contents

Abstract

Acknowledgments

List of Tables

1 Introduction

1.1 Notation .................................................. 3

2 Linear Programming ........................................ 5

2.1 Linear Programming Problems .............................. 5

2.2 Primal-Dual Interior Point Methods .................. 6

2.3 The Predictor-Corrector Variant ....................... 8

2.4 Computing the Search Directions ...................... 10

2.5 Approaches for Solving the Linear Systems .......... 11

2.5.1 Matrix Factorizations ................................ 11

2.5.2 Solving the Schur Complement ...................... 11

2.5.3 Solving the Augmented System ...................... 12

2.5.4 Search Directions for the Predictor-Corrector Variant .... 14

2.6 Modeling Linear Programming Problems .............. 15

2.6.1 Inequality Constraints ............................... 16

2.6.2 Bounded Variables .................................... 18

2.6.3 Free Variables ......................................... 19

2.6.4 The General Case ..................................... 21

2.7 Stable Reduction ........................................ 24
3 Implementation Issues for the Primal-Dual Method 26

3.1 Preprocessing .............................................. 26
   3.1.1 Other Preprocessing Techniques ....................... 29
3.2 Special Structures ........................................ 30
3.3 Scaling ...................................................... 30
3.4 Starting Point .............................................. 32
   3.4.1 Starting Point and Inequality Constraints ............ 33
   3.4.2 Starting Point and Bounded Variables ................ 34
   3.4.3 The Augmented System Representation for the Starting Point 35
3.5 The Parameters $\tau$ and $\sigma$ ............................ 35
3.6 Stopping Criteria .......................................... 36
3.7 The Schur Complement ..................................... 37

4 The Augmented System 39

4.1 Introduction ............................................... 39
4.2 Properties of the Augmented System ....................... 40
4.3 Preconditioning ............................................ 41
   4.3.1 Motivation ........................................... 41
   4.3.2 An Introduction on Preconditioning ................. 42
   4.3.3 Some Classes of Preconditioners .................... 44
4.4 Preconditioners for the Augmented System ............... 47
   4.4.1 Existing Preconditioners ............................ 47
   4.4.2 Modified Preconditioners ............................ 48
   4.4.3 Unsymmetric Preconditioners ....................... 51
4.5 Why Work with the Augmented System? ................... 52
   4.5.1 The Importance of the Augmented System ............ 53
5 A New Class of Preconditioners

5.1 Building a Preconditioner ........................................... 57
  5.1.1 The Matrix $B$ .................................................. 59
  5.1.2 First Considerations About the Preconditioned Matrix .... 60
5.2 Theoretical Properties of the Preconditioned System .......... 61
5.3 Practical Aspects .................................................... 64
  5.3.1 Stability of the Matrix-vector Product ....................... 64
  5.3.2 Recovering the Solution ....................................... 65
5.4 Reduction to Positive Definite Systems ......................... 66
  5.4.1 Properties of the Positive Definite Matrices ................ 66
  5.4.2 Equivalence to the Schur Complement ........................ 68
5.5 Choosing the Set of Columns ....................................... 69
  5.5.1 Preliminary Experiments ..................................... 70
  5.5.2 Scaling the Columns ......................................... 73
5.6 Implementation Issues .............................................. 74
  5.6.1 The $LU'$ Factorization ...................................... 74
  5.6.2 Keeping the Set of Columns .................................. 75
  5.6.3 Incomplete $LU'$ Factorization ............................... 76
  5.6.4 Using Indicators .............................................. 77
  5.6.5 Avoiding Dependent Columns .................................. 78
  5.6.6 Computing a Second $LU'$ Factorization ...................... 78
5.7 Identifying Symbolically Dependent Columns .................... 80
  5.7.1 Previous Works ............................................... 81
  5.7.2 Using Key Columns ............................................ 82
  5.7.3 Matching During the Factorization ........................... 83
  5.7.4 Numerical Experiments Using Symbolically Dependent Columns 84
5.7.5 The Preprocessing Step ........................................ 85
5.8 Identifying Symbolically Independent Columns ................. 85
  5.8.1 Key Columns and Independent Columns .................... 87
  5.8.2 Merging Symbolically Independent and Dependent Columns . 87
  5.8.3 Numerical Experiments Using Symbolically Independent Columns .................................................. 88
5.9 Special Structures .................................................. 90
5.10 An Equivalent Preconditioner ..................................... 92
5.11 Conclusion .......................................................... 92

6 Linear System Solvers .................................................. 94
  6.1 The Preconditioned Conjugate Gradient Method ............... 94
  6.2 Projection Methods ................................................ 95
    6.2.1 Projection Methods and Linear Systems .................. 96
    6.2.2 Projection Methods and Eigenvalues ...................... 98
  6.3 The Quasi Minimal Residual Method ............................ 99
  6.4 Implicit Restart for the Arnoldi Factorization ............... 99
  6.5 Implicit Restart for Linear Systems .......................... 101
  6.6 Invariant Subspaces and Preconditioners ..................... 102
  6.7 Implementation Issues for MINRES .............................. 103
    6.7.1 Computing the Residual ................................... 103
    6.7.2 Computing the Approximate Solution .................... 103
    6.7.3 Computing the Shifts ..................................... 104
  6.8 The Eigenvalue Problem Approach ............................... 105
    6.8.1 Introduction ................................................. 105
    6.8.2 Making the Matrix Singular ................................ 106
7 Numerical Experiments and Conclusions 114

7.1 The Set of Test Problems .......................... 114
7.2 Numerical Experiments ............................ 115
7.3 Conclusions ....................................... 125
7.4 Future Work ...................................... 127

Bibliography 130
Tables

3.1 SC205 Matrix-vector products ........................................ 29
3.2 SC205 Matrix-vector products ........................................ 31

4.1 Existing Preconditioners. $B^t = LU$ ............................... 49
4.2 SC205 Matrix-vector products for M4 and modified Preconditioner .. 49

5.1 KEN13 Conjugate Gradient Method Iterations .......................... 71
5.2 KEN13 New Factorization versus Keeping LU ......................... 76
5.3 TRUSS Number of nonzero entries .................................... 79
5.4 TRUSS Updated and Skipped Columns ................................... 84
5.5 TRUSS Flops Count $\times 10^3$ ........................................ 85
5.6 KEN13 Number of Nonzero Entries on the Factorization ............... 89
5.7 KEN13 Flops Count $\times 10^3$ for the Factorization ................... 89
5.8 Time and Flops Count for All Techniques .............................. 89

7.1 Basic Statistics ................................................................ 117
7.2 Comparison between Cholesky and LU approach - Number of Iterations 119
7.3 Comparison between Cholesky and LU approach - Time and Flops .. 120
7.4 DIFFICULT Factorization Heuristics .................................... 124
Chapter 1

Introduction

Interior point methods have been used successfully for solving linear programming problems for about a decade now. Their good performance in practice and their theoretical properties have motivated the implementation of sophisticated codes to solve large scale problems. These methods are effective because they converge in relatively few iterations. However an iteration of an interior point method is more expensive than the iterations of the traditional simplex method.

Each iteration of an interior point method involves the solution of one or more linear systems. For many interior point methods, these systems come from applying Newton's method on the perturbed Karush-Kuhn-Tucker optimality conditions for the linear programming problem. This leads to the augmented indefinite linear system which can be written in a symmetric form.

There are several ways for solving the linear system. The most common approach reduces the indefinite augmented system to a smaller positive definite one called the Schur complement. After this reduction, the solution for the linear system is computed via the Cholesky factorization for the majority of the implementations. The latter is more ill-conditioned and dense than the former and this has motivated the study of methods based upon solving the indefinite system. Another less common strategy is to apply the preconditioned conjugate gradient method to solve the positive definite problem. But the computational results with this approach are not very good due to the highly ill-conditioned systems that appear near a solution of the linear
programming problem. A few implementations approach this problem by solving the augmented system with direct methods. These implementations are more stable than the ones that solve the Schur complement system but they take more computational time to solve the linear systems.

In this work, we study efficient ways to solve the augmented linear system by iterative methods. This approach has been explored only recently. Special attention is given to the choice of a preconditioner. This is a key point in applying iterative methods for solving linear systems. We will show that every preconditioner for the reduced system yields an equivalent preconditioner for the augmented system but the converse is not true. Therefore, we choose to work with the augmented system because of the better opportunity to find an effective preconditioner.

We propose a new class of preconditioners which avoid computing the Schur complement. These preconditioners rely on an $LL^T$ factorization of a subset of columns of the constraint matrix instead. We show some theoretical properties of the preconditioned matrix and reduce it to positive definite systems. The discussion on how to implement these preconditioners efficiently is an important part of this work. Among the several techniques used is the study of the nonzero structure of the matrix to speed up the numerical factorization. We investigate the performance of some particular cases with this preconditioner. Also, we discuss the application of several Krylov type methods for both linear systems and eigenvalue problems. applied to the indefinite system. Finally, we illustrate how the preprocessing step for the linear programming problem improves the performance of the iterative methods.
This thesis is organized as follows. In Chapter 2 we present the primal-dual interior point method, the linear systems that arise from the method and some of the approaches being used to solve these systems. Chapter 3 discusses the implementation issues of a primal-dual interior point method with the exception of the strategies for solving the linear system. In Chapter 4 we study the augmented system in detail with special attention given to preconditioners for iterative methods. We also show why it is important to work with the augmented system. A new class of preconditioners is presented in Chapter 5 and its properties are studied. The computational issues for an efficient implementation are discussed as well. In Chapter 6 we present some of the most popular Krylov type iterative methods for solving linear systems and eigenvalue problems. Chapter 7 contains the results of the computational experiments and the conclusions of this work.

1.1 Notation

We use the following notation throughout this work. Lower case Greek letters denote scalars, lower case Latin letters denote vectors and upper case Latin letters denote matrices. Components of matrices and vectors are represented by the corresponding Greek letter with subscripts. Thus, the scalar $\alpha_{ij}$ represents the $i$th row, $j$th column entry of the matrix $A$. A row or column of a matrix is represented by the corresponding lower case letter with a subscript denoting the row (or column). The symbol $0$ will denote the scalar zero, the zero column vector and the zero matrix, its dimension will be clear from context. The identity matrix will be denoted by $I$, a subscript will determine its dimension when it is not clear from context. The symbol $e_i$ will denote the column vector whose $i$th component is one and all the others components are zero, while the letter $e$ will denote the column vector of all ones, their dimension will
be clear from context as well.

The Range space of the matrix \( A \) will be denoted by \( \mathcal{R}(A) \) and its null space by \( \mathcal{N}(A) \). The Euclidean norm is represented by \( \| \cdot \| \) which will also represent the 2-norm for matrices. The other \( p \)-norms for both vector and matrices are represented by \( \| \cdot \|_p \). The relation \( X = diag(x) \) means that \( X \) is a diagonal matrix whose the diagonal entries are \( (\xi_1, \xi_2, \ldots, \xi_n) \). On the other hand, \( diag(A) \) means the column vector formed by the diagonal entries of \( A \). A capital Latin letter \( B \) as a subscript of a matrix \( A \) represents a square submatrix of \( A \) corresponding to the set of indexes \( B \). Either a superscript or subscript \( k \) for a scalar, vector or matrix will denote their value at the \( k \)th step of an iterative procedure.
Chapter 2

Linear Programming

The literature of linear programming is one of the most extensive in the field of applied mathematics. Reference [10] provides an excellent introduction to this topic. The goal of this Chapter is to present our notation for the problem. The most popular interior point methods and a few preliminary considerations about the implementation of these methods.

2.1 Linear Programming Problems

Consider the linear programming problem in the standard form:

\[
\text{minimize} \quad c^t x \\
\text{subject to} \quad A x = b, \quad x \geq 0.
\]

where \( A \) is a full row rank \( m \times n \) matrix and \( c, b \) and \( x \) are column vectors of appropriate dimension. Associated with problem (2.1) is the dual linear programming problem

\[
\text{maximize} \quad b^t y \\
\text{subject to} \quad A^t y + z = c, \quad z \geq 0.
\]

where \( y \) is a \( m \)-vector of free variables and \( z \) is the \( n \)-vector of dual slack variables.

The optimality conditions for (2.1) and (2.2) can be written as a nonlinear system of equations with some nonnegative variables:

\[
\begin{pmatrix}
A x - b \\
A^t y + z - c \\
X Z \varepsilon
\end{pmatrix} = 0, \quad (x, z) \geq 0.
\]

(2.3)
where $X = \text{diag}(x)$ and $Z = \text{diag}(z)$.

2.2 Primal-Dual Interior Point Methods

Since Karmarkar [29] presented the first polynomial time interior point method for linear programming, many variants of these methods have been appeared. Among these variants, the most successful class is known as primal-dual interior point methods, introduced by Megiddo [38] and developed in [30]. These methods have the same computational effort per iteration as the other classes of interior point methods but they have better theoretical properties for worst case complexity analysis as for instance in [41] and also for asymptotic convergence rate analysis (see [58] among others). These results were partially motivated by the superior performance in practice of the primal-dual method over the primal and the dual approaches as in [34, 37]. The majority of the primal-dual interior point methods found in the literature can be seen as variants of Newton’s method applied to the optimality conditions (2.3):

Method 2.1 (Primal-dual Interior Point Method)

Given $y^0$ and $(x^0, z^0) > 0$.

For $k = 0, 1, 2, \ldots$ do

(1) Choose $\sigma^k \in [0, 1)$ and set $\mu^k = \sigma^k(\frac{\gamma^k}{n})$

where $\gamma^k = (x^k)^t z^k$.

(2) Solve the following linear system:

\[
\begin{pmatrix}
0 & I & A^t \\
Z^k & X^k & 0 \\
A & 0 & 0
\end{pmatrix}
\begin{pmatrix}
\Delta x^k \\
\Delta z^k \\
\Delta y^k
\end{pmatrix}
= \begin{pmatrix}
r_d^k \\
r_c^k \\
r_p^k
\end{pmatrix}
\] (2.4)
where,

\[
\begin{align*}
    r_d^k &= c - A^t y^k - z^k \\
    r_z^k &= \mu^k \epsilon - X^k Z^k \epsilon \\
    r_p^k &= b - A x^k.
\end{align*}
\]

(3) Choose a step length \( \alpha^k = \min(1, \tau^k \rho_p^k, \tau^k \rho_d^k) \)

for \( \tau^k \in (0, 1) \) where

\[
\rho_p^k = \frac{-1}{\min_i \left( \frac{\Delta x_i^k}{x_i^k} \right)} \quad \text{and} \quad \rho_d^k = \frac{-1}{\min_i \left( \frac{\Delta z_i^k}{z_i^k} \right)}.
\]

(4) Form the new iterate

\[
(x^{k+1}, y^{k+1}, z^{k+1}) = (x^k, y^k, z^k) + \alpha^k (\Delta x^k, \Delta y^k, \Delta z^k).
\]

Different choices for \( \sigma^k \) and \( \tau^k \) characterize the various methods. We want \( \tau^k \) to be as close as possible to one without creating numerical instability. Thus, keeping the nonnegative variables away from zero and taking full advantage of the direction. In general \( \tau^k \) is a fixed number chosen from numerical experiments. On the other hand, the parameter \( \sigma^k \), sometimes called centering parameter represents an opposite philosophy. Its goal is to avoid letting the point get too close to the bounds and hence, allowing the method to obtain good directions for the incoming iterations. The particular choice \( \sigma^k = 0 \) leads to the affine version of the method. Notice that when the full Newton step is acceptable i.e., it maintains \((x, z)\) nonnegative, the method takes the step length \( \alpha^k = 1 \).

This class of primal-dual interior point methods with an initial feasible point and for wise choices of \( \sigma \) and \( \tau \) enjoy very good theoretical properties including, polynomial complexity (see [27] for references to the more important results) and as showed
more recently, fast convergence [58]. In practice it is common to start with an
infeasible point since in general it is very expensive to find a feasible one. A feasible
point is a point whose nonnegative variables are strictly positive and the residuals \(r_p\)
and \(r_d\) are zero. The theory for these methods with an infeasible starting point is a
current area of research. Some good theoretical results have been obtained thus far
although they are not as good as the results for a feasible starting point. see [57] and
references therein.

Also, it has been found in practice that it is better to separate the step length for
the primal and dual problems. Thus, we define a primal step length \(\alpha_p^k = \min(1, \tau^k \rho_p)\)
and a dual step length \(\alpha_d^k = \min(1, \tau^k \rho_d)\) which are used compute

\[
\begin{align*}
x^{k+1} &= x^k + \alpha_p^k \Delta x^k \\
y^{k+1}, z^{k+1} &= (y^k, z^k) + \alpha_d^k (\Delta y^k, \Delta z^k)
\end{align*}
\]
on step (4) of the method. This separation of step length is ad-hoc but has been
found to work well in practice.

### 2.3 The Predictor-Corrector Variant

The variant of the primal-dual interior point method just described is no longer the
most commonly used version in practice. Soon after its appearance, the predictor-
corrector variant [40] has become the most popular choice. The predictor-corrector
approach differs from the method described in 2.1 in the choice of the search direc-
tions. The new search directions are obtained by solving two linear systems. First
we solve
\[
\begin{pmatrix}
0 & I & A^t \\
Z^k & X^k & 0 \\
A & 0 & 0
\end{pmatrix}
\begin{pmatrix}
\Delta \hat{z}^k \\
\Delta \hat{z}^k \\
\Delta \hat{y}^k
\end{pmatrix}
= \begin{pmatrix}
r_d^k \\
r_a^k \\
r_p^k
\end{pmatrix}
\]
where
\[
\begin{align*}
  r_d^k &= c - A^t y^k - z^k \\
  r_a^k &= -X^k Z^k \epsilon \\
  r_p^k &= b - Ax^k
\end{align*}
\]
and $\Delta \hat{z}^k$, $\Delta \hat{z}^k$ and $\Delta \hat{y}^k$ are called affine directions. Then, the search directions are given by
\[
\begin{pmatrix}
0 & I & A^t \\
Z^k & X^k & 0 \\
A & 0 & 0
\end{pmatrix}
\begin{pmatrix}
\Delta x^k \\
\Delta z^k \\
\Delta y^k
\end{pmatrix}
= \begin{pmatrix}
r^k_d \\
r^k_a \\
r^k_p
\end{pmatrix}
\]
where
\[
r^k_m = \mu^k \epsilon - X^k Z^k \epsilon - \Delta \hat{x}^k \Delta \hat{z}^k \epsilon
\]
and $r^k_d$, $r^k_p$ are the same as before.

This variant reduces significantly the number of iterations for the primal-dual method. This reduction is obtained at the price of solving two linear systems per iteration. However, these linear systems have the same matrix in common. It was found in practice that the work for solving the extra linear system is easily compensated by the savings on the reduction of the number of iterations.

In order to avoid confusion, from now on we refer to the primal-dual interior point method described early in this work as the standard variant. The method just described is referred to as the predictor-corrector variant.
2.4 Computing the Search Directions

In the following discussion we will be restricted to the standard primal-dual method. We will point out where the predictor-corrector variant differs from this discussion in a separate section.

The key step in a given iteration in terms of computational cost is the solution of linear system (2.4). By eliminating the variables $\Delta z^k$ the system reduces to:

\[
\begin{pmatrix}
-D^k & A^t \\
A & 0
\end{pmatrix}
\begin{pmatrix}
\Delta x^k \\
\Delta y^k
\end{pmatrix}
= \begin{pmatrix}
 r^k_d - (X^k)^{-1}r^k_c \\
r^k_p
\end{pmatrix}
\]

(2.5)

where $D^k = (X^k)^{-1}Z^k$ and $\Delta x^k$ can be recovered by

\[
\Delta z^k = (X^k)^{-1}(r^k_c - Z^k \Delta x^k)
\]

We refer to (2.5) as the augmented system. Eliminating $\Delta x^k$ from (2.5) we get

\[
S^k \Delta y^k = r^k_p + A((D^k)^{-1}r^k_d - (Z^k)^{-1}r^k_c).
\]

(2.6)

In (2.6) $S^k \equiv A(D^k)^{-1}A^t$ is called the Schur complement and $\Delta x^k$ can be recovered by

\[
\Delta x^k = (D^k)^{-1}(A^t \Delta y^k - r^k_d - (X^k)^{-1}r^k_c).
\]

We like to stress that $D^k$ is a diagonal matrix with positive entries and it is the only change in the matrices of systems (2.5) and (2.6) at each iteration. Several entries of $D^k$ converge to zero as the method approaches a solution while other entries tend to infinity. The same type of systems appear in other interior point methods such as the dual and primal versions [55]. For instance, on the primal affine version [13] we have $D^k = (X^k)^2$ and the right hand side is given by $(c^t o^t)^t$ on (2.5) and for the dual affine method [1], $D^k = (Z^k)^2$ and the right hand side is given by $(0^t b^t)^t$. 
Notice that for the affine methods the centering parameter is zero. These methods do not share the nice theoretical properties of the primal-dual method and have been outperformed by it on numerical experiments \cite{37}.

### 2.5 Approaches for Solving the Linear Systems

#### 2.5.1 Matrix Factorizations

The most common approach for computing the search directions for the primal-dual interior point methods is solving \((2.6)\) by direct methods and then recovering \(\Delta x^k\) and \(\Delta z^k\). Direct methods for linear systems consist on decomposing a permutation of the matrix into the product of two triangular matrices \(PAQ = LU\) and then solving the system by back substitution. This procedure is known as \(LU\) factorization and it will be described in detail later.

It is possible to compute an unique factorization for symmetric positive definite matrices such that \(LU = L^t\) where the diagonal entries of \(L\) are positive. This is known as the Cholesky factorization. There is a variant that expresses the matrix as the product of two unit triangular matrices and a diagonal matrix. For symmetric positive definite matrices this variant gives \(LDL^t\).

#### 2.5.2 Solving the Schur Complement

Using the Cholesky factorization of \(S^k\) to solve for the search directions in interior point methods is by far the most widely used approach (see for example \cite{1, 34, 35, 37}). This approach takes full advantage of \(S^k\) being symmetric and positive definite. However, \(S^k\) can have much less sparsity and is often more ill-conditioned than the matrix of system \((2.5)\). The extreme case for loss of sparsity consists of \(A\) having a
full column leading to a full Schur complement matrix.

One way around this problem is to use iterative methods. These methods consist of constructing an iterative sequence of approximate solutions for the linear system until a desirable tolerance is achieved. Since these methods require the matrix only for computing matrix-vector products there is no need to compute the Schur complement unless the preconditioner* depends on it. Therefore, loss of sparsity may not be an issue for this approach.

The preconditioned conjugate gradient method is the most popular of the iterative methods for solving positive definite systems. The conjugate gradient method is easy to implement and will converge rapidly with a good preconditioner. Attempts to solve (2.6) using the preconditioned conjugate gradient method have achieved mixed results [35, 39], mainly because the linear systems become highly ill-conditioned as the interior point method approaches an optimal solution. Another disadvantage of these implementations is that they rely on the computation of the Schur complement to build the preconditioner. Thus, they can suffer from loss of sparsity in the presence of a few dense column.

2.5.3 Solving the Augmented System

For the reasons stated above several researchers have begun to consider the augmented system even though it is indefinite. Consequently, the Cholesky factorization can not be applied since there is no numerically stable way to factor a general indefinite matrix onto $LDL'$ with $D$ diagonal. However, the reduction to the Schur complement system is embedded in this approach given that the diagonal entries of $D$ are chosen

*Preconditioning will be discussed in detail on subsequent chapters.
as pivots [55]. Stability considerations apart. A similar result regarding preconditioners for iterative methods will be shown later.

The Bunch-Parlett factorization [7] is a direct method which has been used for solving the indefinite system. It avoids the above mentioned problems by writing the matrix as $LDL^t$ where $L$ is a triangular matrix and $D$ a block diagonal matrix whose blocks have size $1 \times 1$ and $2 \times 2$. A stable factorization is achieved by choosing the diagonal blocks in such way that the growth of the elements in the reduced matrices is controlled. By growth we mean the size of the largest entry of the factored matrix compared to the size of the largest entry of the original matrix. The choice of the diagonal blocks usually involves a symmetric permutation of rows and columns.

Implementations using the Bunch-Parlett factorization proved to be more stable but they are slower than solving (2.6) (see [19, 24, 55]). A multifrontal approach applied to the augmented system has been investigated in [16].

The conjugate gradient method is not well defined for indefinite systems. Thus, it is not used for solving (2.5). MINRES and SYMMLQ [43] are two iterative methods for solving symmetric indefinite systems. Extensions of the conjugate gradient method are presented in [32, 33] but the numerical properties of these methods are inferior to MINRES and SYMMLQ. In [24], SYMMLQ is used to solve the augmented system for a few small problems. Some of the computational results presented in this work use MINRES for solving the augmented system. We are not aware of any successful implementation of an iterative method for solving the indefinite system arising in interior point methods for large scale linear programming problems. However, this is a current area of research and it will be no surprise if good results start to appear.
For instance, in [21] it is mentioned that computational results for large scale problems for the preconditioner outlined there will be reported in a forthcoming paper.

The primary reason for this lack of results is that there has been only limited success with the preconditioners proposed so far. Later we will study preconditioners for system (2.5) and will see that all preconditioners for the positive definite system are in some sense, embedded in those for the indefinite system. We will then propose a new class of preconditioners and study their behavior.

2.5.4 Search Directions for the Predictor-Corrector Variant

All the above discussion applies to the predictor-corrector variant since the structure of the matrices for the interior point methods discussed here is the same.

One important difference is that much work can be saved by computing only one factorization per iteration for the predictor-corrector variant. The reason is that both linear systems share the same matrix thus, the factorization can be used for both. This also applies for the computation of the Schur complement for the approaches where it is needed. That is the primary reason for popularity of this variant.

It is not completely clear whether the predictor-corrector variant is the most appropriated for the approaches using iterative methods to solve the linear systems. It depends on the expense of computing the preconditioner relative to the work for solving both linear systems and how many interior point iterations the predictor-corrector variant saves relative to the standard approach.
The linear systems are related since they have the same matrix and the right hand side of the second linear system depends on the solution of the first. With that in mind, it might be possible to obtain better performance when computing the solution of the second linear system. Apparently, there is no obvious practical way for accelerating the computation of this solution probably because the nonlinearity of the perturbation on the right hand side. Thus, the idea of finding a better initial guess for the second linear system will no longer be studied in this work. Nonetheless, it seems to be a promising subject for future research.

2.6 Modeling Linear Programming Problems

Few linear programming problems arise naturally in the standard form (2.1). In practice we have to consider inequality constraints, range constraints, bounded and free variables. It is possible to transform these type of problems into the standard form, but this increases their dimension. In this section we discuss the changes in the interior point method when considering special structures. These structures will cover all the cited cases except the range constraint which usually is reduced to the bounded variable case by adding one variable for each of these constraints. that is, given the range constraint $a_t \leq a^t x \leq a_u$, we replace it by $a^t x - \omega = 0$ and $a_t \leq \omega \leq a_u$. Thus, with respect to the size of the problem, we replace an inequality constraint by a bounded variable with a single entry in the constraint matrix.

The final goal is to obtain a version for the primal-dual method that works with the special cases without increasing the dimension of the problem. This will be achieved by transforming these problems into the standard form and applying the primal-dual interior point method to the optimality conditions for the transformed problem. Finally, the new variables are eliminated from the linear system in the same
way it is done for $\Delta z^k$ to obtain the augmented system. In order to avoid unnecessary complications, the three cases above will be studied separately and combined later.

### 2.6.1 Inequality Constraints

Consider a linear programming problem where all constraints are inequalities. Without loss of generality, we can assume that all constraints are of the same type:

$$
\begin{align*}
\text{minimize} & \quad c^t x \\
\text{subject to} & \quad Ax \geq b, \quad x \geq 0.
\end{align*}
$$

Associated with this problem is the dual linear programming problem

$$
\begin{align*}
\text{maximize} & \quad b^t y \\
\text{subject to} & \quad A^t y + z = c, \quad y \geq 0, \quad z \geq 0.
\end{align*}
$$

Defining the primal slack variables $w = Ax - b$, the optimality conditions for both problems can be written as

$$
\begin{pmatrix}
Ax - w - b \\
A^t y + z - c \\
XZ\varepsilon \\
WY\varepsilon
\end{pmatrix} = 0, \quad x, w, z, y \geq 0.
$$

Applying the primal-dual method to this problem leads to the following linear system

$$
\begin{pmatrix}
0 & I & 0 & A^t \\
Z^k & X^k & 0 & 0 \\
0 & 0 & Y^k & W^k \\
A & 0 & -I & 0
\end{pmatrix}
\begin{pmatrix}
\Delta x^k \\
\Delta z^k \\
\Delta w^k \\
\Delta y^k
\end{pmatrix} =
\begin{pmatrix}
r_d^k \\
r_c^k \\
r_i^k \\
r_p^k
\end{pmatrix}
$$

where, $r_i^k = \mu^k \varepsilon - W^k Y^k \varepsilon$. Now, by eliminating $\Delta z^k$ and $\Delta w^k$, we obtain

$$
\begin{pmatrix}
-D^k & A^t \\
A & E^k
\end{pmatrix}
\begin{pmatrix}
\Delta x^k \\
\Delta y^k
\end{pmatrix} =
\begin{pmatrix}
r_d^k - (X^k)^{-1} r_c^k \\
r_p^k + (Y^k)^{-1} r_i^k
\end{pmatrix}
$$

(2.8)
where \( D^k = (X^k)^{-1}Z^k \) and \( E^k = (Y^k)^{-1}W^k \). Equation (2.8) is the augmented system for inequality constraints. For the Schur complement we have

\[
S^k \Delta y^k = r^k_p + (Y^k)^{-1}r^k_i + A((D^k)^{-1}r^k_t - (Z^k)^{-1}r^k_c)
\]

where \( S^k = A(D^k)^{-1}A^t + E^k \).

This model allows the matrix \( A \) to have more rows than columns. In that case, we can eliminate \( \Delta y^k \) on (2.8) and define the Schur complement

\[
\tilde{S}^k = -(A^t(E^k)^{-1}A + D^k)
\]

instead. This is equivalent to considering the dual (2.7) as the original problem.

Another possible consequence of eliminating the primal slack variables \( w \) is getting a matrix that is not full row rank. For instance consider the constraints \( a^t x = \lambda_1 \) and \( a^t x \geq \lambda_2 \). By adding a slack variable to the second constraint we obtain two linearly independent rows. But if we decide to handle slack variables implicitly, we have to eliminate the second constraint or conclude that the problem is infeasible before applying the interior point method.

In this instance the problem is easy to spot. In a more general context, we may have a set of linearly dependent rows combining equalities and inequalities which is not easy to expose without resorting to expensive computational work. The technique of identifying these and other type of problems before starting the linear programming method itself is called preprocessing. We shall say more about preprocessing later.
Other changes in the method include the computation of the step length, in order to keep \( w^k \) and \( y^k \) positive, and the perturbation \( \mu^k \). The perturbation now has to take into account these nonnegative variables since the dimension of the problem and the computation of the gap \( \gamma^k \) have changed. Similar results can be obtained for constraints of the type \( Ax \leq b \).

### 2.6.2 Bounded Variables

Consider the linear programming problem where all variables have finite upper bound:

\[
\begin{align*}
\text{minimize} & \quad c^t x \\
\text{subject to} & \quad Ax = b, \quad 0 \leq x \leq u.
\end{align*}
\]

Associated with this problem is the dual linear programming problem

\[
\begin{align*}
\text{maximize} & \quad b^t y - u^t t \\
\text{subject to} & \quad A^t y - t + z = c, \quad t \geq 0, z \geq 0.
\end{align*}
\]

Defining \( v = u - x \), the optimality conditions for both problems can be written as

\[
\begin{pmatrix}
Ax - b \\
u - x - v \\
A^t y - t + z - c \\
XZ\epsilon \\
VT\epsilon
\end{pmatrix}
= 0, \quad (x, v, z, t) \geq 0.
\]

Applying the primal-dual method to this problem leads to the following linear system

\[
\begin{pmatrix}
0 & I & -I & 0 & A^t \\
Z^k & X^k & 0 & 0 & 0 \\
I & 0 & 0 & I & 0 \\
0 & 0 & V^k & T^k & 0 \\
A & 0 & 0 & 0 & 0
\end{pmatrix}
\begin{pmatrix}
\Delta x^k \\
\Delta z^k \\
\Delta t^k \\
\Delta v^k \\
\Delta y^k
\end{pmatrix}
= \begin{pmatrix}
r_d^k \\
r_c^k \\
r_u^k \\
r_b^k \\
r_p^k
\end{pmatrix}
\]

(2.9)
where $r^k_b = \mu^k e - V^k T^k e$ and $r^k_u = u - x^k - i^k$. Now, by eliminating $\Delta z^k$, $\Delta u^k$ and $\Delta t^k$, we obtain

\[
\begin{pmatrix}
-D^k & A^t \\
A & 0
\end{pmatrix}
\begin{pmatrix}
\Delta x^k \\
\Delta y^k
\end{pmatrix}
= \begin{pmatrix}
r^k_d - (X^k)^{-1} r^k_c + (V^k)^{-1} r^k_h \\
0
\end{pmatrix}
\]  

(2.10)

where $D^k = (X^k)^{-1} Z^k + (V^k)^{-1} T^k$. Equation (2.10) is the augmented system for bounded variables. For the Schur complement we have

\[
S^k \Delta y^k = r^k_p + A(D^k)^{-1}(r^k_d - (X^k)^{-1} r^k_c + (V^k)^{-1} r^k_h)
\]

where $S^k = A(D^k)^{-1} A^t$.

Other changes in the method include the computation of the step length in order to keep $r^k$ and $t^k$ positive and the perturbation $\mu^k$ just like in the inequality constraint case. Similar results can be obtained for lower bounds not equal to zero.

### 2.6.3 Free Variables

We now will consider the linear programming problem where all variables are free i.e. there is no constraint on their value:

\[
\text{minimize} \quad c^t x \\
\text{subject to} \quad A x = b.
\]

Associated with this problem is the dual linear programming problem

\[
\text{maximize} \quad b^t y \\
\text{subject to} \quad A^t y = c.
\]

This class of problems rarely will occur in practice, but we shall study it for the sake of completeness in order to understand the effect of free variables on the augmented
system. The optimality conditions for both problems can be written as

\[
\begin{pmatrix}
Ax - b \\
A^T y - c
\end{pmatrix} = 0.
\]

Applying the primal-dual method to this problem leads to the following linear system

\[
\begin{pmatrix}
0 & A^T \\
A & 0
\end{pmatrix}
\begin{pmatrix}
\Delta x^k \\
\Delta y^k
\end{pmatrix} =
\begin{pmatrix}
r_d^k \\
r_p^k
\end{pmatrix}
\]  
(2.11)

equation (2.11) is the augmented system for free variables.

Notice that the Schur complement as we had before is not well defined in the presence of free variables. This matrix has other interesting characteristics like being singular for any rectangular matrix \( A \) and remaining fixed for all iterations of the interior point method. However, as it was said before, we do not expect to find a problem of this type in practice since the primal variables often represent goods, resources or have some other physical meaning where negative values make no sense. Therefore, we shall not study this system further.

There are other reasons for working with free variables in the model besides avoiding an increase in dimension. The traditional way to handle a free variable \( \xi_j \) is to split it into two nonnegative variables \( \xi_j = \xi_j^+ - \xi_j^- \) where \( \xi_j^+, \xi_j^- \geq 0 \). This partition leads to an unbounded set of feasible points since for a solution with \( (\xi_j^+, \xi_j^-) \), the point \( (\xi_j^+ + \epsilon, \xi_j^- + \epsilon) \) for any \( \epsilon > 0 \) will be a solution too. This is a problem because the current theory for interior point methods assumes the existence of a strictly feasible primal-dual point which implies a bounded feasible set [57]. Moreover, the dual variables \( \zeta_j^+ \) and \( \zeta_j^- \) associated with the splitting variables will be zero at the solution as we can see from the optimality conditions. Thus, those variables will have small
values close to the solution which can potentially cause numerical problems. It has been observed in practice [55] that implementations which adopt splitting for free variables have trouble solving these problems.

2.6.4 The General Case

In this section a more general linear programming problem is discussed. The variables can be nonnegative, bounded or free, while the constraints can be equalities or inequalities. Following the previous discussion, all lower bounds are assumed to be zero except for free variables and all inequality constraints to be of the "greater than" type. Thus, consider the following linear programming problem

\[
\begin{align*}
\text{minimize} & \quad c^t x \\
\text{subject to} & \quad A_1 x = b_1 \\
& \quad A_2 x \geq b_2 \\
& \quad x_1 \geq 0. \\
& \quad 0 \leq x_2 \leq u_2. \\
& \quad x_3 \text{ free}
\end{align*}
\]

where \( x = (x_1^t, x_2^t, x_3^t)^t \).

Associated with this problem is the dual linear programming problem

\[
\begin{align*}
\text{maximize} & \quad b_1^t y_1 + b_2^t y_2 - u_2^t l_2 \\
\text{subject to} & \quad\left( \begin{array}{ccc}
A_{11}^t A_{21}^t \\
A_{12}^t A_{22}^t \\
A_{13}^t A_{23}^t
\end{array} \right) \left( \begin{array}{c}
y_1 \\
y_2 \\
y_3
\end{array} \right) + \left( \begin{array}{c}
z_1 \\
z_2 - l_2 \\
0
\end{array} \right) = \left( \begin{array}{c}
c_1 \\
c_2 \\
c_3
\end{array} \right)
\end{align*}
\]

\( y_1 \text{ free.} \)

\( y_2, l_2, z_1, z_2 \geq 0 \)
where \( c = (c_1^t, c_2^t, c_3^t)^t \) and the \( m \times n \) matrix

\[
A = \begin{pmatrix}
A_1 & A_{12} & A_{13} \\
A_2 & A_{22} & A_{23}
\end{pmatrix}
\]

are partitioned in accordance with the partition of \( x \).

Defining \( w_2 = A_2x - b_2, v_2 = u_2 - x_2 \) and the following vectors with appropriated dimension

\[
\begin{align*}
b &= (b_1^t, b_2^t)^t \\
y &= (y_1^t, y_2^t)^t \\
u &= (0, u_2^t, 0)^t \\
l &= (0, t_2^t, 0)^t \\
z &= (z_1^t, z_2^t, 0)^t \\
w &= (0, (A_2x - b_2)^t)^t \\
v &= (0, (u_2 - x_2)^t, 0)^t
\end{align*}
\]

the optimality conditions for both problems can be written as

\[
\begin{pmatrix}
A_x - w - b \\
u - x - v \\
A^y y - t + z - c \\
XZ\epsilon \\
VT\epsilon \\
WY\epsilon
\end{pmatrix} = 0. \quad x_1, x_2, w_2, v_2, z_1, z_2, t_2, y_2 \geq 0. \quad (2.12)
\]
Applying the primal-dual method to this problem leads to the following linear system

\[
\begin{pmatrix}
  0 & I & -I & 0 & A^t & 0 \\
  Z^k & X^k & 0 & 0 & 0 & 0 \\
  I & 0 & 0 & I & 0 & 0 \\
  0 & 0 & V^k & T^k & 0 & 0 \\
  A & 0 & 0 & 0 & 0 & -I \\
  0 & 0 & 0 & 0 & Y^k & W^k
\end{pmatrix}
\begin{pmatrix}
  \Delta x^k \\
  \Delta z^k \\
  \Delta t^k \\
  \Delta v^k \\
  \Delta y^k \\
  \Delta w^k
\end{pmatrix}
= \begin{pmatrix}
  r_d^k \\
  r_z^k \\
  r_t^k \\
  r_b^k \\
  r_p^k \\
  r_i^k
\end{pmatrix}
\]  

(2.13)

where the residuals are defined as in the previous sections.

Now, by eliminating \( \Delta z^k \), \( \Delta v^k \), \( \Delta t^k \) and \( \Delta w^k \), the augmented system for problem (2.13) will be:

\[
\begin{pmatrix}
  -D^k & A^t \\
  A & E^k
\end{pmatrix}
\begin{pmatrix}
  \Delta x^k \\
  \Delta y^k
\end{pmatrix}
= \begin{pmatrix}
  r_d^k - (X^k)^t r_z^k + (V^k)^t r_b^k \\
  r_p^k + (Y^k)^t r_i^k
\end{pmatrix}
\]  

(2.14)

with

\[
D^k = (X^k)^t Z^k + (V^k)^t T^k \quad \text{and} \quad E^k = (Y^k)^t W^k
\]

where, for a given diagonal matrix \( D \), we define its pseudo-inverse \( D^t \) as

\[
\delta_{ii}^t = \begin{cases} 
\delta_{ii}^{-1} & \text{if } \delta_{ii} \neq 0 \\
0 & \text{otherwise.}
\end{cases}
\]

For the Schur complement we have

\[
S^k \Delta y^k = r_p^k + (Y^k)^t r_i^k + A(D^k)^t (r_d^k - (X^k)^t r_z^k + (V^k)^t r_b^k)
\]
where $S^k = A(D^k)^t A^t + E^k$ provided the primal problem has no free variables and $m \leq n$.

Equation (2.14) is the augmented system for problems with inequality constraints, bounded variables and free variables. It contains no slack variables either for inequality constraints or bounds and there is no need to split the free variables into nonnegative variables. It is the smallest system we can get without further information on $A$ or without changing its sparse pattern as computing the Schur complement does.

2.7 Stable Reduction

It is possible to eliminate $\Delta x^k$ instead of $\Delta z^k$ in (2.4). If we do that and make the change of variables $\Delta z^k = -D^k \Delta z^k$, we obtain the same matrix as in the augmented system with a different right hand side

$$
\begin{pmatrix}
-D^k & A^t \\
A & E^k
\end{pmatrix}
\begin{pmatrix}
\Delta z^k \\
\Delta y^k
\end{pmatrix}
=
\begin{pmatrix}
r_d^k \\
r_p^k - A^t(Z^k)^{-1}r_c^k
\end{pmatrix}
$$

at the price of an extra matrix-vector product.

The motivation of this elimination is to avoid cancellation when computing the right hand side in (2.5) because it involves the inverse of $X^k$ which contains entries that tend to zero as $x^k$ approaches the solution. However, this elimination will introduce a similar problem for the variables $z^k$ since they will appear on the modified right hand side.

Based on this idea, Freund and Jarre [21] suggest what they call the stable reduction scheme for problems in the standard form which consists in eliminating the
direction corresponding to the smallest value between $\xi_i^k$ and $\zeta_i^k$. They also suggest to recover the eliminated $\Delta z^k$ directions by the relation

$$
\Delta z^k = c - A^t(y^k + \Delta y^k) - z^k
$$

(2.15)

instead of

$$
\Delta z^k = (X^k)^{-1}(r_z^k - Z^k \Delta x^k)
$$

which guarantees that the inverse of the smallest value for the pair $(\xi_i^k, \zeta_i^k)$ is never used to compute the right hand side nor to recover the eliminated directions. This relation again costs an extra matrix-vector product. We point out that the (2.15) can be used to compute all the components of $\Delta z^k$ independent of the method of elimination used. Thus, one also avoid working with $(X^k)^{-1}$ for those components which are not eliminated. This option can be used even together with the traditional elimination scheme and it is the one adopted in this work.

The stable reduction idea can be extended to system (2.13) where we can consider the elimination of the largest entries among $(x^k, z^k, \tau^k, \iota^k)$ and between the pairs on $(y^k, w^k)$ in order to obtain the augmented system. Preliminary experiments with this extension did not show a clear improvement in the behavior of the interior point methods.
Chapter 3

Implementation Issues for the Primal-Dual Method

In this chapter we discuss the details of our implementation of the primal-dual interior point method. Here we describe the preprocessing and scaling steps: specify the starting point: the parameters $\sigma$ (perturbation) and $r$ (step length); describe the stopping rule and discuss the computation of the Schur complement. The solution of the linear system is considered in the next chapters.

3.1 Preprocessing

A preprocessing procedure is essential in practical implementations of linear programming solvers. This fact is known almost since the first applications of linear programming. The first work on this subject to appear in the literature is due to Brearley, Mitra and Williams [6]. Most of the techniques are known at least since then. More recently with the advent of interior point methods, new techniques have been developed [2, 54].

Before starting the interior point method itself, the problem is preprocessed not only to reduce its size but to eliminate variables that can be troublesome for the iterative methods, e.g., constraints of the type $\xi_j = 0$. The preprocessor also looks for obvious primal and dual infeasibility. It has the following rules:

1. Check for ranged constraints with zero range gap and change it into equality constraint when appropriate:
2. Eliminate empty rows and columns (variables) from $A$:

3. Remove fixed variables and update the right hand side:

4. Inequality constraints with one entry are changed into bounds:

5. Study rows whose all entries have the same sign and eliminate those inequality constraints that are redundant or set all variables to zero when possible. For instance, let all variables be nonnegative. Thus, the constraint $\xi_1 + \xi_2 \geq 0$ is redundant and the variables on the constraint $\xi_3 + \xi_4 \leq 0$ are fixed.

6. Pivot out one variable for equality rows containing exactly two nonzero entries with distinct sign:

7. Change variables with one entry on the matrix into slacks when it is possible:

8. Look for redundant rows by computing an $LL^T$ factorization of $A$. This technique is very expensive but it is necessary to apply for some of the strategies for solving the linear system presented later.

Rule 1 is applied first then rules 2 – 7 are applied until no elimination occurs then rule 8 is applied once. We will see later row rule 8 is implemented. Notice that the elimination of redundant rows might introduce single entry columns (but never empty columns) on the reduced matrix. Those single entry columns may allow the application of rule 7 again which in turn can allow other rules to be applied, but we believe that it is not worth repeating the preprocessing procedure after the initial factorization. None of the above rules increase the number of nonzero entries of $A$.

The performance of the iterative method can be very sensitive to preprocessing procedures. Table 3.1 compares the performance of the standard primal-dual interior
point method using the MINRES in terms of the number of matrix-vector products per iteration with preprocessing and without it. The test problem is SC205 [28]. It is a staircase problem from a dynamic multi-sector development planning model. It is included on the netlib* collection of linear programming problems [22].

We choose this problem because its size is suitable for experiments with MATLAB, but the results are representative for the other netlib problems we have tried. The augmented system has dimension 522 before preprocessing and 440 after it. The preconditioner used will be described on Chapter 4. It reduces the augmented system into a positive definite system.

All experiments in this chapter are carried out in MATLAB, on a SUN SPARC station IPX. This MATLAB version only works with problems in the standard form (2.1). The floating point arithmetic is IEEE standard double precision. The stopping tolerance for the interior point method and linear system solvers is the square root of epsilon machine. The stopping rules and the parameters are described later in this chapter.

We can conclude from table 3.1 that the preprocessing step will not only reduce the problem size but it will also reduce the total number of matrix-vector products required by MINRES to converge. Therefore, for implementations where the linear system is solved by iterative methods, preprocessing can be advantageous for reducing the amount of work in two ways.

<table>
<thead>
<tr>
<th>IP Iteration</th>
<th>Preprocessing</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No</td>
</tr>
<tr>
<td>1</td>
<td>31</td>
</tr>
<tr>
<td>2</td>
<td>17</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>17</td>
</tr>
<tr>
<td>6</td>
<td>20</td>
</tr>
<tr>
<td>7</td>
<td>49</td>
</tr>
<tr>
<td>8</td>
<td>54</td>
</tr>
<tr>
<td>9</td>
<td>59</td>
</tr>
<tr>
<td>10</td>
<td>58</td>
</tr>
<tr>
<td>11</td>
<td>56</td>
</tr>
<tr>
<td>12</td>
<td>55</td>
</tr>
<tr>
<td>13</td>
<td>50</td>
</tr>
<tr>
<td>14</td>
<td>51</td>
</tr>
<tr>
<td>15</td>
<td>42</td>
</tr>
<tr>
<td>Total</td>
<td>593</td>
</tr>
</tbody>
</table>

Table 3.1 SC'205 Matrix-vector products

3.1.1 Other Preprocessing Techniques

We now present some more sophisticated procedures for preprocessing. These have not been implemented here because they are outside the scope of this work. However, in an industrial application these techniques would have to be considered.

1. Parallel rows and columns. can eliminate equivalent rows or columns and find infeasible problems:

2. Row bound analysis. can detect redundant constraints. infeasibility and turn inequality constraints into equality constraints by studying effect of the variable bounds on the constraints:
3. Column bound strengthening. can find redundant inequality constraints and tight bounds for variables:

4. Aggregation. write a variable as a function of the others in a given row. It is equivalent to the implemented rule 6 but with more than two variables. thus this rule may increase the number of nonzero entries on the constraint matrix.

Additional preprocessing techniques for improving the performance of interior point methods have been proposed recently. One idea [2] is to make \(A\) more sparse by finding suitable linear combinations of its rows. Another idea is to split dense columns [54] thus making the Schur complement less dense.

### 3.2 Special Structures

After preprocessing, range constraints are transformed into equality constraints by adding a new bounded variable for each constraint. The bounds are given by the respective ranges of the constraint. In order to get a cleaner code and a little less overhead, all lower bounds are shifted to zero and inequality constraints are changed to the "\(\geq\)" type if necessary.

### 3.3 Scaling

Scaling consists in changing the problem to reduce the range among the entries on the matrix. After scaling, problem (2.1) will be written as:

\[
\begin{align*}
\text{minimize} & \quad \tilde{c}^T \tilde{x} \\
\text{subject to} & \quad \tilde{A} \tilde{x} = \tilde{b}, \quad \tilde{x} \geq 0.
\end{align*}
\]

with \(\tilde{A} = R^{-1}AC^{-1}, \tilde{x} = Cx, \tilde{b} = R^{-1}b, \tilde{c} = C^{-1}c\) where \(R\) and \(C\) are diagonal matrices.
There are several choices for $R$ and $C$. The purpose of scaling is to obtain better numerical behavior for the linear programming solver and to decrease the total number of iterations. Scaling can neatly improve the performance on some problems and deteriorate it on others. The reason for this erratic behavior is not yet understood.

Next we present the equilibration scaling method which was implemented: Set $\rho_{ii} = \max_j |\alpha_{ij}|$ and $\gamma_{jj} = \max_i \frac{|\alpha_{ij}|}{\rho_{ii}}$. This choice for $R$ and $C$ guarantees that the largest entry in any row or column of $\hat A$ has magnitude one. For other scaling methods and some comparisons among them see [53].

Table 3.2 shows the effect of scaling in a linear programming problem. The augmented system has dimension 440 after preprocessing. The preconditioner used will
be described in Chapter 4. It reduces the augmented system into a positive definite system.

Notice that the primal-dual method took 14 iterations to converge for the non-scaled version, while it needed 15 iterations required to solve the linear programming problem in the scaled version. Nonetheless, the total number of matrix-vector products for the scaled version is smaller giving an overall better result for the scaled version. Thus one of the goals of scaling was achieved with very good results while it not only failed to achieve the other goal but it deteriorated the performance of the interior point method. We chose this problem to stress how unpredictable scaling can be. But overall, it is advisable to adopt it.

3.4 Starting Point

The starting point is another important issue for implementing primal-dual methods. A bad choice can make the method fail to converge. Below, we present the default starting point for OB1 [35] one of the best known implementations of interior point methods. It is adopted as the default for OB1 after extensive experiments with several linear programming problems. The starting point \((t^0, v^0, x^0, y^0, z^0)\) is computed as follows:

\[
\hat{x} = A^t(AA^t)^{-1}b.
\]

\[\epsilon_1 = \max(-\min(\hat{\xi}_j), \epsilon_2, \frac{\|b\|_2}{c_2}) \text{ then:}
\]

\[
\begin{align*}
\xi_j^0 &= \max(\hat{\xi}_j, \epsilon_1), \\
\nu_j^0 &= \max(v_j - \hat{\xi}_j, \epsilon_1).
\end{align*}
\]
let $y^0 = 0, \epsilon_3 = 1 + \|c\|_1$. and for $z^0$ set:

$$
\zeta_j^0 = \begin{cases} 
\gamma_j + \epsilon_3, & \text{if } \gamma_j \geq 0 \\
-\gamma_j, & \text{if } \gamma_j < -\epsilon_3 \\
\epsilon_3, & \text{if } -\epsilon_3 \leq \gamma_j \leq 0 
\end{cases}
$$

finally, for $t^0$

$$
\tau_j^0 = \begin{cases} 
\epsilon_3, & \text{if } \gamma_j \geq 0 \\
\epsilon_3 - \gamma_j, & \text{if } \gamma_j < -\epsilon_3 \\
2\epsilon_3, & \text{if } -\epsilon_3 \leq \gamma_j \leq 0 
\end{cases}
$$

where $\epsilon_2 = 100$.

### 3.4.1 Starting Point and Inequality Constraints

Since OB1 does not handle inequality constraints explicitly we compute the equivalent starting point obtained by adding slack variables:

let $\tilde{w} = -(A^t A)^{-1} b$

$$
\epsilon_1 = \max(- \min(\tilde{\zeta}_j), - \min(\tilde{\omega}_j), \epsilon_2, \frac{\|b\|_1}{\epsilon_2})
$$

$$
\omega_j^0 = \max(\tilde{\omega}_j, \epsilon_1)
$$

let $\eta_j^0 = \begin{cases} 
0 & \text{if } \eta_j \text{ is free} \\
\epsilon_2 & \text{otherwise.} 
\end{cases}$

Notice that OB1 computes a scale dependent starting point. The following modification on the calculation of $\epsilon_1$ is aimed to fix this:

$$
\epsilon_1 = \max(- \min(\tilde{\zeta}_j), - \min(\tilde{\omega}_j), \frac{\|b\|_1}{\epsilon_2 \|A\|_1}).
$$
3.4.2 Starting Point and Bounded Variables

A more important modification concerns bounded variables. Problems which have these kind of variables are more likely to cause numerical difficulties. This is particularly true if there are variables whose bounds are too tight. This is the reason why the starting point in OB1 allows the residual for the bound constraints be non-null. We also noticed that for this starting point, some problems behave better if we do not change simple inequalities into bounds during preprocessing. For all these reasons we compute the initial point as if the bounded constraints were part of $A$, in a similar fashion as it is done for the slack variables. Summarizing, our implementation computes the initial point $(l^0, v^0, w^0, x^0, y^0, z^0)$ as follows:

\[
2\hat{\dot{w}} = -(AA^t)^{-1}(2b - Au)
\]

\[
2\dot{x} = A^t(2b - Au) + u
\]

\[
\dot{c} = u - \dot{x}
\]

\[
\epsilon_1 = \max(-\min(\hat{\dot{x}}), -\min(\hat{\dot{w}}), -\min(\nu), \epsilon_2, \frac{\|b\|_1}{\epsilon_2\|A\|_1})
\]

then:

\[
\begin{align*}
\omega_j^0 &= \max(\omega_j, \epsilon_1) \\
\xi_j^0 &= \max(\xi_j, \epsilon_1) \\
\nu_j^0 &= \max(\nu_j, \epsilon_1)
\end{align*}
\]

let \( \eta_j^0 = \begin{cases} 
0 & \text{if } \eta_j \text{ is free} \\
\epsilon_2 & \text{otherwise}
\end{cases} \)

\[
\zeta_j^0 = \begin{cases} 
\gamma_j + \epsilon_3, & \text{if } \gamma_j \geq 0 \\
-\gamma_j, & \text{if } \gamma_j < -\epsilon_3 \\
\epsilon_3, & \text{if } -\epsilon_3 \leq \gamma_j \leq 0
\end{cases}
\]
and \( r_j^0 = \varepsilon_2 \).

These changes over the OB1 starting point provided slightly better computational results for both the interior point method and the iterative solver for the problems of the netlib collection tried.

3.4.3 The Augmented System Representation for the Starting Point

This approach involves the solution of a positive definite linear system whose matrix \( A^t A \) has the same structure of the Schur complement. However, this system can be rewritten in the equivalent form:

\[
\begin{pmatrix}
I & A^t \\
A & 0
\end{pmatrix}
\begin{pmatrix}
2\dot{x} - u \\
2\dot{w}
\end{pmatrix}
= \begin{pmatrix}
0 \\
2b - Au
\end{pmatrix}
\]

which has an equivalent structure of the augmented system (2.5). Thus, we have the same options to find a starting point as we have for the iterations of the interior point methods. solve an augmented linear system or its Schur complement.

3.5 The Parameters \( \tau \) and \( \sigma \)

The parameter \( \tau^k \) has a fixed value \( \tau = 0.99995 \), the value recommended in [36] after numerical experiments with the netlib test set. For the standard primal-dual method we have for \( \sigma \)

\[
\sigma^k = \begin{cases} 
p^{-\frac{1}{2}} & \text{if } \gamma^k > 1 \\
\gamma^k \cdot p^{-\frac{1}{2}} & \text{else.}
\end{cases}
\]

where the gap \( \gamma^k = (x^k)^t z^k + (u^k)^t t^k + (w^k)^t y^k \) and \( p \) is the the number of complementary pairs. The choice for \( \sigma \) is based on the results of Zhang and Tapia [58] where it is shown that the duality gap sequence converges to zero with Q-rate \( \lambda \) if, among
other assumptions. \( \sigma = O(\gamma^\lambda), \lambda \in (0, 1). \)

For the predictor-corrector variant, the affine direction can be used for computing \( \mu \) since it does not depend on it. Thus, if we define

\[
\hat{z}^k = (x^k + \alpha_p^k \Delta x^k, t^k + \alpha_q^k \Delta t^k)
\]

\[
\mu^k = (\nu^k + \alpha_p^k \Delta \nu^k, y^k + \alpha_q^k \Delta y^k)
\]

then,

\[
\sigma^k = \begin{cases} 
\left(\frac{\hat{z}^k}{\mu^k}\right)^3 & \text{if } \gamma^k > 1 \\
\gamma^k \cdot p^{-\frac{1}{2}} & \text{else}. 
\end{cases}
\]

This rule gives a small perturbation parameter when the affine direction gives good improvement on the gap and a large one otherwise. We preserve the choice for the gap smaller than one given for the standard primal-dual method following [35]. In this work, the rule based on the affine direction is reported to be numerically unstable for some problems.

### 3.6 Stopping Criteria

The stopping criteria is based in the optimality conditions (2.12). It measures the relative primal and dual infeasibility

\[
\left(\begin{array}{c}
\frac{\|b - A x^k + u^k\|}{1 + ||z^k|| + ||u^k||} \\
\frac{||c - A^t y^k - z^k + t^k||}{1 + ||y^k|| + ||z^k|| + ||t^k||} \\
\frac{||u - x^k - \nu^k||}{1 + ||x^k|| + ||t^k||}
\end{array}\right) \leq \epsilon
\]
and the relative gap
\[
\frac{\gamma^k}{1 + |c^t x^k + b^t y^k - u^t t^k|} \leq \epsilon.
\]
This rule works well on practice.

The scale independent rule below can also be used, but we found that it is too restrictive. This is probably because it is common for \( b \) and \( c \) to have very few nonzero entries. Actually there are problems where all entries of \( b \) are null.

\[
\left( \frac{\|p - Ax^k + x^k\|}{1 + \|b\|} \right) \leq \epsilon
\]

\[
\left( \frac{\|c - A^t y^k - z^k + t^k\|}{1 + \|c\|} \right) \leq \epsilon
\]

\[
\left( \frac{\|u - z^k - \xi^k\|}{1 + \|a\|} \right) \leq \epsilon
\]

3.7 The Schur Complement

For implementations that compute the Schur complement \( S \), it is advisable to determine its nonzero pattern before starting the first iteration since it remains unchanged for the whole process. Thus, given that

\[
\sigma_{ij} = \sum_{k=1}^{n} \frac{a_{ik} a_{jk}}{\delta_{kk}} + \epsilon_{ij}.
\]

there is a nonzero entry on \( S \) for each pair of rows of \( A \) whose inner product sum \( a_i a_j^t \) has at least a nonzero entry. Remember that \( E \) is diagonal and thus it does not affect the nonzero structure of \( S \). Incidentally, this relation explains why a dense column on \( A \) introduce many nonzero entries on the Schur complement.
It is possible to create a list for each nonzero entry $\sigma_{ij}$ containing the nonzero products $\sigma_{ik}\sigma_{jk}$ which also remain fixed. This list saves half of the floating point multiplication for computing the Schur complement.
Chapter 4

The Augmented System

In this chapter we will study the properties of the augmented system and discuss preconditioning in detail. We present some of the existing preconditioners for the augmented system. Finally, we show that every preconditioner for the Schur complement system yields an equivalent preconditioner for the augmented system. We also show that the converse is not true. This is an important result and it is the main motivation for the development of the a new class of preconditioners for the augmented system which we shall present in the next chapter.

4.1 Introduction

A slightly more general form for the augmented system (2.5) arises naturally in several areas of applied mathematics such as optimization, fluid dynamics, electrical networks, structural analysis and heat equilibrium. In some of these applications, the matrix \( D \) is not necessarily diagonal although it is symmetric. The techniques for solving this system varies widely within these areas since the characteristics of the system changes with the problems. See for example [5, 18, 45, 52, 56] among several others. Still, these systems have several properties in common as we will see next.

In this chapter, we introduce an important concept used to measure the sensitivity of a matrix \( A \), the condition number \( \kappa_p(A) \) defined by

\[ \kappa_p(A) = \|A\|_p\|A^{-1}\|_p. \]
If $\kappa_p(A)$ is large the matrix $A$ is said to be ill-conditioned. Notice that for symmetric positive definite matrices the condition number for the 2-norm is given by the ratio of its largest and smallest eigenvalue.

### 4.2 Properties of the Augmented System

The augmented system is nonsingular if its first $n$ columns are linearly independent. This is always the case when $D$ is a diagonal matrix with no zero diagonal entries and $A$ has full row rank. The augmented system is also clearly indefinite. Moreover, it can be seen from the relation

$$
\begin{pmatrix}
-D & 0 \\
A & I
\end{pmatrix}
\begin{pmatrix}
-D^{-1} & 0 \\
0 & AD^{-1}A^t + E
\end{pmatrix}
\begin{pmatrix}
-D & A^t \\
0 & I
\end{pmatrix}
= 
\begin{pmatrix}
-D & A^t \\
A & E
\end{pmatrix}
$$

that by Sylvester's law of inertia [26] it has $m$ positive and $n$ negative eigenvalues if $D$ is positive definite and $E$ is positive semi-definite.

It can easily be verified that

$$
\begin{pmatrix}
-D & A^t \\
A & E
\end{pmatrix}^{-1} = 
\begin{pmatrix}
-D^{-1} + D^{-1}A^tS^{-1}AD^{-1} & D^{-1}A^tS^{-1} \\
S^{-1}AD^{-1} & S^{-1}
\end{pmatrix}
$$

(4.1)

where $S = AD^{-1}A^t + E$. Later, we shall introduce a preconditioner based on the inverse of the augmented system.

In [4], Björck studies the matrix

$$
\begin{pmatrix}
\alpha I & A^t \\
A & 0
\end{pmatrix}
$$
where $0 \leq \alpha \leq \|A\|$. Showing that its eigenvalues are
\[
\begin{cases}
\alpha \text{ with multiplicity } n - m \\
\frac{a^2}{2} \pm (\frac{a}{2} + \lambda_i)^\frac{1}{2}
\end{cases}
\]
where $\lambda_i$ are the eigenvalues of $A A^t$. Björck gives an optimal value for the scaling parameter $\alpha$ to minimize the condition number of the matrix. Since the matrix is symmetric, the condition number $\kappa_2(A A^t) = \frac{\sigma_{\text{max}}}{\sigma_{\text{min}}}$ gives the value $\alpha = (\frac{\lambda_{\text{max}}}{\lambda_{\text{min}}})^\frac{1}{2}$. It is possible to generalize this idea to the augmented system by modifying it in the following way:
\[
\begin{pmatrix}
D^{-\frac{1}{2}} & 0 \\
0 & I
\end{pmatrix}
\begin{pmatrix}
D & A^t \\
A & 0
\end{pmatrix}
\begin{pmatrix}
D^{-\frac{1}{2}} & 0 \\
0 & I
\end{pmatrix}
= 
\begin{pmatrix}
I & D^{-\frac{1}{2}} A^t \\
A D^{-\frac{1}{2}} & 0
\end{pmatrix}
\]
but as he pointed out this system is highly ill-conditioned when $D$ has very small entries since then its eigenvalues depend on the eigenvalues of the Schur complement $A D^{-\frac{1}{2}} A^t$.

4.3 Preconditioning

4.3.1 Motivation

Iterative methods are very sensitive to the condition number of the matrix. For instance, the following error bound can be obtained for the conjugate gradient method [26]:
\[
\|x - x_k\|_A \leq 2 \|x - x_0\|_A \alpha^k
\]
where $\|w\|_A^2 \equiv w^t A w$ and $\alpha = \frac{\sqrt{\kappa_2(A)}}{\sqrt{\kappa_2(A) + 1}}$. Thus, the conjugate gradient behaves very well when $\kappa_2(A) \approx 1$. 
A matrix with a large condition number is said to be *ill-conditioned*. In most applications it is essential to modify an ill-conditioned linear system into an equivalent better conditioned system. Otherwise, the iteration may be very slow or even fail to converge. This is done in such a way that it is easy to recover the original system solution from the modified one. The technique just described is known as preconditioning.

### 4.3.2 An Introduction on Preconditioning

Consider the following situation: given $Ax = b$, we solve the equivalent linear system $M^{-1}A\mathcal{N}^{-1}\tilde{x} = \tilde{b}$, where $\tilde{x} = \mathcal{N}x$ and $\tilde{b} = M^{-1}b$. The system is said to be preconditioned and $M^{-1}A\mathcal{N}^{-1}$ is called the preconditioned matrix. Notice that usually it is not necessary to compute the preconditioned matrix since most of the iterative methods only access the matrix to compute matrix-vector products.

We say that a preconditioner is symmetric if $\mathcal{N}^t = \mathcal{N}$ because in that case if $A$ is symmetric, the preconditioned matrix $M^{-1}AM^{-t}$ will also be symmetric. Due to Sylvester's law of inertia, the preconditioned matrix has the same number of positive (or negative) eigenvalues as the original matrix for symmetric preconditioners. Thus, symmetric preconditioners cannot be used to change an indefinite system into a positive definite one.

To accomplish our goal of getting fast convergence, $M^{-1}A\mathcal{N}^{-1}$ must be closer to the identity matrix than $A$. Another aim to be considered is to move the eigenvalues of $A$ closer to plus or minus one since the rate of convergence of iterative methods usually depend on the eigenvalue distribution as the previous result for the conjugate gradient shows (4.2). On the other hand, it is preferable for practical reasons that
$M$ and $N$ be easy to work with. that is, we should be able to easily solve linear systems with them. Otherwise, the iteration time of the preconditioned linear system solver can become too high. Hence, the savings from the reduction in the number of iterations will be overcome by the expense in solving the linear systems. Because of that, the preconditioning matrices are often constructed to be block diagonal or triangular.

Another issue to be considered is the cost of computing the preconditioners since the final goal is to minimize the total computational time. However, this is usually amortized over the total computation time.

Observe that both conditions that determine an effective preconditioner are conflicting. For instance, a preconditioner whose linear system solution is too easy to obtain is not likely to provide enough improvement of the eigenvalue spectrum of the matrix in order to give a much faster convergence. This feature can make the search for a good preconditioner a very difficult task.

It should be clear by now that the ideal preconditioner is highly dependent on the problem being solved and often there is no recipe for choosing the best one, although there are some general directions to guide us towards a good choice. Other important aspects we did not discussed in detail are the size of the system, the sparsity pattern of the matrix and dealing with multiple right hand sides. Another one, particularly interesting for the problem we are studying, is the solution of many close related systems. In this situation, work may be saved on computing successive preconditioners and perhaps the knowledge obtained by solving a linear system may be used to compute better preconditioners for the next ones.
4.3.3 Some Classes of Preconditioners

There are several classes of preconditioning methods. In this section we are mainly concerned with the class of preconditioners called incomplete factorization methods. This approach has been used successfully to develop preconditioners for problems similar to ours. Usually these preconditioners have been applied to the Schur complement.

It is well known that given the right permutation, any matrix $A$ can be written as a product of a lower triangular $L$ and an upper triangular matrix $U$. that is we can write $PA = LU$, where $P$ is a permutation matrix. i.e. the identity matrix with its rows reordered. The left product of $A$ by a permutation matrix results in the permutation of its rows. Similarly, the right product by a permutation matrix permutes the columns of $A$.

The permutation of the rows of $A$ is necessary to avoid nonzero pivots and advisable for stability purposes by avoiding small pivots whenever it is possible. This product is called $LU$ factorization and it is not unique. For symmetric positive definite matrices, the Cholesky factorization described before is a popular option. The symmetric property can be kept by simultaneously permuting both, rows and columns of $A$ giving $PAP^t = LL^t$. This factorization is known to be stable for any symmetric permutation because for positive definite matrices with diagonal pivots no growth occur [14]. This is an important concept because the round-off error for the linear system can be bounded by a function of the growth factor, that is the quotient between the largest entry of the factored and the original matrix.
The sparsity pattern of $L$ does not necessarily coincide which the sparsity pattern of the lower half of $A$. For instance, if $a_{ij} = 0$ and both $a_{ik}$ and $a_{kj}$ are nonzero we may have at some point of the factorization the update

$$
\lambda_{ij} = -\frac{a_{ik}}{a_{kj}}
$$

which is nonzero. The nonzero entries on $L$ and $U$ which are zero on $A$ are called \textit{fill-ins}. In some matrices the number of fill-ins can be prohibitively high causing the computation of the full factorization infeasible in an acceptable time. The number of fill-ins varies with the permutation of $A$. The permutation that minimizes the number of fill-ins is too expensive to compute. Actually, it is a harder problem than solving the linear system problem itself. However, there are several successful heuristic methods for finding good permutations [14]. The most popular heuristic, called minimum degree ordering, chooses as the next pivot the entry which will generate the least number of fill-ins of the current step among all choices.

Observe that the permutation is independent of the value of the nonzero entries. This property can lead to computational savings when many linear systems are solved for matrices with the same nonzero structure such as in the interior point methods.

The idea of incomplete factorization is to ignore a set of those fill-ins obtaining a matrix $LU$ which hopefully resembles $A$. As the sparsity pattern (often the pattern of $A$) is selected during the course of the factorization, operations that would cause fill are set to zero. In PDE problems there are physical explanations to justify the use of these type of preconditioners. Understanding the underlying physical motivation has produced very effective preconditioners. The success of this approach has motivated the use of incomplete factorization on other problems where there is not much
information to guide the choice of preconditioners. For symmetric positive definite matrices we compute say \( \hat{L} \) and let \( M = \hat{N} = \hat{L} \) thus obtaining a (preconditioned) symmetric positive definite matrix which should be closer to the identity than the original matrix \( A \). This preconditioner is called the incomplete Cholesky factorization.

The incomplete Cholesky factorization as described before is not well defined in general because nonpositive pivots can occur since some of the fill-ins are ignored. There are several ways to remedy this situation. We favor the one which consists in setting the pivot to the absolute sum of the off-diagonal entries when a nonpositive pivot is found. This ensures that this particular row is diagonally dominant.

The incomplete Cholesky factorization is a natural choice to apply on the Schur complement and it was done before in [39] with some minor changes. We will consider this factorization later in a similar context. In this work, we are concerned with the incomplete Cholesky factorization version that ignores all fill-ins that may occur on \( L \). This particular incomplete Cholesky factorization can be done in place which saves space and reduces data movement for this approach.

Another preconditioner for symmetric matrices we wish to study is the SSOR which is defined as

\[
M = (L + D)D^{-1}(L^t + D) = A + LD^{-1}L^t
\]

and \( N = I \). where \( D \) is the diagonal of \( A \) and \( L \) its strict lower triangular part

\[
A = L + D + L^t.
\]

The SSOR preconditioner is usually applied to positive definite matrices since these matrices have nonzero diagonal entries. That is not true for indefinite matrices in
general. In particular, the augmented matrix for any linear programming problem with equality constraints has zero diagonal entries. Later we shall see a modification to the SSOR so that it can be applied to the augmented system.

4.4 Preconditioners for the Augmented System

Consider the following class of symmetric preconditioners*:

\[
\begin{pmatrix}
H & 0 \\
F & G
\end{pmatrix}
\begin{pmatrix}
-D & A^t \\
A & E
\end{pmatrix}
\begin{pmatrix}
H^t & F^t \\
0 & C^t
\end{pmatrix}
= \begin{pmatrix}
-HDH^t & B^t \\
B & C
\end{pmatrix}
\]

(4.3)

where \( B = -FDH^t + GAH^t \) and \( C = -FD^t + FA^t C^t + GAF^t + GEC^t \). We will see next that some of the preconditioners proposed to the augmented system belong to this class.

4.4.1 Existing Preconditioners

Gill et al. [24] introduce some preconditioners for the standard form which try to avoid the numerical problems caused by the ill-conditioned behavior of \( D \) as the primal-dual method progresses. The first preconditioner they proposed was:

\[
\hat{M} = \begin{pmatrix}
\hat{H} & 0 \\
0 & I
\end{pmatrix}
\]

where \( \hat{H} \) is a diagonal \( n \times n \) matrix whose nonzero entries are:

\[
\hat{\eta}_{ii} = \begin{cases}
\delta_i^{-\frac{1}{2}} & \text{for the } n - m \text{ biggest entries of } D \\
1 & \text{otherwise.}
\end{cases}
\]

Notice that this is equivalent to choosing \( H \equiv \hat{H}, F \equiv 0 \) and \( G \equiv I \) in (4.3).

*From now on we drop the superscript \( k \) since we are concerned with one iteration of the primal-dual interior point method.
If we adopt the partition $A = [B \, N]$ where $N$ corresponds to the $n - m$ variables closest to a bound. (we omit for simplicity the permutation of the columns of $A$) and use the notation $H_B$ and $H_N$ to denote the corresponding submatrices for a matrix $H$, their preconditioners correspond to the choices $M1$ to $M4$ given in table 4.1.

In the same work, they also propose a preconditioner based upon the Bunch-Parlett factorization $\hat{L}\hat{D}\hat{L}^t$ [7] for an approximation of the matrix in the system (2.5). By constructing a diagonal matrix $\hat{D}$ whose eigenvalues are the absolute value of those in $\hat{D}$ (recall that $\hat{D}$ is a block diagonal matrix whose blocks have dimension 1 or 2), they obtain a positive definite matrix and therefore a symmetric preconditioner.

In [3] Batterman and Heinkenschloess introduce and analyze another class of preconditioners for the quadratic programming problem. Table 4.1 also gives the choice for the block matrices for the first two preconditioners $P1$ and $P2$ presented by them. The third preconditioner introduced has a different partitioning:

$$P3 = \begin{pmatrix}
I & 0 & DBB^{-1} \\
0 & 0 & B^{-1} \\
-(B^{-1}N)^t & I & (B^{-1}N)^t DBB^{-1}
\end{pmatrix}.$$  

Notice that this is the representation for these preconditioners for linear programming problems. See [3] for a complete description.

### 4.4.2 Modified Preconditioners

The preconditioners in [24] presented on table 4.1 break down for degenerate problems, that is, problems which have an optimal solution where more than $m$ primal variables $x$ are not at their bounds. A consequence of degeneracy is that the number of diagonal entries on $D$ that will converge to zero is unknown. Since these precon-
<table>
<thead>
<tr>
<th>Preconditioner</th>
<th>$H_N$</th>
<th>$H_B$</th>
<th>$F_N$</th>
<th>$F_B$</th>
<th>$G$</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>$-D_N^{-1}$</td>
<td>$I$</td>
<td>0</td>
<td>0</td>
<td>$I$</td>
</tr>
<tr>
<td>M2</td>
<td>$-D_N^{-1}$</td>
<td>$B^{-t}$</td>
<td>0</td>
<td>0</td>
<td>$I$</td>
</tr>
<tr>
<td>M3</td>
<td>$-D_N^{-1}$</td>
<td>$B^{-t}$</td>
<td>0</td>
<td>$-\frac{1}{2}B^{-t}H_BB^{-1}$</td>
<td>$I$</td>
</tr>
<tr>
<td>M4</td>
<td>$-D_N^{-1}$</td>
<td>$L^{-1}$</td>
<td>0</td>
<td>$-\frac{1}{2}L^{-1}H_BL^{-1}$</td>
<td>$U^t$</td>
</tr>
<tr>
<td>P1</td>
<td>$-D_N^{-1}$</td>
<td>$-D_B^{-\frac{1}{2}}$</td>
<td>0</td>
<td>0</td>
<td>$-H_BB^{-1}$</td>
</tr>
<tr>
<td>P2</td>
<td>$-D_N^{-1}$</td>
<td>$-D_B^{-\frac{1}{2}}$</td>
<td>$-H_BB^{-1}ND_N$</td>
<td>$-H_B$</td>
<td>$H_B^{-1}B^{-1}$</td>
</tr>
</tbody>
</table>

**Table 4.1** Existing Preconditioners. $B^t = LU$

<table>
<thead>
<tr>
<th>IP Iteration</th>
<th>M4</th>
<th>MOD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>157</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>262</td>
<td>23</td>
</tr>
<tr>
<td>3</td>
<td>285</td>
<td>25</td>
</tr>
<tr>
<td>4</td>
<td>314</td>
<td>67</td>
</tr>
<tr>
<td>5</td>
<td>322</td>
<td>27</td>
</tr>
<tr>
<td>6</td>
<td>327</td>
<td>26</td>
</tr>
<tr>
<td>7</td>
<td>338</td>
<td>48</td>
</tr>
<tr>
<td>8</td>
<td>359</td>
<td>51</td>
</tr>
<tr>
<td>9</td>
<td>365</td>
<td>140</td>
</tr>
<tr>
<td>10</td>
<td>368</td>
<td>77</td>
</tr>
<tr>
<td>11</td>
<td>371</td>
<td>86</td>
</tr>
<tr>
<td>12</td>
<td>361</td>
<td>104</td>
</tr>
<tr>
<td>13</td>
<td>362</td>
<td>92</td>
</tr>
<tr>
<td>14</td>
<td>350</td>
<td>66</td>
</tr>
<tr>
<td>15</td>
<td>311</td>
<td>60</td>
</tr>
<tr>
<td>16</td>
<td>162</td>
<td>55</td>
</tr>
<tr>
<td>17</td>
<td>104</td>
<td>53</td>
</tr>
<tr>
<td>Total</td>
<td>5118</td>
<td>1030</td>
</tr>
</tbody>
</table>

**Table 4.2** SC205 Matrix-vector products for M4 and modified Preconditioner
ditioners assume it occurs for a fixed number \((n - m)\) their performance is sensibly affected by degenerate problems. This breakdown happens because the way \(\hat{H}\) is computed, by setting this fixed number of entries to one is too rigid. Also, they neglect the fact that at the first few iterations of the interior point methods, most entries on the diagonal of \(D\) are typically far away from zero and there is no need to set \(m\) of them apart even for non-degenerated problems. We modified the first preconditioner to address these shortcomings:

\[
\eta_{ii} = \begin{cases} 
\delta_{ii}^{-\frac{1}{2}} & \text{if } \delta_{ii} > 10^{-8} \hat{\delta} \\
(10^{-8} \hat{\delta})^{-\frac{1}{2}} & \text{otherwise}
\end{cases}
\] (4.4)

where \(\hat{\delta} = \max_i(\delta_{ii})\). The idea is to eliminate a variable number of entries of \(D\). Thus, at the first few iterations of the interior point method, the entries of diagonal \(D\) are within a relatively small range and typically almost all of them are eliminated. On the other hand, in the last iterations some entries are very large and hence eliminated, while others are very small and the preconditioner avoids working with their reciprocals.

We also introduce the incomplete Cholesky factorization of the Schur complement \(S\). Thus, the modified symmetric preconditioner we propose is given by:

\[
M = \begin{pmatrix} H & 0 \\ 0 & \hat{L}^{-1} \end{pmatrix}
\] (4.5)

where, \(H\) is defined in (4.4) and \(\hat{L}\hat{L}^t\) is an incomplete Cholesky factorization of \(S\).

These modifications achieved better computational results as we can see in table 4.2. It shows results for two different preconditioners using MINRES, the fourth precondition given by Gill et al. \(M4\), and the modification (4.5). This experiment was
carried out in the same computing environment described in Chapter 3.

The problem shown in table 4.2 is SC205 from the netlib test set. The size of the indefinite system is 440. The column labeled “Iteration” has the number of the iteration for the interior point method, which corresponds to a linear system. The columns “M4” and “MOD” in table 4.2 contain the number of matrix-vector products the iterative method needed to converge for the respective preconditioner.

4.4.3 Unsymmetric Preconditioners

Consider the inverse of the augmented system with implicit slack variables:

\[
\left( \begin{array}{cc}
-D & A^t \\
A & E
\end{array} \right)^{-1} = \left( \begin{array}{cc}
-D^{-1} + D^{-1}A^tS^{-1}AD^{-1} & D^{-1}A^tS^{-1} \\
S^{-1}AD^{-1} & S^{-1}
\end{array} \right)
\]  

(4.6)

where \( S = AD^{-1}A^t + E \).

From relation (4.6) we can derive an unsymmetric preconditioner by computing an incomplete Cholesky factorization \( \hat{L}\hat{L}^t \) of \( S \) and replacing \( S \) by \( \hat{S} = \hat{L}\hat{L}^t \). Notice that we call it unsymmetric because the preconditioned matrix may be unsymmetric although both the original matrix and the preconditioner are symmetric. This preconditioner can be very effective but its matrix-vector product is more expensive than the others since it requires the solution of two linear systems involving \( \hat{S} \). We will see that taking it as a right preconditioner for the GMRES method (to be described later), gives another option for solving the augmented system.

Freund and Jarre [21] present another unsymmetric preconditioner based upon the SSOR where the matrix \( D \) is now block diagonal with 1 \( \times \) 1 and 2 \( \times \) 2 blocks about the diagonal. In order to achieve this, it is necessary to permute the augmented
matrix since its last diagonal block consists of all zeros for problems in the standard form which is adopted in their work. Their preconditioner has the following form \( X = I \) and \( M = U^T D^{-1} U \) where, \( U = (L^T + D)P^T \) and \( P \) is a permutation for the augmented matrix.

The block diagonal matrix \( D \) in this preconditioner has \( m \) \((2 \times 2)\) blocks and \( m - n \) \((1 \times 1)\) blocks since the augmented system has \( m \) zero diagonal entries. The \( 2 \times 2 \) blocks have the following form

\[
\begin{pmatrix}
-\delta_{ij} & \alpha_{ij} \\
\alpha_{ij} & 0
\end{pmatrix}.
\]

In order to \( M \) be a good approximation for \( A \), \( \|LD^{-1}L^T\| \) should be as small as possible. With this goal in mind they propose to choose the permutation by minimizing the product of the largest eigenvalues of the blocks on \( D^{-1} \) over all possible choices of \( 2 \times 2 \) blocks. This approach gives different permutations per iteration. The problem of finding the permutation can now be seen as a weighted bipartite matching problem which they propose to solve by heuristic methods.

4.5 Why Work with the Augmented System?

The reduction to the Schur complement has the inherent disadvantage of changing the sparse structure of the matrix which can be disastrous for some problems. Even if iterative methods are being applied, the computation of the Schur complement is necessary for obtaining a preconditioner in all implementations we are aware of. The computational time required to construct and apply it may not compensate the reduction of the problem from indefinite to a smaller positive definite system. For instance, the constraint matrix of problem FIT2P from the netlib test set has about
50,000 nonzero entries while the lower half of $AA^t$ has about 4,500,000 nonzero entries.

The results of this section are very important because they give theoretical support for working with the augmented system. They show that by working only with the Schur complement, we are loosing the whole picture of the problem and may not understand it completely. On the other hand, by studying the augmented system, no information is lost. Moreover, it is possible to come back to the Schur complement system when it is advantageous to do so since it is in some sense contained in augmented system. We shall see an example where this understanding can be applied in a practical way in Chapter 5.

4.5.1 The Importance of the Augmented System

The following lemma gives a justification for working with the augmented system instead of the Schur complement when applying iterative methods. It will be shown that every preconditioner for the Schur complement can be replicated on the augmented system while the converse statement is not true. Therefore, the preconditioned Schur complement system can be seen as a particular case contained on the more general class of preconditioners for the augmented system.

Lemma 4.1 Let

$$
\begin{pmatrix}
-D & A^t \\
A & E
\end{pmatrix}
$$

be nonsingular and $D$ be symmetric positive definite. Then there exist a preconditioner $M$ and $N$ such that this matrix can be reduced to:

$$
M^{-1} \begin{pmatrix}
-D & A^t \\
A & E
\end{pmatrix} N^{-1} = \begin{pmatrix}
I & 0 \\
0 & GSH^t
\end{pmatrix}
$$
where $S$ is the Schur complement and $G$ and $H$ are arbitrarily chosen nonsingular matrices of appropriated dimension.

**Proof** Let $D = LL^t$ and consider the preconditioner

$$M^{-1} = \begin{pmatrix}
-L^{-1} & 0 \\
GAD^{-1} & G
\end{pmatrix}$$

and

$$N^{-1} = \begin{pmatrix}
L^{-t} & D^{-1}A^tH^t \\
0 & H^t
\end{pmatrix}$$

then,

$$M^{-1} \begin{pmatrix}
-D & A^t \\
A & E
\end{pmatrix} N^{-1} = \begin{pmatrix}
I & 0 \\
0 & GSH^t
\end{pmatrix} \quad (4.7)$$

where $S = AD^{-1}A^t + E$.

This relation indicates that every preconditioner for the Schur complement system has an equivalent for the augmented system. Observe that this result holds for singular matrices $G$ and $H$. However, it does not make sense to obtain a singular matrix in the context of preconditioning. Incidentally, the preconditioner used for proving the lemma is the one used for the numerical experiments in Chapter 3. In these experiments $G$ and $H$ are the incomplete Cholesky factorization of $S$.

The next lemma shows that for the augmented system (2.14), the converse is not true.

**Lemma 4.2** Consider the augmented system given by

$$\begin{pmatrix}
-D & A^t \\
A & E
\end{pmatrix} \begin{pmatrix}
\Delta x \\
\Delta y
\end{pmatrix} = \begin{pmatrix}
r_y - X^tr_z + V^tr_t \\
r_x + Y^tr_w
\end{pmatrix}$$
and its Schur complement $S = AD^{-1}A^t + E$ where $D$ is nonsingular and $A$ has full row rank. Then any symmetric block triangular preconditioner

$$
\begin{pmatrix}
H & 0 \\
F & G
\end{pmatrix}
$$

leads to a preconditioned system for $GSG^t$ independent of $H$ and $F$.

**Proof** Consider again the class of symmetric preconditioners given by:

$$
\begin{pmatrix}
H & 0 \\
F & G
\end{pmatrix}
\begin{pmatrix}
-D & A^t \\
A & E
\end{pmatrix}
\begin{pmatrix}
H^t & F^t \\
0 & G^t
\end{pmatrix}
= 
\begin{pmatrix}
-HDH^t & B^t \\
B & C
\end{pmatrix}
$$

where $B = -FDH^t + GAH^t$ and $C = -FDG^t + FA^tG^t + GAF^t + GEG^t$. Therefore, the preconditioned system will be as follows:

$$
\begin{pmatrix}
-HDH^t & B^t \\
B & C
\end{pmatrix}
\begin{pmatrix}
\Delta \hat{x} \\
\Delta \hat{y}
\end{pmatrix}
= 
\begin{pmatrix}
H(r_y - X^t r_z + V^t r_t) \\
G(r_x + Y^* r_w) + F(r_y - X^t r_z + V^t r_t)
\end{pmatrix}
$$

now, by eliminating $\Delta \hat{x}$ we get:

$$
GSG^t \Delta \hat{y} = G(r_x + Y^* r_w + AD^{-1}(r_y - X^t r_z + V^t r_t))
$$

(4.8)

this system does not depend on either $H$ or $F$. thus any choice for these matrices that preserves nonsingularity are valid preconditioners which lead to (4.8). □

Notice that these results can be applied in a more general context. For instance $D$ can be any symmetric positive definite matrix not necessarily diagonal. Systems with these characteristics occur in other fields including nonlinear programming and PDE problems. Moreover, there is no restriction to $E$ whatsoever. However, if $E$ is positive semi-definite as in the augmented system, the Schur complement will be positive definite.
In Chapter 5 we will propose a new class of preconditioners similar to (4.3) for the augmented system (2.5). Our main concern in developing this class is to avoid working with the Schur complement, the source of most computational work for the approaches we are competing with.
Chapter 5

A New Class of Preconditioners

In this chapter we will construct and study a class of symmetric preconditioners for the augmented system which exploits its structure. In view of the discussion on previous chapters the preconditioners are designed to avoid forming the Schur complement. Our goal is to obtain a preconditioner that is relatively cheap to compute that retains excellent theoretical and practical properties. After studying its properties we will discuss how to implement a competitive code for this class of preconditioners.

5.1 Building a Preconditioner

Since the augmented system is naturally partitioned into block form, let's start with the most generic possible block symmetric preconditioner for the augmented system

$$M^{-1} = N^{-1} = \begin{pmatrix} F & G \\ H & J \end{pmatrix}$$

and choose the matrix blocks step by step according to our goals. We only consider symmetric preconditioners. We believe there is insufficient motivation for loosing symmetry.

The preconditioned augmented matrix (2.5) will be the following *:

$$\begin{pmatrix} -FD + FA^tG + GA^t & -FD + FA^tJ + GA^t \\ -HF + HA^tG + JA^t & -HF + HA^tJ + JA^t \end{pmatrix} \right).$$

(5.1)

*We will be restricted for now to the problem without inequality constraints and free variables to facilitate the discussion.
At this point we begin making decisions about the blocks. We start by observing that the right lower block is critical in the sense that the Schur complement \( AD^{-1}A^t \) appears in the expression for many reasonable choices of \( J \) and \( H \). The Schur complement may be avoided more easily if we set \( J = 0 \). This choice seems to be rather drastic at first glance leaving few options for the selection of the other blocks. But, as we will soon see, the careful selection of the remaining blocks will lead to promising situations. With \( J = 0 \) the preconditioned augmented system reduces to

\[
\begin{pmatrix}
-DFD^t + FA^tG^t + GA^t & -FDH^t + GAH^t \\
-HDF^t + HA^tG^t & -HDH^t
\end{pmatrix}.
\]

Let's turn our attention to the off-diagonal blocks. If we can make them zero blocks, the preconditioned matrix appears to be closer to the identity matrix and the problem decouples into two smaller linear systems. Thus, one idea is to write \( F^t = D^{-1}A^tG^t \). However, it is easy to verify that if the null space of \( F^t \) and \( G^t \) have a nonzero vector in common, the preconditioner is singular. Since \( G^t \) is rectangular, its null space has dimension at least \( n - m \). Clearly \( \mathcal{N}(F^t) \subset \mathcal{N}(G^t) \) therefore, this choice is not acceptable. A more reasonable choice is \( G^t = (HA^t)^{-1}HDF^t \) giving

\[
\begin{pmatrix}
-DFD^t + FA^tG^t + GA^t & 0 \\
0 & -HDH^t
\end{pmatrix}.
\]

Now let's decide how to choose \( H \). The choices \( A, AD^{-1} \) or variations of it will not be considered since matrices with the nonzero pattern of the Schur complement will appear in the right lower block and also as part of \( G \). On the other hand, setting
\[ H = [I \ 0]P \] where \( P \) is a permutation matrix such that \( HA^t \) is nonsingular does not introduce a Schur complement type matrix. The right lower block reduces to a diagonal matrix \(-D_B \equiv -HDH^t\) where.

\[
PDP^t = \begin{pmatrix} D_B & 0 \\ 0 & D_N \end{pmatrix}
\]

and we achieve one of the main goals, namely avoiding the Schur complement.

Therefore, we can concentrate on \( F \) at the upper left block. The choice \( F = D^{-\frac{1}{2}} \) seems to be natural and as we shall see later leads to some interesting theoretical properties for the preconditioned matrix. Summarizing, the final preconditioned matrix takes the form

\[
M^{-1} \begin{pmatrix} -D & A^t \\ A & 0 \end{pmatrix} M^{-t} = \begin{pmatrix} -I + D^{-\frac{1}{2}}A^tG^t + GAD^{-\frac{1}{2}} & 0 \\ 0 & -D_B \end{pmatrix}
\] (5.2)

where.

\[
M^{-1} = \begin{pmatrix} D^{-\frac{1}{2}} & G \\ H & 0 \end{pmatrix}
\]

with \( G = H^tD_B^{-\frac{1}{2}}B^{-1} \). \( HP^t = [I \ 0] \) and \( AP^t = [B \ N] \). We remark that this notation for the partition of \( A \) borrowed from the literature of the simplex method [10] does not mean that the set of columns \( B \) form a particular basis for the linear programming problem. The only concern up to this point is that these columns form a nonsingular matrix.

### 5.1.1 The Matrix B

We observe that the goal of avoiding the Schur complement was achieved. The price paid is that now we have to find \( B \) and solve linear systems with this matrix. However,
the factorization \( QB = LL' \) is typically easier to compute than the Cholesky factorization. The reason is that the selection of the columns of \( B \) causes no change on the structure of the problem, in contrast to the computation of \( AA' \). In fact, it is known [23] that the sparsity pattern of \( L' \) and \( L' \) is contained in the sparsity pattern of \( R \), where \( AA' = R' R \) for any valid permutation \( Q \). In practice, the number of nonzero entries of \( R \) is typically much larger than the number of nonzero entries of \( L \) and \( L' \) added. Moreover, the \( LL' \) factorization of \( B \) does not depend on \( D \) which is the only change in the matrix of the augmented system from one iteration to another. Thus, if we decide to keep the permutation matrix \( P \) unchanged, the computation of the next preconditioned matrix is almost free. It only requires computing \( D^{-\frac{1}{2}} \) from \( D^{-1} \) that is, the computation of \( n \) square roots.

Another advantage of this preconditioner compared with other augmented system preconditioners is that it reduces the size of the linear system from \( n + m \) to \( n \) since the lower block equation is easily solved for the diagonal matrix. It still gives a larger system than the Schur complement reduction though, but as we will see later, this disadvantage will be overcome too.

### 5.1.2 First Considerations About the Preconditioned Matrix

In the next section we will study some of the theoretical properties of the upper left block preconditioned matrix (5.2)

\[
-I + D^{-\frac{1}{2}}A'G' + GAD^{-\frac{1}{2}}
\]  

(5.3)

and later show some strategies on how to choose the permutation matrix, that is, the columns of \( A \) that we select to form \( B \). But first we will make a few observations
about the preconditioned matrix.

Recall that the augmented matrix (2.5) has \( m \) positive and \( n \) negative eigenvalues. Thus, the preconditioned matrix (5.2) have the same inertia of eigenvalues. Since the lower right block matrix has \( m \) negative eigenvalues, the preconditioned matrix (5.3) has \( m \) positive and \( n - m \) negative eigenvalues. Therefore the preconditioned matrix is indefinite except for the odd case where \( m = n \).

We close this section by showing a property of the matrices in the preconditioned block (5.3) which leads to the results of the next section.

**Lemma 5.1** Let \( A = [B \; \mathcal{N}] \) with \( B \) nonsingular and \( G = H^t D_B^{\frac{1}{2}} B^{-1} \) where \( H = [I \; 0] \). Then \( G^t D^{-\frac{1}{2}} A^t = A D^{-\frac{1}{2}} G = I \).

**Proof** It is sufficient to show that \( G^t D^{-\frac{1}{2}} A^t = I \).

\[
B^{-t} D_B^{\frac{1}{2}} [I \; 0] D^{-\frac{1}{2}} [B \; \mathcal{N}]^t = \]

\[
B^{-t} D_B^{\frac{1}{2}} [D_B^{-\frac{1}{2}} 0] [B \; \mathcal{N}]^t = \]

\[
B^{-t} [I \; 0] [B \; \mathcal{N}]^t = I \]

\( \square \)

This result can be easily extended for any permutation \( A = [B \; \mathcal{N}] P \) with \( B \) nonsingular.

### 5.2 Theoretical Properties of the Preconditioned System

In this section we will study some properties of matrices of the type

\[
K = -I_n + U^t V^t + VU \tag{5.4}
\]

where \( UV = V^t U^t = I_m \)
and $U, V^t$ are $m \times n$ matrices. By lemma 5.1 it easy to verify that the preconditioned matrix 5.3 given in the previous section belongs to this class of matrices. First we show that the eigenvalues of $K$ are bounded away from zero.

**Theorem 5.1** Let $\lambda$ be an eigenvalue of $K$ given by (5.4) where $U$ and $V^t \in R^{m \times n}$ then $|\lambda| \geq 1$.

**Proof** Let $v$ be a normalized eigenvector of $K$ associated with $\lambda$ then.

\[
Kv = \lambda v
\]

\[
K^2v = \lambda^2 v
\]

\[
(I - U^tV^t - VU + U^tV^tVU + VUU^tV^t)v = \lambda^2 v
\]

\[
v + (U^tV^t - VU)(VU - U^tV^t)v = \lambda^2 v.
\]

Multiplication on the left by $v^t$ gives

\[
1 + v^t(U^tV^t - VU)(VU - U^tV^t)v = \lambda^2
\]

\[
1 + w^tw = \lambda^2
\]

where $w = (VU - U^tV^t)v$. Thus, we obtain $\lambda^2 \geq 1$. \hfill \Box

**Corollary 5.1** The preconditioned matrix (5.2) is nonsingular.

Thus, by theorem 5.1 $K$ is not only nonsingular but it has no eigenvalues in the neighborhood of zero. This is a desirable property for getting good performance by the iterative methods for solving linear systems. The following remarks are useful for showing other important results.

**Remark 5.1** The eigenvectors of any matrix $K$ are the same eigenvectors of $K + I$ with the respective eigenvalues shifted by one.
**Remark 5.2** Since \( U \) and \( V^t \in R^{m \times n} \) are such that \( UV = I_m \). \( VU \) is an oblique projection onto \( \mathcal{R}(V) \). Thus, if \( x \in \mathcal{R}(V) \), then \( VUx = x \).

**Theorem 5.2** The matrix \( K \) in (5.4) where \( U \) and \( V^t \in R^{m \times n} \) has at least one eigenvalue \( \lambda \) such that \( |\lambda| = 1 \).

**Proof** Observe that if \( K + I \) is singular, by remark 5.1 \( K \) has at least an eigenvalue \( \lambda = -1 \). Lets consider three cases:

(i) \( n > 2m \). \( K + I \) is singular, since for any square matrices \( A \) and \( B \),

\[
\text{rank}(A + B) \leq \text{rank}(A) + \text{rank}(B).
\]

(ii) \( n < 2m \). observe that \( \dim(\mathcal{R}(U^t) \cup \mathcal{R}(V)) \leq n \). Also, \( U^t \) and \( V \) have rank \( m \) since \( UV = I_m \). Thus, \( \dim(\mathcal{R}(U^t) \cap \mathcal{R}(V)) > 0 \) since \( n < 2m \). Hence, there is at least one eigenvector \( v \neq 0 \) such that \( v \in \mathcal{R}(U^t) \cap \mathcal{R}(V) \) therefore, by remark 5.2, \( Kv = (-I + U^tV^t + VU)v = -v + v + v = v \).

(iii) \( n = 2m \). from (ii) if \( \mathcal{R}(U^t) \cap \mathcal{R}(V) \neq 0 \) there is a eigenvalue \( \lambda = 1 \). Otherwise there exist an eigenvector \( v \) such that \( v_{\mathcal{R}(U^t)} = 0 \) or \( v_{\mathcal{R}(U^t)} = 0 \). Without loss of generality consider that \( v_{\mathcal{R}(V^t)} = 0 \), i.e. \( v \in \mathcal{N}(U^t) \) and by remark 5.1 there is an eigenpair \( (\theta = \lambda + 1, v) \) where

\[
VUv = \theta v.
\]

But then \( \lambda + 1 = (0 \text{ or } 1) \) since \( UV = I \) and from theorem 5.1 it must be \( \lambda = -1 \).

\( \square \)

**Corollary 5.2** The condition number \( \kappa_2(K) \) in (5.4) is given by \( \max |\lambda(K)| \).
Proof. The proof is immediate from theorems 5.1, 5.2, the definition of \( \kappa_2(K) = \frac{\sigma_{\max}}{\sigma_{\min}} \) and recalling that \( K \) is symmetric.

5.3 Practical Aspects

In this section, we discuss preliminary aspects that must be considered in an implementation that apply the preconditioned matrix (5.3). A more detailed study on an efficient implementation is made at the end of this chapter.

5.3.1 Stability of the Matrix-vector Product

The iterative methods we will see later need access to the matrix only to compute matrix-vector products. In this section, we present a more stable way to compute this product versus using the matrix directly as in \( w = Kx \). Let us consider for simplicity our matrix to be of the form (5.4). We can write any \( n \)-dimensional vector \( x \) as \( x = x_{R(V)} + x_{A(V)} \), where \( x_{R(V)} \in R(V) \) and \( x_{A(V)} \in A(V) \). Thus,

\[
Kx = -x + \mathcal{U}^tV^tx_{R(V)} + V\mathcal{U}x
\]

\[
= -x_{A(V)} + \mathcal{U}^tV^tx_{R(V)} + V\mathcal{U}x_{A(V)}
\]

by remark 5.2. Now, since \( V^t = [I \ 0]\mathcal{P} \) its null and range spaces can be easily represented in a code and all the calculations for it consist in managing some indexes properly. Observe that the first two terms do not have nonzero entries in common for any of the positions. Hence, no floating point operation is needed to add them. If we compute the product \( Kx \) without these considerations, some round-off error will be introduced for the zero sum \( -x_{R(V)} + \mathcal{U}Vx_{R(V)} \) and often this error is large enough compared with the other entries of \( x \). A welcomed side effect is that \( n \) floating point operations are saved with this strategy.
5.3.2 Recovering the Solution

The approximate solution for the original system

\[
\begin{pmatrix}
-D & A^t \\
A & E
\end{pmatrix}
\begin{pmatrix}
\Delta x \\
\Delta y
\end{pmatrix}
= 
\begin{pmatrix}
b_1 \\
b_2
\end{pmatrix}
\]

can be easily recovered from the solution for the preconditioned system \((\hat{x}, \hat{y})\) by computing

\[
\begin{pmatrix}
x \\
y
\end{pmatrix} = M^{-1}
\begin{pmatrix}
\hat{x} \\
\hat{y}
\end{pmatrix}
\]

where the error of the solution is given by

\[
\begin{pmatrix}
r_1 \\
r_2
\end{pmatrix} = 
\begin{pmatrix}
b_1 + Dx - A^t y \\
b_2 - Ax - Ey
\end{pmatrix}.
\]

Sometimes the norm of the error \(\|r\|^2 = \|r_1\|^2 + \|r_2\|^2\) is too large due to the round-off error introduced on computing the preconditioned system and recovering the solution. It can get particularly large at the final iterations of the interior point method. One way to reduce this error is to compute \(\hat{x} = D^{-1}(A^ty - b_1)\) and form a new approximate solution \((\hat{x}, y)\). The error for the new solution \(\hat{r}\) will be given by \(\hat{r}_2 = r_2 + AD^{-1}r_1\) if we assume that \(\hat{r}_1 = b_1 + D\hat{x} - A^t y\) is zero. Thus, we update the solution whenever \(\|r\|\) is above a given tolerance and \(\|\hat{r}\| < \|r\|\). This update is necessary for the successful convergence of the interior point method for some of the problems tested with this preconditioner. Notice that this strategy can be used for any symmetric preconditioner for the augmented system.
5.4 Reduction to Positive Definite Systems

The initial idea is to apply iterative methods for solving the indefinite linear system at the left upper block

$$K = -I + D^{-\frac{1}{2}}A^tGt + GAD^{-\frac{1}{2}}$$

where $G = H^tD_B^\frac{1}{2}B^{-1}$ and $HP^t = [I \ 0]$. However, it is possible to exploit the structure of the problem even further, reducing it to a smaller positive definite system. If we expand the above equation we obtain the following matrix

$$P^t \begin{pmatrix} I & D_B^\frac{1}{2}B^{-1}D_N^{-\frac{1}{2}} \\ D_N^{-\frac{1}{2}}N^tB^{-1}D_B^\frac{1}{2} & -I \end{pmatrix}P \quad (5.5)$$

where we used the relation

$$PDP^t = \begin{pmatrix} D_B & 0 \\ 0 & D_N \end{pmatrix}.$$  

Therefore the problem can be reduced to solve a positive definite linear system involving either matrix

$$I_m + D_B^\frac{1}{2}B^{-1}N_D^{-\frac{1}{2}}N^tB^{-t}D_B^\frac{1}{2} \quad (5.6)$$

or

$$I_{n-m} + D_N^{-\frac{1}{2}}N^tB^{-t}B^{-1}N_D^{-\frac{1}{2}}. \quad (5.7)$$

These matrices have dimension $m$ and $n - m$ respectively. They also have some interesting theoretical properties related to the indefinite matrix $K$.

5.4.1 Properties of the Positive Definite Matrices

Lets first define $W = D_N^{-\frac{1}{2}}N^tB^{-t}D_B^\frac{1}{2}$. Hence, the positive definite matrices can be written $I + W^tW$ and $I + WW^t$ respectively while the indefinite matrix not considering
permutations reduces to
\[
\begin{pmatrix}
I & \mathbf{W}'
\\
\mathbf{W} & -I
\end{pmatrix}.
\]

**Theorem 5.3** The matrices in (5.6) and (5.7) are positive definite and their eigenvalues are greater or equal to one.

**Proof** Let \( \mathbf{v} \) be a normalized eigenvector of \( I + \mathbf{W} \mathbf{W}' \) and \( \theta \) its associated eigenvalue. then
\[
(I + \mathbf{W} \mathbf{W}') \mathbf{v} = \theta \mathbf{v}
\]
\[
\mathbf{v}'(\mathbf{v} + \mathbf{W} \mathbf{W}' \mathbf{v}) = \theta \mathbf{v}' \mathbf{v}
\]
\[
1 + \mathbf{u}' \mathbf{u} = \theta
\]
where \( \mathbf{u} = \mathbf{W}' \mathbf{v} \). Thus \( \theta \geq 1 \). The proof for \( I + \mathbf{W}' \mathbf{W} \) is similar. \( \square \)

**Remark 5.3** The matrices in (5.6) and (5.7) have the same set of eigenvalues with the exception of the extra eigenvalue equal to one for the matrix of higher dimension.

From the above theorem and theorem 10.2.1 in [26] we can conclude that the conjugate gradient method converges in at most \( \min(m, n - m) \) iterations in exact arithmetic for both positive definite matrices. Thus, we do not expect very different behavior between the two linear systems although they have different dimension. Besides the work for computing the matrix-vector product is about the same for all three systems.

The next theorem is also important since it relates the eigenpairs of the indefinite matrix with the eigenpairs of the positive definite matrices.
Theorem 5.4 Consider the eigenvalue problem
\[
\begin{pmatrix}
I & W^t \\
W & -I
\end{pmatrix}
\begin{pmatrix}
u \\
v
\end{pmatrix}
= \lambda
\begin{pmatrix}
u \\
v
\end{pmatrix}
\]
then \((\theta, u)\) is an eigenpair of \(I + W^tW\) and \((\theta, v)\) is an eigenpair of \(I + WW^t\), where \(\theta = \lambda^2\).

Proof Notice that
\[
\begin{pmatrix}
I & W^t \\
W & -I
\end{pmatrix}
\begin{pmatrix}
I & W^t \\
W & -I
\end{pmatrix}
= \begin{pmatrix}
I + W^tW & 0 \\
0 & I + WW^t
\end{pmatrix}
\]
thus.
\[
\begin{pmatrix}
I + W^tW & 0 \\
0 & I + WW^t
\end{pmatrix}
\begin{pmatrix}
u \\
v
\end{pmatrix}
= \lambda^2
\begin{pmatrix}
u \\
v
\end{pmatrix}
\]
Therefore the indefinite system can still be an option for solving the linear system since it has a better eigenvalue spectrum distribution than the positive definite systems. Incidentally, this theorem gives an alternative proof that the smallest eigenvalue for the positive definite matrices (5.6) and (5.7) is at least equal to one.

5.4.2 Equivalence to the Schur Complement

An useful fact is that the matrix (5.6) can be obtained via the Schur complement. Recall that the Schur complement is defined as \(S = AD^{-1}A^t\) and that \(A = [BN]^P\) thus.
\[
S = BD_B^{-1}B^t + ND_N^{-1}N^t
\]
now, multiplying it by \(D_B^{\frac{1}{2}}B^{-1}\) and post-multiplying by its transpose leads to
\[
D_B^{\frac{1}{2}}B^{-1}SB^{-t}D_B^{\frac{1}{2}} = I + D_B^{\frac{1}{2}}B^{-1}ND_N^{-1}N^tB^{-t}D_B^{\frac{1}{2}}
\]
(5.8)
whose matrix on the right hand side is equal to (5.6). It can be shown that the right hand side vector for both preconditioned systems is the same. Therefore the systems are completely equivalent. On the other hand, matrices (5.7) and (5.5) can not be obtained from the Schur complement. Therefore, it is necessary to take into consideration the augmented system to know about the existence of these matrices.

On the practical point of view, we favor the use of the matrix-vector product on the left hand side of (5.8) even though it is more expensive, because it is more stable. Moreover, this representation of the preconditioner fits very well with the preconditioned conjugate gradient method as we will see on the next chapter.

5.5 Choosing the Set of Columns

In this section we discuss how to select the columns of $A$ that form $B$. The type of matrices in (5.6) and (5.7) suggest a choice for the columns of $B$ by looking at the values of $D$. If we can chose the columns related to the smallest values of $diag(D)$, both $W'W'$ and $W'W$ approach zero at the final iterations of the interior point method. Thus, a good strategy consists in taking the first $m$ linearly independent columns of $A$ ordered by the value of $\delta_{ii}$ in non decreasing order. This choice of columns have the peculiar characteristic of getting better conditioned matrices for a certain set of columns as the interior point method approaches a solution, since at least $n - m$ diagonal entries entries of $D$ became large, diminishing the importance from $W'W'$ and $W'W$.

During the first few iterations, the diagonal values of $D$ are roughly the same. In fact they are all equal to one for the computation of the starting point. Hence, for these linear systems the strategy described before is ineffective. Another way to
obtain a good set of columns is to minimize $\|W\|$. This is a problem that is very hard to solve but it can be approached by a cheap heuristic, that is we choose the first $m$ linearly independent columns of $D^{-1}A$ with smallest 1-norm diminishing the importance of $WW^T$ and $W^TW$ as before. This approach is adopted for the computation of the starting point. After that the ordering of columns by the diagonal values of $D$ is used.

Contrary to some of the preconditioners given in [24], degenerated problems do not seem to play an important role in this preconditioner. This is a nice property since the selection of $m$ columns of $A$ naturally leads one to think in terms of basis for the linear programming problem.

### 5.5.1 Preliminary Experiments

Before discussing the implementation of the preconditioner in more detail, we present some preliminary experiments to show what is the typical behavior of the $LU^T$ preconditioner compared to the incomplete Cholesky approach.

Table 5.1 shows the results for the conjugate gradient method with respect to the number of iterations for the incomplete Cholesky factorization and the new preconditioner as the interior point method progresses. All experiments in this chapter are carried out in C. on a SUN ULTRA-SPARC station. The floating point arithmetic is IEEE standard double precision. The predictor-corrector variant of the primal-dual interior point method is used as default. The parameters for the interior point method are described in Chapter 3.

The chosen problem is KEN13 [9], a multi-commodity network flow problem. It can be also obtained from netlib. The dimension of the linear system is 14627 af-
<table>
<thead>
<tr>
<th>IP Iteration</th>
<th>Inner iterations</th>
<th></th>
<th>New LC Preconditioner</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Incomplete Cholesky</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>49</td>
<td></td>
<td>195</td>
</tr>
<tr>
<td>1</td>
<td>49</td>
<td></td>
<td>203</td>
</tr>
<tr>
<td>2</td>
<td>45</td>
<td></td>
<td>258</td>
</tr>
<tr>
<td>3</td>
<td>39</td>
<td></td>
<td>190</td>
</tr>
<tr>
<td>4</td>
<td>24</td>
<td></td>
<td>171</td>
</tr>
<tr>
<td>5</td>
<td>24</td>
<td></td>
<td>185</td>
</tr>
<tr>
<td>6</td>
<td>20</td>
<td></td>
<td>128</td>
</tr>
<tr>
<td>7</td>
<td>22</td>
<td></td>
<td>130</td>
</tr>
<tr>
<td>8</td>
<td>22</td>
<td></td>
<td>133</td>
</tr>
<tr>
<td>9</td>
<td>32</td>
<td></td>
<td>126</td>
</tr>
<tr>
<td>10</td>
<td>44</td>
<td></td>
<td>108</td>
</tr>
<tr>
<td>11</td>
<td>71</td>
<td></td>
<td>91</td>
</tr>
<tr>
<td>12</td>
<td>104</td>
<td></td>
<td>92</td>
</tr>
<tr>
<td>13</td>
<td>171</td>
<td></td>
<td>76</td>
</tr>
<tr>
<td>14</td>
<td>323</td>
<td></td>
<td>63</td>
</tr>
<tr>
<td>15</td>
<td>480</td>
<td></td>
<td>52</td>
</tr>
<tr>
<td>16</td>
<td>834</td>
<td></td>
<td>43</td>
</tr>
<tr>
<td>17</td>
<td>1433</td>
<td></td>
<td>34</td>
</tr>
<tr>
<td>18</td>
<td>2146</td>
<td></td>
<td>30</td>
</tr>
<tr>
<td>19</td>
<td>4070</td>
<td></td>
<td>22</td>
</tr>
<tr>
<td>20</td>
<td>7274</td>
<td></td>
<td>18</td>
</tr>
<tr>
<td>21</td>
<td>11739</td>
<td></td>
<td>17</td>
</tr>
<tr>
<td>22</td>
<td>15658</td>
<td></td>
<td>15</td>
</tr>
<tr>
<td>23</td>
<td>24102</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>24</td>
<td>13463</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>25</td>
<td>5126</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>Average</td>
<td>3360</td>
<td></td>
<td>84</td>
</tr>
</tbody>
</table>

Table 5.1  KEN13 Conjugate Gradient Method Iterations
ter preprocessing. The iteration number zero corresponds to the linear system for computing the starting point for the interior point method. Only the number of iterations of the conjugate gradient method for solving the first linear system of the interior point iterations are shown. The number of iterations for solving the second linear system is very close to the number for the first system. An interesting property of these approaches is that the incomplete Cholesky preconditioners in general starts taking few iterations to get convergence and its behavior deteriorates as the interior point method converges to a solution. With the $LU'$ based preconditioner the exact opposite occurs. The last few iterations are the ones where it performs better. This property of the $LU'$ preconditioner is highly desirable since the last linear systems are the most ill-conditioned.

The results obtained for problem KEN13 can be considered as typical for both preconditioners. That leads naturally to the idea of a hybrid approach. We start with the incomplete Cholesky preconditioner and change to the $LU'$ preconditioner at a certain point. A crucial consists in finding a suitable way to decide when to switch the preconditioner from the incomplete Cholesky factorization to the new $LU'$ factorization approach. The number of of iterations for the conjugate gradient method to achieve convergence would be a good indicator to determine when is advisable to switch. It could be when it takes more iterations than a given parameter or when a sudden increase on this number occur.

The hybrid approach is a promising idea and it is a subject for future research. However, in this work we are mostly concerned with a successful implementation of the new preconditioner.
5.5.2 Scaling the Columns

Looking at the expression $W = D_N^{-\frac{1}{2}}N^tB^{-t}D_B^{\frac{1}{2}}$ again, it is tempting to scale the matrix after selecting the columns of $B$ such that $\|W\| \approx 0$. The following lemma shows that this idea is not easy to implement since the scaling will disappear on the preconditioned matrix.

**Lemma 5.2** Consider the following scaling of the augmented system

$$
\begin{pmatrix}
C & 0 \\
0 & I
\end{pmatrix}
\begin{pmatrix}
-D & \hat{A}^t \\
\hat{A} & 0
\end{pmatrix}
\begin{pmatrix}
C & 0 \\
0 & I
\end{pmatrix}
\begin{pmatrix}
C^{-1} & 0 \\
0 & I
\end{pmatrix}
\begin{pmatrix}
\Delta x \\
\Delta y
\end{pmatrix}
= 
\begin{pmatrix}
C & 0 \\
0 & I
\end{pmatrix}
\begin{pmatrix}
b_1 \\
b_2
\end{pmatrix}.
$$

Then the preconditioned matrix (5.5) is independent of the scaling matrix $C$.

**Proof** The scaled matrix is given by

$$
\begin{pmatrix}
-CDC & \hat{A}^t \\
\hat{A} & 0
\end{pmatrix}
$$

where $\hat{A} = AC$. Its preconditioned form is as follows

$$
P^t
\begin{pmatrix}
I & D_B^{\frac{1}{2}}B^{-1}ND_N^{-\frac{1}{2}} \\
D_N^{-\frac{1}{2}}N^tB^{-t}D_B^{\frac{1}{2}} & -I
\end{pmatrix}P
$$

where $AP^t = [B,Y]$

It is still possible to use the idea of scaling with good results. After computing the first $LU^*$ factorization, the columns not on $B$ can be rescaled giving a new linear programming problem. With a proper choice of the scaling factor, the first linear systems will be better conditioned than before and yet the problem will not be badly scaled. Before a second $LU^*$ factorization is computed, the original linear programming problem is recovered by undoing the scaling. Therefore, the linear programming
problem will be properly scaled for the remaining of the procedure and the first iterations of the interior point method generate better conditioned systems which are the most difficult for this class of preconditioners.

5.6 Implementation Issues

This class of preconditioners is not competitive against the direct method approach by computing the Cholesky factorization without a careful implementation. This is due to the computation of an $LU'$ factorization where the set of independent columns is unknown at the start of the factorization. This factorization may be too expensive for two reasons. First, it may generate too many fill-in entries. Second, it may be necessary to factor too many columns before the completion of the factorization since the dependent columns must be discarded.

In this and next sections we discuss several techniques for the implementation of a competitive code. Most of the techniques presented here concern the computation of the $LU'$ factorization and are used on the numerical experiments presented on Chapter 7.

5.6.1 The $LU'$ Factorization

For this application, the most economical way to compute the $LU'$ factorization is to work with one column at a time. This version of the $LU'$ factorization is sometimes called the delayed update form. It fits very well with our problem because when a linearly dependent column appears, it is eliminated from the factorization and the method proceeds with the next candidate column. No time is wasted in updating the matrix using columns which may turn out to be dependent. This procedure is repeated until $m$ linearly independent columns are found. If $A$ has less than $m$ lin-
early independent columns, the factorization is applied to the right hand side of the linear programming problem. If it is linearly independent to the set of columns, the problem is infeasible. otherwise the remaining rows of $A$ are redundant and can be eliminated from the problem.

Observe that this situation can happen only once and it corresponds to the preprocessing rule for eliminating redundant rows described in chapter 3. For problems with inequality constraints considered implicitly, it may be necessary to introduce some of the slack variables in order to compute the $LU$ factorization since it is not possible simply to eliminate a row as redundant in this case.

### 5.6.2 Keeping the Set of Columns

A nice property of this preconditioner is that we can work with the selected set of columns for some iterations. Actually, for some small problems the same matrix $B$ can be used for all iterations of the interior point method. As a consequence, the preconditioner is very cheap to compute for those iterations that use the previous set of columns. Observe that most of the work for computing the preconditioner consists of choosing the columns and computing the $LU^T$ factorization which is already available.

It is important to notice that keeping the matrix $B$ from previous iterations does not mean to keep the same preconditioner since $D$ will change from iteration to iteration and the preconditioner depends on it too. Thus, this strategy may give different preconditioners at each iteration that are very inexpensive to compute. However, these preconditioners do not have the best selection of columns after the first factorization according to the heuristic.
<table>
<thead>
<tr>
<th>Version</th>
<th>Factorizations</th>
<th>M-Flops</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard</td>
<td>25</td>
<td>117</td>
<td>2370</td>
</tr>
<tr>
<td>Keep LU</td>
<td>4</td>
<td>126</td>
<td>2162</td>
</tr>
</tbody>
</table>

**Table 5.2** KEX13 New Factorization versus Keeping LU

Table 5.2 illustrates this idea. In column version we have the standard approach of computing a factorization at every iteration against the idea for keeping the factorization. A new $LU'$ factorization is computed whenever the preconditioned conjugate gradient method takes more iterations to achieve convergence than one over twenty times the dimension of the linear system. Notice that this version computes only 4 factorizations and it is faster than the standard approach. However, it takes more floating point operations on average for solving a linear system. The explanation is that there is a high overhead on computing the $LU'$ factorization. In contrast, the solution of the triangular systems is straightforward with practically no overhead. This result cannot be considered typical as far our experience with this preconditioner goes. It can be expected more factorizations per number of iterations in average than the obtained for this problem.

We can envision several strategies for deciding when to keep the set of columns. In the experiments given we change the set whenever the iterative method takes more iterations than a certain threshold value or when the solution given by the iterative method is not accurate.

### 5.6.3 Incomplete $LU'$ Factorization

It was observed in practice that the $LU'$ factorization often generates too many fill-in entries. The reason for this is that no reordering procedure for reducing the number
of fill-ins can be used since the columns of the matrix are not known until they are accepted. Later in this chapter we shall present techniques developed to lessen this problem.

Here we discuss another possibility which is not adopted as a default option. It consists of computing an incomplete $LU$ factorization. The standard incomplete factorization where the nonzero structure of the original matrix coincides which the nonzero structure of the triangular matrices $L \backslash U$ does not work well for this problem. In fact, our implementation of the interior point method does not converge for any of the problems tested using this incomplete factorization. On the other hand, the use of drop tolerance seems to be a viable approach. The idea is to eliminate any entry smaller than a preset value. For a carefully chosen tolerance, this technique can be very useful and it actually gives better performance on some of the problems tested. This line of research deserves more investigation on future works.

5.6.4 Using Indicators

Another approach that can be exploited for reducing work on the factorizations when the interior point method is close to the solution is the use of indicators [17]. An indicator is a tool for determining if a column is not part of any optimal basis before the method converges. Such columns can thus be eliminated from the problem. In the context of this work, indicators can be used to keep these columns at the end of the list for finding a independent set of columns, saving work on the factorization. Since these columns are not being eliminated from the problem, it is possible to be less rigorous on the way they are determined without taking the risk of getting a wrong solution for the linear programming problem. Observe that the diagonal entries of $D$ are also valid indicators. Thus, the approach adopted as standard actually uses
indicators for reordering the columns although on a different manner compared to the one described here.

5.6.5 Avoiding Dependent Columns

A more sophisticated implementation should record set of columns that make another column dependent. Therefore this information can be used to save computational work on the following factorizations by not considering these columns whenever they appear behind this set on the new ordering.

In order to have an efficient search for these sets and to avoid excessive use of memory, this type of information can be stored at the bit level. Thus, if \( A \) has \( n \) columns, a set can be stored on \( n \) bits and operating with these bits will be much faster than managing arrays of indexes. Moreover, memory restriction can be a critical issue if arrays are used for storing these sets for large scale problems.

5.6.6 Computing a Second \( LU' \) Factorization

This technique is also related to the excessive number of fill-in entries on the \( LU' \) factorization. The idea of the second factorization on the chosen set of independent columns is to apply any desirable technique for computing an efficient sparse \( LU' \) factorization, such as reordering the columns, finding strongly connected components, choosing the pivots with sparsity in mind, etc.

This approach improved the results significantly for some problems because the reduction of the floating point operations on the iterative linear system solver compensates the extra work for computing the factorization. It also benefits better form
<table>
<thead>
<tr>
<th>IP Iteration</th>
<th>Nonzero Entries</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>First LC</td>
<td>Second LC</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>21713</td>
<td>15202</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>27367</td>
<td>15797</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>28159</td>
<td>19195</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>37436</td>
<td>17933</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>41139</td>
<td>18051</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>37369</td>
<td>18140</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>37060</td>
<td>17540</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>41858</td>
<td>17616</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>39400</td>
<td>20661</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>40832</td>
<td>17191</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>43442</td>
<td>21716</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>40826</td>
<td>23510</td>
<td></td>
</tr>
</tbody>
</table>

| Time(s)      | 185.8           | 168.2     |  |
| M-Flops      | 106.8           | 86.1      |  |

**Table 5.3** TRUSS Number of nonzero entries

the predictor-corrector variant because the preconditioner is used for solving two linear systems.

Table 5.3 illustrates this savings for problem TRUSS. It is a problem for minimizing the weight of structures subject to external loads. For further details see [51]. This problem is part of the netlib test collection of linear programming problems. The dimension of the linear system is 1000.

Only the iterations where the second $LL^T$ factorization is computed are shown. The second factorization is computed whenever the number of nonzero entries of $L$ plus $L^T$ is more than four times the dimension of the linear system. The first column shows the number of nonzero entries for the first $LL^T$ factorization. The second column has the number of nonzero entries after reordering the chosen columns. At the
bottom of the table we have the time in seconds and the average number of floating point operations for solving the linear systems for both approaches. The second approach saves work because the time for solving the linear systems with a more sparse preconditioner compensates the time for computing the second factorization.

The following techniques are the default options for the second $LU^\ast$ factorization on our code. We stress that it is not possible to use them on the first $LU^\ast$ factorization because the structure of $B$ is not known prior to the factorization.

The columns are permuted by the ordering of $B^tB$ given by the minimum degree ordering. We included a threshold parameter for the choice of the pivot. At each step of the factorization, we chose a row permutation with the pivot being chosen among all candidates within the threshold. The one with least entries on its row for the remaining columns of the original matrix is chosen. For future implementations, we plan to find strongly connected components and to choose the pivot by considering the updated matrix instead the original matrix for minimizing the amount of fill-in entries.

5.7 Identifying Symbolically Dependent Columns

In the last section we presented an idea for saving work on computing the $LU^\ast$ factorization by recording the ordering of the columns from previous factorizations. In the following sections we will present techniques which save computational work by studying the ordering of columns prior to computing the factorization.

Given an ordering of columns, we want to find the unique set of $m$ independent columns which preserves the ordering. The brute force approach for this problem con-
sists in computing the factorization column by column and discarding the dependent columns on the way. The strategies developed here will change the initial ordering and indicate when a column can be ignored on the factorization. It is important to notice that the set of independent columns found by these techniques is the same set obtained by the brute force approach.

5.7.1 Previous Works

A powerful tool for reducing the work of computing the $LU$ factorization is the identification of columns that are symbolically dependent. That is, columns that are linearly dependent in structure for all numerical values of their nonzero entries. The idea is to find a set of say $k$ columns with nonzero entries in at most $k - 1$ rows. This set of columns is symbolically dependent. As we shall see, there are efficient algorithms for finding such sets.

Let us first consider square matrix for simplicity. In this situation, the problem is equivalent to permuting nonzero entries onto the diagonal. For any diagonal which a nonzero entry could not be assigned we have a symbolically dependent column. This problem is equivalent to find a matching of a bipartite graph where the rows and columns form the set of vertices and the edges are represented by the nonzero entries. If there is not a perfect matching, where all diagonal positions have a nonzero entry, then the matrix is symbolically singular. This idea was first used by Duff [15] and it is applied as a first step for permuting a matrix to block triangular form. In [15] several matching algorithms are compared. A Fortran code for the best of them is given. The implementation used in the numerical experiments presented later is based on this code.
In [12] this idea is extended to rectangular matrices. They were concerned in finding a sparse basis for the null space of the matrix. In order to obtain a sparse basis, they need to find a set of independent columns of the matrix which gives a sparse $LU$ factorization. Thus, the columns are reordered by degree and the matching algorithm applied. As a result, it will give a set of columns, denominated here *key columns*, that are not symbolically dependent. Next, an $LU$ factorization of the key columns is computed. If the set is found to be numerically dependent, the dependent columns are eliminated from the matrix. Then a new matching is computed and the process is repeated.

It is reported in [12] that for the tested problems few dependent columns are found at the first factorization and that the factorization itself is very cheap since the columns are reordered by smallest degree. This idea cannot be applied to our problem because the $LU$ factorization is expected to be much more expensive since the ordering of columns does not take into account the degree. Thus, recomputing a factorization for another set of key columns is out of question. It is worth noticing that the problems we intend to solve are much larger than the size of the reported problems.

5.7.2 Using Key Columns

In this work we have another use for the key columns. Our idea comes from the fact that the number of independent columns before the $k$th key column on the matrix is at most $k - 1$. Therefore, it is possible to speed up the $LU$ factorization whenever we find $k - 1$ numerically independent columns located before the $k$th key column. This is done by just skipping all the columns from the current one to the $k$th key column. This can be repeated for all the key columns of the matrix. Notice that we
cannot just ignore the non-key columns from the beginning because key columns may be numerically dependent among them. This could be done in [12] because loosing a few sparse columns does not affect the amount of sparsity of the resulting matrix too much. Moreover, since the key columns are reordered for sparsity, it is more likely that columns with largest degree be placed at the end of the ordering. Therefore, if the set is dependent the discarded columns are probably those with larger degree.

This idea has application on other problems, like finding a basis for a linear programming problem after running an interior point method, finding the null space of a matrix like in [12] for the null-space methods for nonlinear programming and on the reduced Hessian method for nonlinear programming.

5.7.3 Matching During the Factorization

Sometimes the use of key columns does not save too much work. The reason is that often these columns are numerically dependent. One way to save floating point operations is to compute the matching during the factorization. Thus, before we update the column with the factorization, we verify whether it is symbolically dependent or not. If it is, the column is discarded and the factorization continues with the next column.

This technique can save computational work because the matching can be done on the original matrix instead of the factored matrix. Thus, the fill-in entries are not accessed. Moreover, no floating point operation is performed. Therefore, if the column is symbolically dependent, much work is saved. If that happens for many columns, the overhead caused for the ones that are not symbolically dependent is compensated and the overall time for computing the factorization is reduced.
5.7.4 Numerical Experiments Using Symbolically Dependent Columns

Tables 5.4 and 5.5 show some results for the techniques presented in this section. For these experiments we again use problem TRUSS. The linear system has dimension 1000 and the problem has 8806 columns. Both tables show the number of nonzero entries for the first $LU^T$ factorization for a few iterations of the interior point method. A factorization is computed at every iteration of the interior point method. Table 5.4 shows the number of columns that are numerically updated for the standard approach, where no symbolic technique is used. It shows the number of columns numerically updated and skipped for the key columns approach and for the matching approach which computes the matching inside the factorization. Table 5.5 has the floating point operation count for computing the factorization for the three approaches.

It is clear that the key columns and the matching approaches can lead to great savings on the total number of floating point operations on the factorization. Later we will present experiments showing that the overhead caused by these techniques is compensated by the savings on floating point operations for most test problems.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Standard</th>
<th>Key Columns</th>
<th>Matching</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iteration</td>
<td>Nonzero</td>
<td>Updated</td>
<td>Updated</td>
</tr>
<tr>
<td>0</td>
<td>3474</td>
<td>6576</td>
<td>6572</td>
</tr>
<tr>
<td>6</td>
<td>6902</td>
<td>4478</td>
<td>3576</td>
</tr>
<tr>
<td>12</td>
<td>37369</td>
<td>3179</td>
<td>1819</td>
</tr>
<tr>
<td>18</td>
<td>40826</td>
<td>3219</td>
<td>1614</td>
</tr>
</tbody>
</table>

Table 5.4 TRUSS Updated and Skipped Columns
<table>
<thead>
<tr>
<th>Iteration</th>
<th>Nonzero</th>
<th>Standard</th>
<th>Key Columns</th>
<th>Matching</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3474</td>
<td>97.4</td>
<td>97.4</td>
<td>78.7</td>
</tr>
<tr>
<td>6</td>
<td>6902</td>
<td>207.9</td>
<td>148.9</td>
<td>28.5</td>
</tr>
<tr>
<td>12</td>
<td>37369</td>
<td>7143</td>
<td>2924</td>
<td>1281</td>
</tr>
<tr>
<td>18</td>
<td>10826</td>
<td>6782</td>
<td>2263</td>
<td>1239</td>
</tr>
</tbody>
</table>

Table 5.5 TRUSS Flops Count $\times 10^3$

5.7.5 The Preprocessing Step

Given an $m \times n$ matrix with $m \leq n$, if it has less than $m$ key columns then it is not full row rank. Thus, the symbolically dependent rows can be discarded from the linear programming problem. Therefore it is theoretically possible to identify some redundant rows in a linear programming problem without performing any floating point operation. These problems can be considered badly modeled and we did not find any instance in practice where it occurs.

5.8 Identifying Symbolically Independent Columns

Just like there exist symbolically dependent columns, it is possible to define columns that are *symbolically independent* i.e., columns that are linearly independent in structure for all numerical values of their nonzero entries. This concept can be also useful for computing the $LL^T$ factorization of a rectangular matrix.

A strategy that can be combined with all the techniques given in this chapter is to move the symbolically independent columns to the beginning of the ordered list since those columns are necessarily going to be on the factorization for this ordering. Then these columns can be reordered further by any known strategy like the number of nonzero entries (degree) in order to reduce the number of fill-ins on the $LL^T$ factor-
ization.

Notice that the symbolically dependent columns can be ignored in this step. Thus, we are concerned only with the key columns given by the matching algorithm. These are the columns candidate to be symbolically independent.

In [11], it is established that a set columns is symbolically independent if and only if it can be permuted into a matrix which contains an upper triangular submatrix with nonzero on the diagonal. We are not aware of any efficient algorithm for finding all the symbolically independent columns from a given ordered set. Therefore, we use a heuristic approach to identify some of the symbolically dependent columns.

On the description of the heuristic below, we say that column $j$ is the first entry column of row $i$ if $j$ contains the first nonzero entry in row $i$ on the ordered set. We consider a column $j$ symbolically independent given an ordered set if at least one of the following rules applies:

1. Column $j$ is the first entry column of at least one row:

2. Column $j$ is the second entry column of a row $i$ and the first entry column of row $i$ is also first entry column for at least another row not present on column $j$.

This set of rules guarantees that the columns selected are symbolically independent but it does not guarantee that all symbolically independent columns are found. For instance, consider the following sparse matrix

$$
\begin{pmatrix}
\times & \times & \times \\
\times & \times \\
\times & 
\end{pmatrix}
$$
applying the heuristic. we determine that the first column is symbolically independent by rule one and the second by rule two. However, we fail to notice that the third column is also symbolically independent.

5.8.1 Key Columns and Independent Columns

Another use for the key columns is to guess how is the sparse structure of \( B \) the matrix to be factored. This information can be used to reduce the number of fill-in entries on the factorization. This idea works fine for some problems but it deteriorates the performance of the preconditioner on others too much to be used as the default approach. One criteria to decide on using this approach is the number of symbolically independent columns found. If this number is close to the total number of columns, the key columns give a better approximation of the sparse pattern of \( B \) since most of the columns in the factorization are known.

5.8.2 Merging Symbolically Independent and Dependent Columns

After reordering the symbolically independent columns, it may possible to reduce still further the number of fill-ins in the factorization by merging the symbolically independent and dependent list of columns using the degree as criteria. This is permitted whenever the symbolically independent columns remain so.

It is very expensive to verify whether the columns remain symbolically independent at every step of the merging process. Therefore, we use the first ordering of the columns as a cheap heuristic. Thus, we move up on the list a symbolically dependent column with lower degree provided it remains behind the symbolically independent columns with lower index on the first ordering. This idea can be implemented very
efficiently.

By placing columns with lower degree into the front we hope to reduce the number of fill-ins in the factorization. However, since the symbolically dependent columns are less likely to be in the factorization, this approach is not as effective as other strategies presented in this section.

5.8.3 Numerical Experiments Using Symbolically Independent Columns

Tables 5.6 and 5.7 show some results for the techniques presented in this section. For these experiments we use problem KEN13. The linear system has dimension 14458 and the problem has 28654 columns after preprocessing. Table 5.6 shows the number of nonzero entries for the $LU$ factorization. Only iterations of the interior point method where the factorization is computed are shown. The number of symbolically independent columns found is shown under Columns. Three different versions of the method are compared. Here the standard approach, computes the matching during the factorization but does not find symbolically independent columns. The second technique finds symbolically independent columns by the heuristic given early, move them to the front and reorder them by smallest degree. The third technique merge the symbolically independent list of columns with the other list as described before. Table 5.7 has the floating point operation count for the three versions.

It can be seen that finding and reordering symbolically independent can save computational work on the factorization by reducing the number of fill-in entries. The merging version also helps saving floating point operations. Notice that the first factorization (iteration 0) is very expensive in terms of floating point operations because this problem has redundant rows. These rows are detected only after all columns are
<table>
<thead>
<tr>
<th>Iteration</th>
<th>Columns</th>
<th>Standard</th>
<th>Independent</th>
<th>Merging</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>12761</td>
<td>48406</td>
<td>49407</td>
<td>49467</td>
</tr>
<tr>
<td>11</td>
<td>12477</td>
<td>56261</td>
<td>53564</td>
<td>53190</td>
</tr>
<tr>
<td>17</td>
<td>12352</td>
<td>73077</td>
<td>59769</td>
<td>59679</td>
</tr>
<tr>
<td>23</td>
<td>12339</td>
<td>63650</td>
<td>55278</td>
<td>55287</td>
</tr>
</tbody>
</table>

Table 5.6  KEN13 Number of Nonzero Entries on the Factorization

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Columns</th>
<th>Standard</th>
<th>Independent</th>
<th>Merging</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>12761</td>
<td>203</td>
<td>230</td>
<td>230</td>
</tr>
<tr>
<td>11</td>
<td>12477</td>
<td>36.2</td>
<td>83.3</td>
<td>81.9</td>
</tr>
<tr>
<td>17</td>
<td>12352</td>
<td>398</td>
<td>397</td>
<td>391</td>
</tr>
<tr>
<td>23</td>
<td>12339</td>
<td>170</td>
<td>130</td>
<td>130</td>
</tr>
</tbody>
</table>

Table 5.7  KEN13 Flops Count $\times 10^3$ for the Factorization

computed on the first factorization.

Table 5.8 present the results for problems TRUSS and KEN13 for total time in seconds and for the average number of floating point operations for solving the linear systems on the predictor-corrector variant. Each column of the table adds one of the techniques to the previous techniques. Thus, the standard column shows the results where none of the technique using symbolic columns is used. Column key adds the

<table>
<thead>
<tr>
<th>Problem</th>
<th>Technique</th>
<th>Standard</th>
<th>Key</th>
<th>Matching</th>
<th>Independent</th>
<th>Merge</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUSS</td>
<td>M-Flops</td>
<td>86.3</td>
<td>78.4</td>
<td>76.9</td>
<td>85.6</td>
<td>85.5</td>
</tr>
<tr>
<td></td>
<td>Time (s)</td>
<td>172</td>
<td>164</td>
<td>171</td>
<td>189</td>
<td>190</td>
</tr>
<tr>
<td>KEN13</td>
<td>M-Flops</td>
<td>412</td>
<td>407</td>
<td>406</td>
<td>366</td>
<td>367</td>
</tr>
<tr>
<td></td>
<td>Time (s)</td>
<td>2432</td>
<td>2487</td>
<td>2591</td>
<td>2329</td>
<td>2343</td>
</tr>
</tbody>
</table>

Table 5.8  Time and Flops Count for All Techniques
key columns technique. Matching computes the matching during the factorization. Independent moves symbolic independent columns to the front and reorder them. Finally, the independent and dependent list are merged.

It is clear from this table that saving floating point operations does not imply on saving computational time. We believe that for problems of larger dimension these techniques became more important because the number of fill-in entries tend to grow faster with the increase of the dimension. Notice that the reordering of symbolically independent columns does not necessarily reduces the number of floating point operations. It is hard to tell a priori if this technique will work well for given problem. On Chapter 7 we shall present numerical results for large-scale problems using this and the other techniques developed in this chapter.

5.9 Special Structures

If we apply the new preconditioner to problems with inequality constraints, we obtain the following preconditioned matrix

$$M^{-1} \begin{pmatrix} -D & A^t \\ A & E \end{pmatrix} M^{-t} = \begin{pmatrix} -I + D^{-\frac{1}{2}} A^t G^t + G A D^{-\frac{1}{2}} + G E G^t & 0 \\ 0 & -D_B \end{pmatrix}$$

the only difference to the previous case being the term $G E G^t$ on $K$. Expanding $K$ one gets:

$$P^t \begin{pmatrix} I + D_{B}^{\frac{1}{2}} B^{-1} E B^{-t} D_{B}^{\frac{1}{2}} & D_{B}^{\frac{1}{2}} B^{-1} N D_{N}^{-\frac{1}{2}} \\ D_{N}^{-\frac{1}{2}} N^t B^{-t} D_{B}^{\frac{1}{2}} & -I \end{pmatrix} P$$

Because of the extra term on the left upper block, some of theoretical the results of the previous section are no longer valid. The reduction to an $m \times m$ system can
still be done giving the matrix

\[ I + \frac{1}{B} \cdot B^{-1}(N \cdot D_N N' + E) \cdot B^{-1} \cdot D_B^{\frac{1}{2}} \]

which can be also related to the Schur complement. However, the \( n - m \) system is not an option anymore.

Besides these drawbacks, there is an important practical reason for adding the slack variables into the problem instead of eliminating them from the linear system. This preconditioner relies on the computation of the \( LU' \) factorization of a linearly independent set of columns from \( A \). By having the slack columns as part of \( A \) we may have very inexpensive factorizations. The best case is a problem with all inequalities where the matrix \( B \) formed by the slack columns is a permutation of the identity matrix. The explicit use of the slack variables may also avoid redundant rows in \( A \). The presence of redundant rows causes the first factorization to be very expensive. The reason is that all the columns have to be factored no matter the ordering, to detect redundant rows.

We recall that the augmented system for problems with bounded variables has the same structure of the augmented system for the standard form. Therefore, the preconditioners developed here can be applied to this type of problem without any change. As for the problems with free variables, this preconditioner leads to a singular matrix. Thus, it is necessary to split the free variables into pairs of nonnegative variables.
5.10 An Equivalent Preconditioner

The choices \( J = 0, F = D^{-\frac{1}{2}}, H = [I \ 0] P D^{-\frac{1}{2}} \) and \( G^t = (H A^t)^{-1} [I \ 0] P \) on (5.1) lead to the preconditioned system

\[
M^{-1} \begin{pmatrix} -D & A^t \\ A & 0 \end{pmatrix} M^{-t} = \begin{pmatrix} -I + D^{-\frac{1}{2}} A^t G^t + G A D^{-\frac{1}{2}} & 0 \\ 0 & -I \end{pmatrix}
\]

where

\[
M^{-1} = \begin{pmatrix} D^{-\frac{1}{2}} & G \\ H & 0 \end{pmatrix}.
\]

This preconditioner is equivalent to (5.2) for all practical purposes on the context of interior point methods because

\[
(H A^t)^{-1} = ([I \ 0] P D^{-\frac{1}{2}} A^t)^{-1}
\]

\[
(H A^t)^{-1} = (D_B^{-\frac{1}{2}} [I \ 0] P A)^{-1}
= B^{-t} D_B^{\frac{1}{2}}
\]

for this choice of \( H \). However, on a more general context, where \( D \) is still positive definite but not necessarily diagonal, this preconditioner requires more computational work.

5.11 Conclusion

We presented a promising class of preconditioners which take advantage the structure of the problem and yet avoid working with the Schur complement. The price for avoiding the Schur complement is that we now have to compute an \( LU^t \) factorization for a set of linearly independent columns of \( A \). We expect that this factorization will be cheaper to compute than computing the Schur complement and its (in)complete
Cholesky factorization on many situations found in practice. Moreover, the $LU^*$ factorization can be used effectively for a number of iterations thus, computing the next few preconditioners is much more cheaper. A feature that is not shared by any known successful approach either to the Schur complement or the augmented system.

A more important advantage of these preconditioners is that they became better in some sense as the interior point method advances towards an optimal solution. This is a very welcome characteristic since these problems are known to be very ill-conditioned close to a solution. Finally, this preconditioner reduces the system to positive definite matrices, and therefore the conjugate gradient method can be applied, making it easier to compete against the Schur complement approach. This reduction on the dimension of the system to be solved from $n+m$ to either $m$ or $n-m$ is an advantage over others augmented approaches since it minimizes the overhead of the vector operations on the iterative linear solver method. This is particularly important if Krylov type methods such as MINRES are being used.

We also discussed how to compute the $LU^*$ factorization of the ordered set of columns of $A$ efficiently. The techniques presented here particularly finding symbolically dependent and independent columns can be applied to several other problems. They can be applied for finding a basis for the linear programming problem from the solution of the interior point method. The way for finding the basis is to order the columns by the distance of the variables to the bounds and then compute a factorization. That leads to exactly the same problem we are dealing with. They also can be used on nonlinear programming methods like null space and reduced Hessian methods.
Chapter 6

Linear System Solvers

In this chapter we discuss several ways to solve the augmented linear system by iterative methods. Since the conjugate gradient method is well defined and convergent only for positive definite matrices, we will focus on a more general class of Krylov subspace projection methods. Also, we investigate a new approach for solving the linear system by relating it to an equivalent eigenvalue problem.

For expository purposes, we shall change our notation. We shall be concerned with solving the $n \times n$ nonsingular indefinite linear system

$$Ax = b.$$  \hspace{1cm} (6.1)

6.1 The Preconditioned Conjugate Gradient Method

For the sake of completeness we present the well-known preconditioned conjugate gradient method. For a full development of the method see [26] for example. The following steps define the $j$-iteration $j > 1$ of the method

Solve $Kz_{j-1} = r_{j-1}$

$$3_j = \frac{(r_{j-1})^Tz_{j-1}}{(r_{j-2})^Tz_{j-2}}$$

$$p_j = z_{j-1} + 3_j p_{j-1}$$

$$\alpha_j = \frac{(r_{j-1})^Tz_{j-1}}{p_j^TAp_j}$$

$$x_j = x_{j-1} + \alpha_j p_j$$

$$r_j = r_{j-1} - \alpha_j A p_j$$
where \( r_j = b - Ax_j \) is the residual of the original linear system and \( K = M M^t \) is formed by the preconditioner. Thus if the system is not preconditioned, \( K = I \).

The advantage of using this version of the preconditioned conjugate gradient method is that it gives an approximate solution of the original system directly without having to compute it from the approximate solution of the preconditioned system.

### 6.2 Projection Methods

Krylov subspace projection methods are useful for approximately solving linear systems and for approximating eigenpairs for a matrix. The Krylov subspaces are defined by:

\[
K^j(A; b) \equiv \text{span} \{ b, Ab, \ldots, A^{j-1}b \}.
\]

Projections methods construct approximate solutions to linear systems by solving \( \min \| b - Ax \| \) such that \( x \in K^j(A; b) \) and approximate eigenpairs by finding critical points of \( \frac{w^t A w}{w^t w} \) such that \( w \in K^j(A; b) \).

One of the best known Krylov subspace method is Arnoldi's method which builds an orthogonal basis for the Krylov subspace. After \( j \)-steps of the Arnoldi method on \( A \) with \( \varepsilon_1 = \frac{b}{\|b\|} \) we get (see [46] for more details)

\[
AV_j = V_jH_j + f_j\varepsilon_j^t \tag{6.2}
\]

where the columns of \( V_j \) form the orthogonal basis of \( K^j \). \( V_j^tf_j = 0 \) and \( H_j \) is a \( j \times j \) upper Hessenberg (\( \eta_{ij} = 0, i > j + 1 \)) matrix. The next step of the Arnoldi factorization can be computed through the following formulas:

\[
\beta_j = \|f_j\|
\]
\[ \begin{align*}
    v_{j+1} &= \frac{f_j}{J_j} \\
    w &= \mathcal{A}v_{j+1} \\
    h &= V_j^t w \\
    \alpha &= v_{j+1}^t w \\
    f_{j+1} &= w - V_j h - \alpha v_{j+1} \\
    H_{j+1} &= \begin{pmatrix} H_j & h \\ J_j e_j^t & \alpha \end{pmatrix}
\end{align*} \]

where the Gram-Schmidt procedure is used for the orthogonalization. In order to guarantee the orthogonality of the columns of \( V_{j+1} \) within machine precision, an iterative refinement procedure may be used. Observe that \( \mathcal{A} \) is used only for computing a matrix-vector product as in the preconditioned conjugate gradient method.

If \( f_{j+1} = 0 \) an invariant subspace is found, that is the Krylov subspace \( K_j \) is the subspace of maximum dimension which can be achieved from the starting vector \( b \). This is a good situation in the context of solving linear systems since it will lead to the exact solution.

When \( \mathcal{A} \) is symmetric, \( H_j \) is also symmetric and therefore tridiagonal and it is denoted by \( T_j \). In this situation, the Arnoldi method reduces to Lanczos method.

### 6.2.1 Projection Methods and Linear Systems

Following the terminology in [44], we call the weak solution \( x_j^w \) the solution given by:

\[ x_j^w = V_j H_j^{-1} \epsilon_1 \| b \| \]

this is the same solution obtained by the \( j \)th step of the conjugate gradient method for a symmetric positive definite matrix. It can be shown that the residual
\( r_j^w = b - Ax_j^w \) is orthogonal to \( K^j(A; b) \) and its norm can be evaluated without computing \( x_j^w \) since \( \|r_j^w\| = \beta_j |e_j^t H_j^{-1} e_1| \cdot \|b\| \). Thus, the weak solution is exact when \( \beta_j = 0 \) as anticipated.

The weak solution does not use all the information given by the Arnoldi factorization. It is possible to obtain a better solution by using \( \beta_j \). The motivation is to find an approximation \( x_j^{MR} \) which gives the minimal residual for the linear system in the Krylov subspace \( K^j \). Thus, \( x_j^{MR} = V_j y_j \) and

\[
  r_j^{MR} = b - AV_j y_j
  = \|b\| v_1 - (V_j H_j - f_j e_j^t) y_j
  = (V_j, v_{j+1}) \begin{bmatrix} \|b\| \begin{pmatrix} \epsilon_1 \\ 0 \end{pmatrix} - \begin{pmatrix} H_j \\ \beta_j e_j^t \end{pmatrix} y_j \end{bmatrix}
\]

where \( f_j = \beta_j v_{j+1} \). Therefore,

\[
  \|r_j^{MR}\| = \left\| \begin{pmatrix} \epsilon_1 \\ 0 \end{pmatrix} - \begin{pmatrix} H_j \\ \beta_j e_j^t \end{pmatrix} y_j \right\|. \tag{6.3}
\]

Using normal equations we obtain

\[
  x_j^{MR} = V_j (H_j^t H_j + \beta_j^2 e_j^t e_j^t)^{-1} H_j^t \epsilon_1 \|b\|. \tag{6.4}
\]

This method is known as the generalized minimal residual method GMRES [47]. Since this is the solution which gives the minimal residual to the linear system in the Krylov subspace \( K^j \), \( x_j^{MR} \) is at least as good as the weak solution because \( x_j^w \in K^j(A; b) \). In fact, it can be shown that they are related by

\[
  \|r_j^{MR}\| = \frac{\|r_j^w\|}{(1 + \beta_j^2 \|H_j^{-1} e_j\|^2)^{1/2}}.
\]

Therefore, the GMRES also deliver the exact solution when \( \beta_j = 0 \). For symmetric matrices the GMRES method reduces to the MINRES method [43].
6.2.2 Projection Methods and Eigenvalues

Besides been used for solving linear systems the Arnoldi factorization (6.2) also leads to one of the most important methods for estimating a few eigenpairs of large matrices. Indeed, that was the motivation for the development of the Arnoldi factorization. We call the eigenvalues in \( H_j \hat{y} = \theta \hat{y} \) Ritz values and the eigenvalues of the generalized problem

\[
(H_j^tH_j + \beta_j^2 \epsilon_j^t \epsilon_j^t)y = \theta H_j^t y
\]  

(6.5)

are called harmonic Ritz values [44]. This name came from the fact that the values \( \theta^{-1} \) can be seen as weighted means of inverse of the eigenvalues of \( A \). In both cases, the pairs \((\theta, V_j y)\) and \((\hat{\theta}, V_j \hat{y})\) are approximations for the eigenpairs of \( A \). Usually the Ritz values are good approximations for the extreme eigenvalues of \( A \) and the reciprocals of the harmonic Ritz values are fair approximations for the extreme eigenvalues of \( A^{-1} \). These approximations are exact when \( \beta_j = 0 \) just as in the case of linear systems.

The eigenvalue approximations given by (6.2) are related to the quality of the solutions \( x_j^w \) and \( x_j^{MR} \). The Ritz pairs associated to eigenvalues of \( A \) close to zero should be a good approximation in order to \( x_j^w \) be a good approximation to the true solution. This explains why these methods work well for positive definite matrices where the eigenvalues close to zero are also extreme eigenvalues. The harmonic Ritz values are associated to \( x_j^{MR} \) in the same way as the Ritz values are associated to \( x_j^w \) [44]. Since the approximation for internal eigenvalues by harmonic Ritz values are not so good compared to the approximation of extreme eigenvalues by the Ritz values at a given step, solving indefinite systems is in general, a much more difficult task. We will see later how the (harmonic) Ritz values can be useful for solving linear systems when applying the implicit restart strategy to the GMRES.
6.3 The Quasi Minimal Residual Method

Another Krylov subspace method is the Non-Hermitian Lanczos method. This method builds a pair of biorthogonal basis for the subspaces

\[ K^j(A; v) \equiv \text{span} \{ v, Av, \ldots, A^{j-1}v \} \quad \text{and} \]
\[ K^j(A^T; w) \equiv \text{span} \{ w, A^Tw, \ldots, (A^T)^{j-1}w \}. \]

By saying that two basis (\( V \) and \( W \)) are biorthogonal we mean \( v_i^Tw_j = \delta_{ij} \).

The solution given by (6.4) when using the non-Hermitian Lanczos method is called the quasi-minimal residual solution (\( x^{QMR} \)) and the method is called QMR. The QMR method needs two matrix-vector products per iteration for unsymmetric matrices. But as Freund and Nachtigal [20] showed it is possible to use a symmetric right preconditioner together with the QMR method with the same work per iteration as the MINRES. By observing that if \( \hat{A}lv = \hat{N}l \hat{A} \) and starting with \( w_1 = \frac{Nv_1}{\|Nv_1\|} \), the Lanczos vectors satisfy \( w_j = \frac{Nv_j}{\|Nv_j\|} \), where \( \hat{A} \) is the preconditioned matrix. Hence, if we choose \( \hat{A} = AN \) (\( N \) symmetric) we save a matrix-vector product by iteration. This scheme fits very well with the preconditioner given in (4.6) and with the modified SSOR preconditioner.

6.4 Implicit Restart for the Arnoldi Factorization

The Arnoldi factorization has two major drawbacks. The computational work and storage increases with the number of iterations and the orthogonality of the basis of the Krylov subspace can be lost due to numerical difficulties. Hence this approach is effective only for \( j \ll n \). The traditional approach to overcome this problem is to restart the Arnoldi factorization from scratch after a given number of iterations using
the information obtained so far to choose the new starting vector.

In [49] Sorensen proposed a new approach that updates the Arnoldi factorization instead of restarting it. The idea is to update the factorization by filtering out unwanted eigenvalues. In this way, the computational and storage requirements can be fixed for a number of iterations and the orthogonality of the basis can be easily maintained for small dimensions.

The implicit shift restart idea is based on the explicitly shifted QR algorithm. Let $\mu$ be the shift and $H_j - \mu I = QR$ from (6.2) we have:

\[
(A - \mu I)V_j - V_j(H_j - \mu I) = f_j \epsilon_j^t
\]
\[
(A - \mu I)V_j - V_j QR = f_j \epsilon_j^t
\]
\[
(A - \mu I)(V_j Q) - (V_j Q)(RQ) = f_j \epsilon_j^t Q
\]
\[
AV_j Q - V_j Q(RQ + \mu I) = f_j \epsilon_j^t Q
\]

equating the first $j-1$ columns on both sides gives

\[
AV_{j-1}^+ = V_{j-1}^+ H_{j-1}^+ + f_{j-1}^t \epsilon_{j-1}^t
\]  (6.6)

where, $V_j^+ = V_j Q$, $H_j^+ = RQ + \mu I$ is upper Hessenberg and

\[
f_{j-1}^t = v_j(\epsilon_j^t H_j^+ \epsilon_{j-1}) + f_j(\epsilon_j^t Q \epsilon_{j-1}).
\]

Moreover, $(V_{j-1}^+)^t f_{j-1}^t = 0$ thus (6.6) is an $(j - 1)$-step Arnoldi factorization.

The idea is to apply say $p$ shifts at once, obtaining a $k$-step Arnoldi factorization where $k = j - p$. From this factorization we can compute $p$ additional steps at a cost of $p$ matrix-vector products to get back to a $j$-step factorization instead of computing $j$ steps in a traditional restarting method.
6.5 \textbf{Implicit Restart for Linear Systems}

This idea was developed for the eigenvalue problem but it can directly applied to the linear system problem. The only relation left to be determined is how to compute the linear system solution from the shifted factorization. Next we will see how it can be done.

Assume we have an initial guess $x_0$. then $r = b - Ax_0$ and $v_1 = \frac{r}{\theta}$ where $\theta = \|r\|$. Thus, if we solve $Ax_1 = r$ then $x = x_0 + x_1$. After shifting, we have the relation $r^+ = r - Ax_0^+$, where $r^+ = \theta^+ v_1^+ = \theta^+ V_j Q \epsilon_1$. But since $r \in K_1$ and $r^+ \in K_2$, then $x_0^+ \in K_1$, and we can write $x_0^+ = \tau V_j \epsilon_1$. Thus,

\[
\begin{align*}
\theta^+ V_j Q \epsilon_1 & = \theta V_j \epsilon_1 - \tau A V_j \epsilon_1 \\
\theta^+ V_j Q \epsilon_1 & = \theta V_j \epsilon_1 - \tau (V_j H_j + f_j \epsilon_j^t) \epsilon_1 \\
\theta^+ Q \epsilon_1 & = \theta \epsilon_1 - \tau (Q R + \mu I) \epsilon_1 \\
\theta^+ Q \epsilon_1 & = \theta \epsilon_1 - \tau (\rho_{11} Q + \mu I) \epsilon_1
\end{align*}
\]

hence $\tau = \frac{\theta}{\mu}$ and from it we get

\[
\begin{align*}
\theta^+ & = -\theta \frac{\rho_{11}}{\mu} \quad \text{and} \\
x_0^+ & = \frac{1}{\mu} r
\end{align*}
\]

Thus, we can compute the norm of the new residual without computing the residual itself. In fact, the norm can be easily computed even before performing the shift. The new solution depends on the Arnoldi factorization and the shift but not on the method being used for solving the linear system, e.g. the GMRES or the weak solution.
Analogously to the eigenvalue problem, the idea is to apply $p$ shifts at once obtaining a $k$-step Arnoldi factorization. The update formula for the residual can be obtained in a similar way as it is done for one shift with the help of the relation $H_jQ = QH_j^+$.

The choice of shifts is very important. The Ritz values and the harmonic Ritz values are two obvious choices. The latter may be used when we are interested in the internal eigenvalues which is the case when solving indefinite linear systems, while the former are more appropriated when working with external eigenvalues.

6.6 Invariant Subspaces and Preconditioners

In exact arithmetic the MINRES method delivers the solution in at most $n$ iterations, where $n$ is the dimension of the linear system. An early termination may occur if an invariant subspace is found. For matrices of the type (5.1) an early termination always occur as the next lemma shows.

**Lemma 6.1** Consider the matrix $K = -I + U^tV + VU^t$ where $U$ and $V^t$ are $m \times n$ matrices then the MINRES takes at most $2(n-m)$ iterations if $2m > n$ and $2m$ iterations if $2m < n$ in exact arithmetic.

**Proof** By the same arguments of theorem 5.2 either $\lambda = 1$ is an eigenvalue of $K$ with multiplicity at least $2m-n$ if $2m > n$, or $\lambda = -1$ has multiplicity at least $n-2m$ if $2m < n$. Recall that the Arnoldi factorization gives exact approximation for the eigenvalues on an invariant subspace. Since an upper Hessenberg matrix with nonzero lower subdiagonal entries has eigenvalues with multiplicity one (see [26]), the Arnoldi factorization finds an invariant subspace after estimating each different eigenvalue at most once. \(\square\)
6.7 Implementation Issues for MINRES

6.7.1 Computing the Residual

It is possible to verify if convergence is achieved at each iteration by computing a QR factorization and the residual of the least square problem (6.3):

\[
\begin{pmatrix}
H_j \\
J_j e_j^t
\end{pmatrix}
= \hat{Q}_j
\begin{pmatrix}
\hat{R}_j \\
0
\end{pmatrix}
\]

thus.

\[
\|r_j^{MR}\| = \|b\| \begin{pmatrix}
\hat{q} \\
\gamma
\end{pmatrix} - \begin{pmatrix}
\hat{R}_j \\
0
\end{pmatrix} y
\]

where.

\[
\begin{pmatrix}
\hat{q} \\
\gamma
\end{pmatrix} = \hat{Q}_j^t \begin{pmatrix}
e_1 \\
0
\end{pmatrix}.
\]

Therefore, \(\|r_j^{MR}\| = |\gamma|\) and \(\hat{R}_j y = \|b\| \hat{q}\). This process is very cheap since it can be done by computing a sweep of Givens rotations [26] applied to a tridiagonal matrix and it is not necessary to compute the approximation solution to verify the residual. Another way the algorithm can stop is by checking whether we achieved an invariant subspace, i.e. we verify if \(J_j\) is smaller than a given tolerance.

6.7.2 Computing the Approximate Solution

Regarding the approximate solution, it is possible to compute the minimal residual solution at about the same work spent to obtain the weak solution:

\[
x_j^{MR} = V_j \hat{R}_j^{-1} \hat{Q}_j^t e_1 \|b\|
\]

\[
x_j^w = V_j R_j^{-1} Q_j^t e_1 \|b\|
\]

where \(H_j = Q_j R_j\).
6.7.3 Computing the Shifts

The implementation includes the implicit restart using optionally either the Ritz values or the harmonic Ritz values as shifts. The computation of the Ritz values is straightforward. Since $H$ is tridiagonal for symmetric matrices, we apply the symmetric $QR$ method with implicit shift. The computation of the harmonic Ritz values requires more careful thought. Consider again its definition

$$\begin{align*}
(H_j^t H_j + 3_j^2 \epsilon_j \epsilon_j^t) y &= 0 H_j^t y
\end{align*}$$

Since we already have $H_j = Q_j R_j$ we can write

$$\begin{align*}
(R_j^t R_j + 3_j^2 \epsilon_j \epsilon_j^t) y &= 0 R_j^t Q_j^t y \\
(R_j + 3_j^2 R_j^{-t} \epsilon_j \epsilon_j^t) y &= 0 Q_j^t y \\
(Q_j R_j + \frac{3_j^2}{\rho_{jj}} Q_j \epsilon_j \epsilon_j^t) y &= 0 y
\end{align*}$$

$$\begin{align*}
\tilde{H}_j y &= 0 y
\end{align*}$$

where $\tilde{H}$ is an upper-Hessenberg matrix since

$$\tilde{H}_j = H_j + \frac{3_j^2}{\rho_{jj}} q \epsilon_j^t$$

and $q$ is the $j$th-column of $Q_j$.

Therefore the harmonic Ritz pairs can be calculated at about the same computational effort as the Ritz values for unsymmetric matrices. This is not true for symmetric matrices because $\tilde{H}_j$ would not be symmetric even though $H_j$ is. Next, it follows an alternative way for the symmetric problem where symmetry can be kept. It implies a little more work for recovering the harmonic Ritz vectors should they be needed. However, it is not the case in this work.
The standard way to solve the generalized eigenvalue problem $Ax = \lambda Bx$ when $B$ is symmetric positive definite is to compute the Cholesky factorization $B = LL^t$ and solve the related eigenvalue problem $L^{-1}AL^{-t}z = \lambda z$. But since $T_j$ is not always positive definite we choose to use the Cholesky factorization of the left hand side matrix in (6.5) and solve a related eigenvalue problem for the reciprocal of the harmonic Ritz values (remember that for the symmetric case $H \equiv T$). The computation of the Cholesky factorization consists only in calculating $\lambda_{jj} = (\rho_{jj}^2 + \beta_j^2)^{\frac{1}{2}}$ since
\[
L_j L_j^t y = (R_j^t R_j + J_j^t \epsilon_j^t \epsilon_j^t) y = \theta T_j y
\] (6.7)
and $R_j$ is already computed. The upper triangular matrix $R_j$ may have negative diagonal entries thus $L_j L_j^t$ it is not necessarily a Cholesky factorization nonetheless. It serves our purpose as well. Notice that we obtain a dense $j \times j$ symmetric matrix. Some work can be saved by taking advantage of its symmetry and from the fact that $T_j$ is tridiagonal.

### 6.8 The Eigenvalue Problem Approach

#### 6.8.1 Introduction

It is known from practice that it may be easier to find a few extreme eigenvalues of a matrix than to solve a linear system. This notion is explained and quantified in [44] where it is shown that in order to compute a good approximate solution for an indefinite linear system it is necessary to obtain good approximations for the smaller eigenvalues. Thus, one approach to find the solution for (6.1) may consist in solving a related eigenvalue problem. Consider
\[
\begin{pmatrix}
\alpha & -c^t \\
-b & A
\end{pmatrix}
\begin{pmatrix}
1 \\
x
\end{pmatrix}
= \lambda
\begin{pmatrix}
1 \\
x
\end{pmatrix}
\] (6.8)
where, $\alpha^* = c^* x = c^* A^{-1} b$ is the unique value such that the whole matrix is singular. hence the eigenvector related to $\lambda = 0$ gives the solution for linear system.

If we choose $c = A^d$, we would take $\alpha = d^b$ to obtain a singular matrix. The disadvantage of this approach is that the system is no longer symmetric. Another problem is the lack of criteria for choosing $d$ unless we have a good starting point. But then this will be a good choice for $d$ only for symmetric problems and we get back to the our first concern. For these reasons we chose to study (6.8) with $c \equiv b$.

The idea of finding the solution for a linear system by changing it to an eigenvalue problem where the matrix is singular is not new. As early as 1952 Lanczos [31] proposed a similar approach and suggested applying the power method for finding the eigenvector.

6.8.2 Making the Matrix Singular

The crucial point of our approach is to compute a good approximation for $\alpha^*$. From (6.8) we have:

$$\alpha = o(\lambda) + \lambda$$

where, $o(\lambda) \equiv b^* x$ and $\alpha^* = o(0)$. Thus, a first idea is to guess an initial value $o_0$. approximate $o$ by the rational function

$$\bar{o} = \frac{\hat{\gamma}_1^2}{\delta_1 - \lambda} + \frac{\hat{\gamma}_2^2}{\delta_2 - \lambda}$$

(6.9)

and get a new value for $\alpha$ setting $\lambda = 0$.

This problem proved to be very hard to solve. One reason is that for ill-conditioned matrices several values for $\alpha$ and $x(\alpha)$ in (6.8) give about the same value for $\bar{o}$ in
(6.9) where \( \lambda \) is the eigenvalue closer to zero. The scheme that works better is as follows: choose \( \alpha_0 = 0 \) and

\[
\alpha_1 = \text{sign}(\phi(\lambda(\alpha_0))) b^T b
\]

(6.10)

this usually is a good approximation for \( \alpha^* \) since we are guessing \( x(\alpha_1) = \text{sign}(\alpha^*) b \) as we will show later. Finally, we approximate \( \phi \) by \( \phi \) and using its first derivative \( \phi'(\lambda) = x^T x \). we compute two more guesses for \( \alpha^* \). Next using the four points in hand we approximate the rational polynomial iteratively without using the derivative. until a good value for \( \alpha^* \) is obtained.

This scheme converges very fast. The drawback is that we have to solve an eigenvalue problem at each time making it non-competitive with MINRES even when these eigenvalue problems converge in few iterations. Sorensen [50] showed superlinear convergence for a similar approach for the minimization of a quadratic function subject to an ellipsoidal constraint. Even though the method does not appear competitive at this point, it is of some theoretical interest and we wish to analyze it.

The following lemma proves that \( \alpha_1 \) defined in (6.10) has the desired sign.

**Lemma 6.2** Consider the problem defined in (6.8) with a given \( \alpha \) and let

\[
\phi(\lambda) = b^T (A - \lambda I)^{-1} b
\]

where \( \lambda \) is the eigenvalue with smallest modulus. Then.

\[
\lambda < 0 \Rightarrow \alpha < \alpha^*.
\]

\[
\lambda = 0 \Rightarrow \alpha = \alpha^* \text{ and}
\]

\[
\lambda > 0 \Rightarrow \alpha > \alpha^*
\]
where \( \alpha^* \) is the unique value which makes (6.8) singular.

**Proof**  We can rewrite \( \phi \) as follows:

\[
\phi(\lambda) = \sum_{i=-r}^{s-1} \frac{\gamma_i^2}{\delta_i - \lambda} + \sum_{i=s}^{s} \frac{\gamma_i^2}{\delta_i - \lambda}
\]

where \( \delta_i (i = -r, \ldots, -1, 1, \ldots, s) \) are the eigenvalues of \( A \). By the interlacing theorem, \( \lambda \in (\delta_{-1}, \delta_1) \) where \( \delta_{-1} \) is the largest negative eigenvalue of \( A \) and \( \delta_1 \) is its smallest positive eigenvalue. Thus, \( \phi \) is monotonically increasing in \( (\delta_{-1}, \delta_1) \).

There are three cases to consider:

(i) \( \lambda = 0 \). therefore \( \alpha = \alpha^* \).

(ii) By (6.8.2) \( \lambda > 0 \) implies \( \alpha > \phi(\lambda) \). As \( \lambda \) decreases towards zero, \( \phi(\lambda) \) decreases too thus \( \alpha > \phi(\lambda) > \phi(0) = \alpha^* \).

(iii) \( \lambda < 0 \) is similar to case (ii).

\( \Box \)

By taking \( \alpha = 0 \), we can see that \( \text{sign}(\phi(\lambda)) = \text{sign}(\alpha^*) \). Thus, \( \alpha_1 \) given by (6.10) has the same sign as \( \alpha^* \).

### 6.8.3 Estimating \( \alpha^* \)

The best results we obtained for this approach use the MINRES. The idea is to run the MINRES until the norm of the residual smaller then the square root of the tolerance desired (note: this is a heuristic approach). Then we compute \( \alpha \) such that

\[
\begin{pmatrix}
\alpha & -\|b\|e_1^T \\
-\|b\|e_1 & T_j
\end{pmatrix}
\]  

(6.11)
is singular. Here $T_j$ is the tridiagonal matrix given by the Lanczos factorization.

If we denote the normalized eigenvector corresponding to $\lambda = 0$ in (6.11) as $(1 \ y)^t$, we notice that $V_jy$ is the weak solution given by the MINRES. This is a very good starting vector for the Lanczos method. The minimum residual solution is also a good starting point. It is also possible to save some work with another starting vector by noting that

$$
\begin{pmatrix}
\alpha & -b^t \\
-b & A
\end{pmatrix}
\begin{pmatrix}
1 & 0^t \\
0 & V_j
\end{pmatrix}
= 
\begin{pmatrix}
\alpha & -b^t V_j \\
-b & AV_j
\end{pmatrix}
$$

now using (6.2) and $v_1 = \frac{1}{\|b\|}$ we have

$$
\begin{pmatrix}
\alpha & -b^t \\
-b & A
\end{pmatrix}
\begin{pmatrix}
1 & 0^t \\
0 & V_j
\end{pmatrix}
\approx 
\begin{pmatrix}
\alpha & -\|b\|\epsilon_1^t \\
-b & V_j T_j + f \epsilon_j^t
\end{pmatrix}
$$

and thus,

$$
\begin{pmatrix}
\alpha & -b^t \\
-b & A
\end{pmatrix}
\begin{pmatrix}
1 & 0^t \\
0 & V_j
\end{pmatrix}
= 
\begin{pmatrix}
1 & 0^t \\
0 & V_j
\end{pmatrix}
\begin{pmatrix}
\alpha & -\|b\|\epsilon_1^t \\
-b & T_j
\end{pmatrix}
+ 
\begin{pmatrix}
0 \\
f_j
\end{pmatrix} \epsilon_{j+1}^t
$$

which is the $(j + 1)th$-step of Lanczos factorization with starting vector $\epsilon_1$. In the tests it worked as well as a fresh start with the weak solution.

6.8.4 **Bounds for $\alpha^*$**

The approximation for $\alpha^*$ given in (6.11) is equivalent to a Gauss quadrature rule for the integration of the function $\frac{1}{x}$ for symmetric positive definite matrices since

$$
\frac{\alpha^*}{\|b\|^2} = \epsilon_1^t T_n^{-1} \epsilon_1 = \sum_{i=1}^{n} \frac{\eta_i^2}{\delta_i} = \int_{\delta_1}^{\delta_n} \frac{1}{x} dw(x)
$$

where $QDQ^t$ is the eigenvalue decomposition of $T_n$ and we assume it has $n$ distinct eigenvalues. Golub and von Matt use it in [25] to compute the following bounds for
\( \alpha^* \) which are valid only for positive definite systems:

\[
e^iT_j^{-1}e_1 \leq e_1^iT_n^{-1}e_1 \leq e_1^i \tilde{T}_j^{-1}e_1
\]

where the lower bound is obtained using the Gauss quadrature rule with the eigenvalues of \( T_j \) as abscissas:

\[
E[\frac{1}{x}] = \int_a^b \frac{1}{x}dw(x) - \sum_{i=1}^J \frac{w_i}{x_i} = \frac{(-1)^{2J}}{\xi_{2J+1}} \int_a^b (x-x_i)^2dw(x) > 0
\]

where \( 0 < a \leq \delta_1 < \delta_n \leq b \). In order to compute the upper bound they use the Gauss-Radau rule fixing one of \( \tilde{T}_j \) eigenvalues to \( a \).

This result can be generalized for indefinite matrices in the following way:

Let \( I = \frac{\alpha^*}{\|b\|^2} = e_1^iT_n^{-1}e_1 = \sum_{i=1}^s \frac{q_{ii}^2}{\delta_i} + \sum_{i=-r}^{-1} \frac{\tilde{q}_{ii}^2}{\delta_i} \)

and

\[
I^+ = \int_{\varepsilon_1}^{\varepsilon_s} \frac{1}{x}dw(x) = \sum_{i=1}^s \frac{q_{ii}^2}{\delta_i}
\]

\[
I^- = \int_{-\varepsilon_1}^{-\varepsilon_s} \frac{1}{x}dw(x) = \sum_{i=-r}^{-1} \frac{\tilde{q}_{ii}^2}{\delta_i}
\]

thus, \( I = I^+ - I^- \). From [25] one can obtain a lower bound to \( I^+ \) and \( I^- \) applying the Gauss quadrature rule to \( T_j \). In a similar way, an upper bound for both is given by the Gauss-Radau rule fixing one of the eigenvalues of \( \tilde{T}_j \) and \( \bar{T}_j \) to \( \delta_1 \) and \( \delta_{-1} \) respectively. Hence if we define

\[
\frac{\alpha_L}{\|b\|^2} = \sum_{i=1}^s \frac{q_{ii}^2}{\delta_i} + \sum_{i=-r}^{-1} \frac{\tilde{q}_{ii}^2}{\delta_i}
\]

and

\[
\frac{\alpha_U}{\|b\|^2} = \sum_{i=1}^s \frac{q_{ii}^2}{\delta_i} + \sum_{i=-r}^{-1} \frac{\tilde{q}_{ii}^2}{\delta_i}
\]
we have $\alpha_L \leq \alpha^* \leq \alpha_U$. These bounds are more costly to compute then in the positive definite case since now we have to solve three eigenvalue problems (for $T_j$, $\tilde{T}_j$ and $\hat{T}_j$) instead of solving a couple of linear systems. Yet they are computable since $j$ is much smaller than $n$.

These bounds can be directly applied to the matrix moments

$$
\mu_{-m} = \epsilon_1^j T_{-m}^j \epsilon_1 \quad \text{with} \quad m \geq 1.
$$

In this case, the integrand function is $\frac{1}{x^m}$. All the results showed before for $m = 1$ hold for any positive $m$. Therefore we have a generalization for symmetric indefinite matrices to theorem 4.1 on [25]:

$$
\sum_{i=1}^{k_j} \frac{q_{ii}^2}{\delta_{ii}^m} + \sum_{i=-c_j}^{-1} \frac{q_{ii}^2}{\delta_{ii}^m} \leq \mu_{-m} \leq \sum_{i=1}^{k_j} \frac{q_{ii}^2}{\delta_{ii}^m} + \sum_{i=-c_j}^{-1} \frac{q_{ii}^2}{\delta_{ii}^m}.
$$

### 6.8.5 Stopping Rules for the Eigenvalue Method

In order to solve the eigenvalue problem we use the Lanczos method, both in its original form and using implicit shifts to restart the process [49]. We use as shifts the $p$ biggest harmonic Ritz values on a $j$ step of Lanczos factorization, since preliminary experiments showed them to be slightly better shifts than the Ritz values. We can use both the Ritz and harmonic Ritz values for stopping criteria. For the Ritz values we have the well known result

$$
\|(M - \lambda I) V_j \tilde{y} \| = \beta_j |\epsilon_j^j \tilde{y}| \quad (6.12)
$$

where

$$
M = \begin{pmatrix}
\alpha & -b \\
-b^t & A
\end{pmatrix}
$$

and $\tilde{y}$ is normalized in such a way that

$$
\begin{pmatrix}
1 \\
x
\end{pmatrix} = V_j \tilde{y}. \quad (6.13)
$$
Now, from the definition of harmonic Ritz values (6.5) we have

\[ H_j y = \lambda y - \delta_j^2 t \epsilon_j^t y \]

where \( t = H_j^{-1} \epsilon_j \) (we have to be careful in computing \( t \) since \( H_j \) is expected to be nearly singular). Applying it to

\[ MV_j = V_j H_j + f_j \epsilon_j^t \quad (6.14) \]

we get:

\[ \| (M - \lambda I) V_j y \| = \delta_j |\epsilon_j^t y| \| v_{j+1} - \delta_j V_j t \| \quad (6.15) \]

where \( v_{j+1} = \frac{f_j}{\delta_j} \).

In both cases, we stop when the norm in the left hand side for the smallest eigenvalue is less than the desired tolerance. The rule given by (6.15) worked better than (6.12) for most of the cases tested.

### 6.8.6 Stopping Rules for the Linear System

The previous rules are applied to measure the error for the smallest eigenvalue, but the goal is to solve the linear system. Hence, if we ignore the first equation of (6.14) it is possible to compute the norm of the residual for the linear system without computing the approximation for \( x \). Thus for the Ritz values we get:

\[ \| b - Ax \| = |\lambda x + (\epsilon_j^t \tilde{y}) \tilde{f} \| \]

where \( \tilde{y} \) is normalized as in (6.13) and \( f_j = (\sigma_1 \tilde{f})^t \). For the harmonic Ritz values we have:

\[ \| b - Ax \| = |\lambda x + (\epsilon_j^t y) (\tilde{f} - \delta_j^2 (\epsilon_j^t y) \tilde{u}) \| \]
where, \((\omega_1 \hat{w})^t = V_j t\) and \(y\) is the same as in (6.15).

On the other hand, the first equation of (6.14) may be used to update \(\alpha\). But, since it is very expensive to restart the Lanczos method from scratch, we suggest to continue to iterate with the modified matrix. This works well because most of the time \(\alpha\) does not change much, although in a few examples we could not get high precision on the solution due to the error introduced.

### 6.8.7 Other Eigenvalue Solvers

Given an estimation for \(\alpha^*\), it may be better to use another method for finding eigenvector approximations instead of Arnoldi's since we need to compute only one eigenvector. A new variant of Davidson's method, the Jacobi-Davidson method proposed in [48], seems to be a promising choice. In Davidson's method, given an orthogonal basis \(V_j\) and a Ritz pair \((\theta, u)\) of \(A\), one computes \(r = Au - \theta u\) and \((\text{diag}(A) - \theta I)t = -r\), and then orthogonalizes \(t\) against \(V_j\) for the next iteration. The new method generalizes Davidson's by making \(t\) be an orthogonal projection on the subspace generated by the desired eigenvector \(u\):

\[
(I - uu^t)(A - \theta I)(I - uu^t)t = -r
\]

besides that in [48], a definition for harmonic Ritz values for general subspaces is given. They can be seen as the eigenvalues of \(H_j = (W_j^tV_j)^{-1}W_j^tAV_j\) where \(W_j = AV_j\) and the columns of \(V_j\) form a basis for a \(j\)-dimensional subspace. If this subspace is a Krylov subspace the definition above reduces to the definition of harmonic Ritz values given before (6.5). This definition can be used to estimate internal eigenvalues by the Jacobi-Davidson method. Preliminary experiments with this method did not get competitive results, but further investigation should be made.
Chapter 7

Numerical Experiments and Conclusions

In this chapter we present several numerical experiments with the new preconditioner. These experiments are meant to expose the type of problems where the new approach performs better. Therefore these experiments are not to be seen as a way to determine which approach for the interior point methods is better in a general context. For instance, we observe that the new preconditioner fails to achieve convergence for many problems from the netlib collection under the strict conditions the experiments were made. Thus, it is necessary to improve robustness of the new approach in order to consider it as the first option in a more general context. Nevertheless, as it can be inferred by the results, it is an important option for some class of problems. Moreover, its importance increases when large scale problems are involved.

7.1 The Set of Test Problems

In this section we briefly describe the problems used on the numerical experiments. Problems FIT1P and FIT2P model the fitting of linear inequalities to data by minimizing a sum of piecewise-linear penalties. These problems belong to the netlib collection of linear programming problems.

The PDS model [9] is a multi-commodity problem with 11 commodities and whose size is a function of the number of days being modeled. Thus, PDS-2 models two days. PDS-20 models twenty days and so on. A generator for this model is available and experiments for problems of a variety of sizes are presented.
DIFFICULT is the linear programming relaxation of a large network design problem. Since the edges have capacity that are integral, the original problem is an integer problem. This problem was supplied by Eva Lee and formulated by Daniel Bienstock.

The QAP problems are models for the linearized quadratic assignment problem [42]. A generator for this type of problems can be obtained from netlib. Another source for this type of problems is the QAPLIB [8]. The problems tested here for the QAP model are from this collection with the modification described in [42]. Since the new preconditioner obtain nice computational results for this type of problems, even for the small ones, we present results for more problems of this class.

7.2 Numerical Experiments

The following computational results compare the behavior of the direct method approach against the new preconditioner based upon the computation of an $LU^*$ factorization for a subset of columns of $A$. The preconditioned positive definite matrix (5.8) is used for the experiments. Recall that it has the dimension of the Schur complement matrix. The $LU^*$ approach uses the preconditioned gradient conjugate method as solver for the linear system.

The algorithms and procedures are coded in C by the author with exception of the reordering function for the Cholesky factorization which was implemented by Eva Lee using the minimum degree strategy. I/O routines from the CPLEX callable library are used to input the problems in the standard MPS format.

All the experiments of this chapter are carried out on a Sun Ultra-Sparc station. The floating point arithmetic is IEEE standard double precision. The stopping tol-
erance for the interior point method and preconditioned conjugate gradient is the square root of machine epsilon. The predictor-corrector variant is adopted. The preprocessing and scaling steps, stopping rules and the parameters are described in Chapter 3. The only exception is that the slack variables are added into the problem explicitly.

Concerning the computation of the $LU'$ factorization for the preconditioner, the following techniques presented on Chapter 5 are used as default:

- A new $LU'$ factorization is computed from one iteration to the other whenever
  - The preconditioned conjugate gradient method takes too many iterations to achieve convergence or:
  - The true residual for the linear system is larger than the working tolerance:
- A second $LU'$ factorization is computed whenever the first factorization is too dense:
- Key columns are used to speed up the $LU'$ factorization:
- Symbolically independent columns are moved to the front and reordered:
- The merging strategy as described before is used.

Table 7.1 contains the basic statistics about the test problems. The dimension and number of nonzero entries shown for the matrix of constraints refer to the preprocessed problems. The number of nonzero entries for the Schur complement includes only the lower half of the matrix. The number of nonzero entries for matrix $L$ of the Cholesky factorization is obtained after reordering the rows of $A$ by the minimum
<table>
<thead>
<tr>
<th>Problem</th>
<th>Dimension</th>
<th>Number of Nonzero Entries</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Matrix $A$</td>
<td>Schur Complement</td>
</tr>
<tr>
<td>FIT1P</td>
<td>$627 \times 1677$</td>
<td>9868</td>
</tr>
<tr>
<td>FIT2P</td>
<td>$3000 \times 13525$</td>
<td>50284</td>
</tr>
<tr>
<td>DIFFICULT</td>
<td>$31427 \times 274347$</td>
<td>814994</td>
</tr>
<tr>
<td>PDS-01</td>
<td>$1148 \times 3729$</td>
<td>7440</td>
</tr>
<tr>
<td>PDS-02</td>
<td>$2499 \times 7535$</td>
<td>15534</td>
</tr>
<tr>
<td>PDS-06</td>
<td>$9068 \times 28655$</td>
<td>60944</td>
</tr>
<tr>
<td>PDS-10</td>
<td>$15538 \times 48670$</td>
<td>104550</td>
</tr>
<tr>
<td>PDS-15</td>
<td>$25187 \times 77043$</td>
<td>165819</td>
</tr>
<tr>
<td>PDS-20</td>
<td>$33874 \times 105728$</td>
<td>227199</td>
</tr>
<tr>
<td>PDS-25</td>
<td>$39977 \times 131239$</td>
<td>281299</td>
</tr>
<tr>
<td>CHR20A</td>
<td>$4219 \times 7810$</td>
<td>27380</td>
</tr>
<tr>
<td>CHR20B</td>
<td>$4219 \times 7810$</td>
<td>27380</td>
</tr>
<tr>
<td>CHR20C</td>
<td>$4219 \times 7810$</td>
<td>27380</td>
</tr>
<tr>
<td>CHR22A</td>
<td>$5587 \times 10417$</td>
<td>36520</td>
</tr>
<tr>
<td>CHR22B</td>
<td>$5587 \times 10417$</td>
<td>36520</td>
</tr>
<tr>
<td>CHR25A</td>
<td>$8149 \times 15325$</td>
<td>53725</td>
</tr>
<tr>
<td>ELS19</td>
<td>$4350 \times 13186$</td>
<td>50882</td>
</tr>
<tr>
<td>HIL12</td>
<td>$1355 \times 3114$</td>
<td>15612</td>
</tr>
<tr>
<td>NUG05</td>
<td>$89 \times 135$</td>
<td>495</td>
</tr>
<tr>
<td>NUG15</td>
<td>$2729 \times 9675$</td>
<td>38910</td>
</tr>
<tr>
<td>ROU10</td>
<td>$839 \times 1765$</td>
<td>8940</td>
</tr>
<tr>
<td>SCR12</td>
<td>$1151 \times 2784$</td>
<td>10716</td>
</tr>
<tr>
<td>SCR15</td>
<td>$2234 \times 6210$</td>
<td>24060</td>
</tr>
<tr>
<td>SCR20</td>
<td>$5079.5 \times 15980$</td>
<td>61780</td>
</tr>
</tbody>
</table>

Table 7.1  Basic Statistics
degree criteria.

Table 7.2 compares the number of iterations between the new preconditioner and the Cholesky factorization approach for the interior point method. Column Fact. contains the number of $LU'$ factorizations needed for the interior point methods on this strategy. This number includes the factorization for computing the starting point. Column Refact. shows the number of refactorizations performed. Notice that the number of iterations for the interior point methods on both approaches is about the same for most problems. Only problem PDS-20 presented a large difference. There is no results for problem PDS-25 on the number of iterations for the Cholesky approach because we stopped the process due to the large amount of time it was taking.

It is interesting to notice that the direct approach does not obtain a clear advantage over the iterative approach for these problems. The most probably reason is the small tolerance chosen for the experiments. We remark that the solutions obtained in these experiments agree on at least eight digits for all the problems whose objective value are known to us.

Table 7.3 shows a comparison between both approaches for the total running time and number of floating point operations. The number of floating point operations shown is the average for computing and solving the linear systems for the interior point method.

For most problems tested, the $LU'$ approach takes less total time for solving the problems and less floating point operations in average. It is no surprise since these models were chosen for this reason. It should be pointed out that for all the other
<table>
<thead>
<tr>
<th>Problem</th>
<th>Cholesky Iterations</th>
<th>LU Approach</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>FIT1P</td>
<td>22</td>
<td>23</td>
<td>14</td>
<td>12</td>
</tr>
<tr>
<td>FIT2P</td>
<td>25</td>
<td>25</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>DIFFICULT</td>
<td>30</td>
<td>30</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>PDS-01</td>
<td>23</td>
<td>21</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>PDS-02</td>
<td>27</td>
<td>29</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>PDS-06</td>
<td>39</td>
<td>38</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>PDS-10</td>
<td>50</td>
<td>54</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>PDS-15</td>
<td>63</td>
<td>61</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>PDS-20</td>
<td>63</td>
<td>73</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>PDS-25</td>
<td>75</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHR20A</td>
<td>24</td>
<td>24</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>CHR20B</td>
<td>27</td>
<td>26</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td>CHR20C</td>
<td>19</td>
<td>19</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>CHR22A</td>
<td>24</td>
<td>24</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>CHR22B</td>
<td>29</td>
<td>30</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td>CHR25A</td>
<td>33</td>
<td>31</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>ELS19</td>
<td>32</td>
<td>31</td>
<td>26</td>
<td>24</td>
</tr>
<tr>
<td>HIL12</td>
<td>18</td>
<td>19</td>
<td>16</td>
<td>14</td>
</tr>
<tr>
<td>NUG05</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>NUG15</td>
<td>21</td>
<td>21</td>
<td>22</td>
<td>20</td>
</tr>
<tr>
<td>ROU10</td>
<td>19</td>
<td>19</td>
<td>15</td>
<td>13</td>
</tr>
<tr>
<td>SCR12</td>
<td>12</td>
<td>15</td>
<td>14</td>
<td>12</td>
</tr>
<tr>
<td>SCR15</td>
<td>23</td>
<td>23</td>
<td>22</td>
<td>20</td>
</tr>
<tr>
<td>SCR20</td>
<td>24</td>
<td>24</td>
<td>25</td>
<td>23</td>
</tr>
</tbody>
</table>

<p>| Table 7.2 | Comparison between Cholesky and LU approach - Number of Iterations |</p>
<table>
<thead>
<tr>
<th>Problem</th>
<th>Cholesky</th>
<th></th>
<th>LU Approach</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>M-Flops</td>
<td>Time</td>
<td>M-Flops</td>
</tr>
<tr>
<td>FIT1P</td>
<td>308</td>
<td>87.6</td>
<td>19.7</td>
<td>4.7</td>
</tr>
<tr>
<td>FIT2P</td>
<td>44122</td>
<td>9110.3</td>
<td>505</td>
<td>106.7</td>
</tr>
<tr>
<td>DIFFICULT</td>
<td>51590</td>
<td>7335.2</td>
<td>29964</td>
<td>6115.8</td>
</tr>
<tr>
<td>PDS-01</td>
<td>4.63</td>
<td>0.3</td>
<td>22.3</td>
<td>6.8</td>
</tr>
<tr>
<td>PDS-02</td>
<td>23.97</td>
<td>1.9</td>
<td>102.6</td>
<td>22.9</td>
</tr>
<tr>
<td>PDS-06</td>
<td>1528</td>
<td>207.1</td>
<td>1676</td>
<td>237.6</td>
</tr>
<tr>
<td>PDS-10</td>
<td>8885</td>
<td>927.7</td>
<td>5565</td>
<td>529.0</td>
</tr>
<tr>
<td>PDS-15</td>
<td>51096</td>
<td>4482.6</td>
<td>16450</td>
<td>1360.0</td>
</tr>
<tr>
<td>PDS-20</td>
<td>103717</td>
<td>8911.9</td>
<td>33130</td>
<td>2275.2</td>
</tr>
<tr>
<td>PDS-25</td>
<td>16705.2</td>
<td>52676</td>
<td>3529.4</td>
<td></td>
</tr>
<tr>
<td>CHR20A</td>
<td>1161</td>
<td>244.1</td>
<td>337</td>
<td>77.2</td>
</tr>
<tr>
<td>CHR20B</td>
<td>1189</td>
<td>232.6</td>
<td>432</td>
<td>93.2</td>
</tr>
<tr>
<td>CHR20C</td>
<td>1093</td>
<td>285.8</td>
<td>273</td>
<td>75.2</td>
</tr>
<tr>
<td>CHR22A</td>
<td>1968</td>
<td>414.8</td>
<td>618</td>
<td>138.6</td>
</tr>
<tr>
<td>CHR22B</td>
<td>2405</td>
<td>433.0</td>
<td>709</td>
<td>110.5</td>
</tr>
<tr>
<td>CHR25A</td>
<td>6501</td>
<td>995.4</td>
<td>1456</td>
<td>232.2</td>
</tr>
<tr>
<td>ELS19</td>
<td>26560</td>
<td>5212.0</td>
<td>16861</td>
<td>454.2</td>
</tr>
<tr>
<td>HIL12</td>
<td>1064</td>
<td>386.6</td>
<td>974</td>
<td>294.9</td>
</tr>
<tr>
<td>NUG05</td>
<td>0.15</td>
<td>0.1</td>
<td>0.27</td>
<td>0.3</td>
</tr>
<tr>
<td>NUG15</td>
<td>11184</td>
<td>3473.8</td>
<td>14643</td>
<td>3431.2</td>
</tr>
<tr>
<td>ROU10</td>
<td>275</td>
<td>98.1</td>
<td>259</td>
<td>107.2</td>
</tr>
<tr>
<td>SCR12</td>
<td>321</td>
<td>136.0</td>
<td>145</td>
<td>57.9</td>
</tr>
<tr>
<td>SCR15</td>
<td>3659</td>
<td>1008.3</td>
<td>1470</td>
<td>190.0</td>
</tr>
<tr>
<td>SCR20</td>
<td>47480</td>
<td>12331.3</td>
<td>25679</td>
<td>1397.0</td>
</tr>
</tbody>
</table>

Table 7.3  Comparison between Cholesky and LU approach - Time and Flops
current netlib problems the Cholesky factorization approach is faster. The purpose of these experiments is to show the type of problems where the new approach is expected to perform better. For example, on problems like FIT1P and FIT2P that have dense columns, the Schur complement matrix is already very dense as it can be seen on table 7.1. Thus, although the minimum degree reordering worked very well since no fill-in entry is generated, this approach takes much more computational effort than the $LL^T$ approach.

![Figure 7.1 Running Time for PDS Model](image)

The QAP model problems also lead to Schur complement matrices that are not much sparse although in a lesser degree than the FIT problems. This, together with the fact that the factorization generates a large number of fill-in entries makes the Cholesky approach less effective. This is more easily observed as the size of the problems grows. The only problems for this class where the Cholesky factorization approach performs better are NUG05 which is very small and NUG15 where it was
faster although it takes a little more floating point operations. The $LU$ approach does not perform very well for this problem because a new factorization is computed for every single iteration. Notice that the opposite occur to problem $ROU'10$. The new approach is faster nonetheless it takes more floating point operations.

The explanation for this apparently contradictory result seems to be the number of $LU$ factorizations performed. This factorization have a large overhead compared with solving triangular systems. Thus, problems that need to compute few factorizations spend less time performing work not related to floating point operations. Since the Cholesky approach takes a significant amount of time finding a reordering, it may take more total running time even though it computes less floating point operations.

![Figure 7.2 Flops Count for PDS Model](image)

The PDS model problems do not generate dense Schur complement matrices. On the other hand, the Cholesky factorization can generate many fill-in entries. As the
dimension of the problem increases, the $LU'$ factorization approach performs better compared to the direct approach. This is better illustrated on figures 7.1 and 7.2.

Another factor that helps the $LU'$ preconditioner obtain good results for the PDS model which is not present for the QAP model is that the number of $LU'$ factorizations is very small compared to the number of iterations. Therefore, computing a solution for the linear systems on a large number of iterations for these problems is inexpensive since no factorization is computed. This fact also applies for problem DIFFICULT explaining the good performance of the new approach even considering that the Schur complement matrix is sparse.

Table 7.1 shows in more detail the effect of the heuristics used for keeping the $LU'$ factorization and computing a second factorization for problem DIFFICULT. Column PCG contains the number of iterations taken by the preconditioned conjugate gradient method to achieve convergence. This result is for the first linear system of the interior point method. The numbers for the second linear system are similar. The number of nonzero entries for the first and the second $LU'$ factorization are shown only for the iterations where a new factorization is computed. This number refers to the sum of nonzero entries of $L$ and $U'$.

Notice that only five factorizations are computed on thirty iterations of the interior point method. A second factorization is computed four times. The second factorization is particularly necessary on iterations 24 and 27 where the number of nonzero entries for the first factorization approaches six million. Notice from table 7.1 that the Cholesky factorization for the Schur complement of this problem gives
<table>
<thead>
<tr>
<th>IP Iteration</th>
<th>PCG</th>
<th>Nonzero Entries</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>First LC</td>
</tr>
<tr>
<td>0</td>
<td>901</td>
<td>109894</td>
</tr>
<tr>
<td>1</td>
<td>897</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>741</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3289</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>267</td>
<td>143739</td>
</tr>
<tr>
<td>5</td>
<td>279</td>
<td>114967</td>
</tr>
<tr>
<td>6</td>
<td>287</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>336</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>418</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>699</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>906</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1434</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>2194</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>277</td>
<td>231102</td>
</tr>
<tr>
<td>14</td>
<td>338</td>
<td>111702</td>
</tr>
<tr>
<td>15</td>
<td>373</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>404</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>367</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>538</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>585</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>753</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>1194</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>1235</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>1845</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>200</td>
<td>5901594</td>
</tr>
<tr>
<td>25</td>
<td>279</td>
<td>124445</td>
</tr>
<tr>
<td>26</td>
<td>610</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>211</td>
<td>5571610</td>
</tr>
<tr>
<td>28</td>
<td>232</td>
<td>168273</td>
</tr>
<tr>
<td>29</td>
<td>221</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>196</td>
<td></td>
</tr>
</tbody>
</table>

Table 7.4 DIFFICULT Factorization Heuristics
over six million nonzero entries.

This strategy works well here because the $LU^*$ factorization generates less nonzero entries than the corresponding Cholesky factorization and also because few factorizations are performed. However, there are problems where this strategy will not work well because the number of fill-in entries can be too large. For these problems, probably a better approach is to select the first $m$ columns from the reordering and compute an $LU^*$ factorization in the standard way. minimizing the number of fill-in entries. Then, the dependent columns are discarded and another factorization is computed with the independent columns plus other columns from the remaining columns on the original ordering to complete a square matrix. The process is repeated until a nonsingular matrix is found.

7.3 Conclusions

We have shown that from the point of view of designing preconditioners it is better to work with the augmented system instead of working with the Schur complement. Two important results support this statement. First, all preconditioners developed for the Schur complement system have in a sense an equivalent for the augmented system. However, the opposite statement is not true. Whole classes of preconditioners for the augmented system can lead to the an unique preconditioner for the Schur complement.

Given this result we decided to design a preconditioner for the augmented system. Our primary goal on developing this preconditioner was to avoid the Schur complement. The main reason to avoid it is the loss of sparsity that can happens by computing it or its factorization. Thus, we developed a new class of preconditioners which relies instead on the computation of an $LU^*$ factorization of a set of columns of
the matrix of constraints for the linear programming problem.

By choosing the blocks of the preconditioner carefully, we are able to partition the augmented system into two smaller systems. The first system of dimension \( m \) is a diagonal system thus it can be considered solved. The other system has dimension \( n \) and it is indefinite. We have shown that all of its eigenvalues are bounded away from zero. This system can be further partitioned into two smaller positive definite systems of dimension \( m \) and \( m - n \) respectively. The indefinite system is better conditioned than the other two systems but since they are smaller and positive definite it seems to be more efficient to solve them. We have shown the relation between the system of dimension \( m \) and the Schur complement. On the other hand, the indefinite system and the other positive definite system cannot be derived from the Schur complement.

This class of preconditioners behaves very well at the end of the interior point method. This is important because the linear systems became more ill-conditioned as the interior point methods approaches a solution. These systems are hard to solve by iterative methods with most of the previously known preconditioners. However, an efficient implementation of these preconditioners is not trivial.

We discussed how a careful implementation of this preconditioner made this approach competitive with the direct method approach. This is particularly true on classes of problems where the Cholesky factorization of the Schur complement matrix has a large number of nonzero entries. Computational experiments presented in this chapter support this statement.
Among the implementation details, we presented several techniques for computing the $LL^T$ factorization given an ordered set of columns. Some of these techniques, in particular finding symbolically dependent and independent columns, can be used in other problems of linear and nonlinear programming.

### 7.4 Future Work

Several points in this work can lead to future research. On the modeling point of view, it will be interesting to develop a variant of this class of preconditioners that deals with free variables explicitly.

Another important aspect is how to compute an intelligent initial guess for second linear system in the predictor-corrector variant. Since this linear system depends on the solution of the first system and both share the same matrix, it may be possible to save work on finding the solution for the second linear system.

From the point of view of improving the preconditioner, it will be interesting to develop a scaled version which may give better conditioned systems at the first iterations of the interior point method without compromising its convergence. This strategy will lead to a more robust preconditioner. We also are interested on finding variations of the preconditioner by choosing its blocks in a different fashion but still avoiding the Schur complement.

Another possibility for improvement is a more sophisticated way to decide when to keep the current $LL^T$ factorization. The savings of keeping a factorization can be so high that this approach deserves more attention. The computation of the first factorization can also be improved. The approach that reorder the columns to reduce
the amount of fill-in entries deserves attention. There are problems where this number can be considerably large. The ordering given by this approach is not the ideal ordering in view of the performance of the preconditioned system. Nonetheless, it is not an important drawback in comparison.

From the numerical point of view, we would like to make experiments with the hybrid approach, that is we start with an incomplete Cholesky factorization and switch to the $LU$ preconditioner at a certain point. If a suitable way to decide when to switch can be found, this approach will be probably the fastest way for solving many problems.

It will be interesting to run experiments with the other preconditioned linear systems developed on Chapter 5. The indefinite linear system and the $n - m$ dimensional positive definite linear system. Some experiments with explicit inequality constraints and relaxed tolerance are also in order. All experiments reported in this work use the square root of epsilon machine as tolerance. However, it is well known that at least on the first iterations of the interior point methods it is not necessary to find the directions with great precision. Thus, a relaxed precision will speed up solving the linear system and perhaps make the preconditioner more robust. Other interesting experiment is to try the incomplete factorization with several drop tolerances to determine how valid this approach is.

Finally, a more sophisticated $LU$ factorization which include at least strongly connected components and choice for the pivot for reducing the number of fill-in entries should improve the results significantly. These techniques are to be used on the second factorization when all the columns are known. Recall that this factorization is used
by the conjugate gradient method to solve the linear systems. For some problems the number of iterations for the preconditioned conjugate gradient can be as large as a few thousands. Therefore, the most efficient $LU$ factorization is necessary in order to take maximum advantage of the preconditioner.
Bibliography


