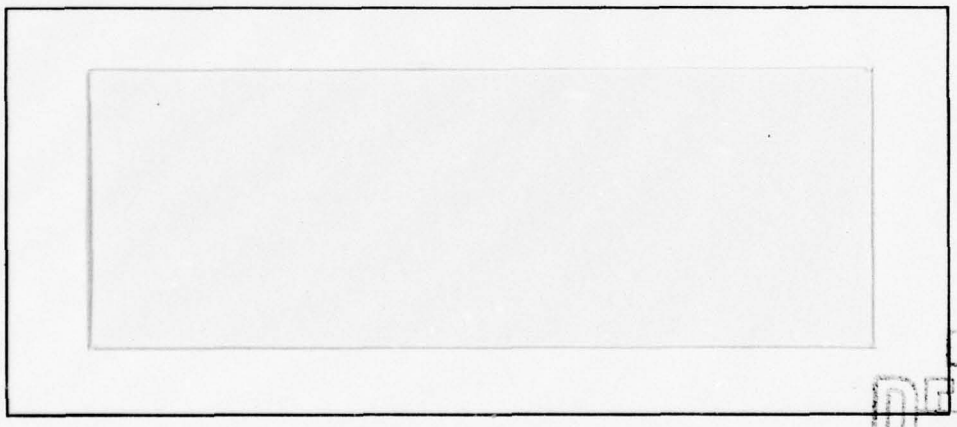
AD-A053 313 YALE UNIV NEW HAVEN CONN DEPT OF COMPUTER SCIENCE F/G 12/1
CONJUGATE GRADIENT METHODS FOR PARTIAL DIFFERENTIAL EQUATIONS.(U)
JAN 78 R CHANDRA N00014-76-C-0277
UNCLASSIFIED RR-129 NL

1 OF 2
AD A053313



YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

code 432
NR-044-401

(12)

This work was presented to the faculty of the Graduate School of Yale University in candidacy for the degree of Doctor of Philosophy.

(10) Rati Chandra

(6) Conjugate Gradient Methods for Partial Differential Equations

(14) RR-129

Rati Chandra

(9) Research Report, #129

(11) January 1978

(12) 136 pp.

(15) This work was supported in part by ONR Grant N00014-76-C-0277, NSF Grant MCS 76-11460, AFOSR Grant F49620-77-C-0037, and ERDA Grant SRSA-EG-77-S-02-4349.

407 051

ABSTRACT

CONJUGATE GRADIENT METHODS FOR
PARTIAL DIFFERENTIAL EQUATIONS

Rati Chandra
Yale University, 1978

The method of conjugate gradients is known as an efficient iterative method for solving large sparse symmetric positive definite systems of linear equations. Such systems arise frequently in scientific computing, as in the solution of self-adjoint elliptic partial differential equations using finite difference and finite element techniques.

In this dissertation, we define a family of variational iterative methods for solving symmetric linear systems that include the conjugate gradient method as its simplest case. These gradient type methods minimize different error-functionals over subspaces of increasing dimension and possess attractive convergence properties. We develop the general theory and computational aspects, and discuss a few important special cases.

We have three main objectives. The first is to investigate the use of preconditioning as a means to enhance the rate of convergence of these methods. Some of the techniques presented have not been used previously in conjunction with any of these methods. We are also interested in the effect of the preconditionings on the asymptotic behavior of the methods when applied, in particular, to the solution of linear systems arising from finite difference approximations to self-adjoint elliptic partial differential equations. We pick a simple model problem, Poisson's equation in a unit cube in p-dimensional space, and show that certain preconditioning techniques applied to the linear system corresponding to the discretized problem can improve the asymptotic rate of convergence by an order of magnitude.

The second objective is to improve the efficiency of the iteration procedures. When the coefficient matrix is two-cyclic, we present a version of the CG iteration which requires less work and storage than the Reid's version, which is currently considered to be the most efficient for this class of matrices. For the model problem in two dimensions, the improvement in work per iteration is about 20%.

The third objective is to enlarge the domain of applicability of the conjugate gradient method. When the coefficient matrix is symmetric but not positive definite, the conjugate gradient method may break down due to instabilities in the iteration process. We present a method called SYMMBK which overcomes this problem, and is more efficient than another known method, SYMMLQ, that does the same. These methods show good convergence properties, at least on simple examples, but no error bounds are available for them. We propose another efficient method, the modified conjugate residual (MCR) method, which belongs to the family of variational iterative methods mentioned earlier. This method minimizes the 2-norm of the residual at each iteration, and, for it, convergence results are easily obtained.

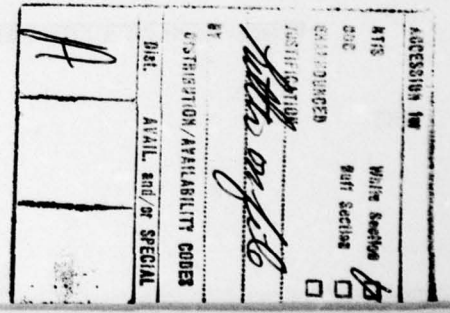


TABLE OF CONTENTS

PAGE

ABSTRACT..... ii

TABLE OF CONTENTS..... iv

TABLE OF ALGORITHMS..... vii

CHAPTER

PART I: THE SYMMETRIC PROBLEM

1. INTRODUCTION..... 1

2. PRELIMINARIES..... 12

2.1 Introduction..... 12
2.2 Definitions and Notation..... 12
2.3 The Model Problems..... 15
2.4 Numerical Experiments..... 21

3. VARIATIONAL ITERATIVE METHODS..... 24

3.1 Introduction..... 24
3.2 Definition and Properties..... 25
3.3 Error Bounds..... 31
3.4 Alternative Generation of Direction Vectors..... 35

CHAPTER PAGE

PART II: THE SYMMETRIC POSITIVE DEFINITE PROBLEM

4. THE CONJUGATE GRADIENT AND CONJUGATE RESIDUAL METHODS..... 39

4.1 Introduction..... 39
4.2 Error Bounds..... 40
4.3 Computational Aspects..... 43
4.4 The Conjugate Gradient (CG) Method..... 45
4.5 The Conjugate Residual (CR) Method..... 47
4.6 Numerical Results..... 49

5. THE PRECONDITIONED CONJUGATE GRADIENT AND PRECONDITIONED CONJUGATE RESIDUAL METHODS..... 55

5.1 Introduction..... 55
5.2 The Preconditioned Variational Methods..... 58
5.3 The Preconditioned Conjugate Gradient (PCG) Method..... 63
5.4 The Preconditioned Conjugate Residual (PCR) Method..... 65

6. SOME PRECONDITIONINGS..... 68

6.1 Introduction..... 68
6.2 Diagonal Scaling..... 71
6.3 Low Rank Remainder Preconditioning..... 73
6.4 Approximate Factorization Algorithms..... 73
6.4.1 Introduction..... 73
6.4.2 Meijerink and van der Vorst Factorization..... 78
6.4.3 Dupont, Kendall and Rachford Factorization..... 81
6.4.4 Stone Factorization..... 84
6.5 ADI-Preconditioning..... 86
6.6 Block-SSOR Preconditioning..... 88
6.6.1 Introduction..... 88
6.6.2 Computational Aspects..... 93
6.6.3 Preconditioning for Anisotropic Problems..... 100
6.7 Numerical Results..... 103

7. THE REDUCED PRECONDITIONED CONJUGATE GRADIENT METHOD FOR TWO-CYCLIC MATRICES..... 123

7.1 Introduction..... 123
7.2 The Reduced Preconditioned Conjugate Gradient (RPCG) Method..... 124
7.3 Comparison with Reid's Method..... 129
7.4 Two-Cyclic Preconditioning..... 137

CHAPTER PAGE

8. COMPARISON WITH OTHER METHODS 143

8.1 Introduction 143
8.2 Comparison with Iterative Methods 143
8.3 Comparison with Direct Methods 147

PART III: THE SYMMETRIC INDEFINITE PROBLEM

9. SOME METHODS FOR THE INDEFINITE PROBLEM 151

9.1 Introduction 151
9.2 Development of SYMMBK and SYMMLQ 153
9.3 The Bunch and Kaufman Decomposition 157
9.4 SYMMBK 161
9.5 SYMMLQ 172
9.6 Luenberger's Method of Hyperbolic Pairs 175
9.7 Numerical Results 188

10. THE LINEAR LEAST SQUARES PROBLEM 197
10.1 Introduction 197
10.2 The Conjugate Gradient Method for Normal Equations 200
10.3 Comparison of Methods 202

11. THE MODIFIED CONJUGATE RESIDUAL METHOD 213

11.1 Introduction 213
11.2 Error Bounds 214
11.3 Computational Aspects 219
11.4 The Modified Conjugate Residual (MCR) Method 225
11.5 The Preconditioned Modified Conjugate Residual Method. 228
11.6 Numerical Results 234

APPENDIX A 243
APPENDIX B 250
REFERENCES 253

TABLE OF ALGORITHMS

ALGORITHM PAGE

3.1 The Lanczos Method for Generation of Orthogonal Vectors... 26

3.2 The Variational Method 27

4.1 The Variational Method for Positive Definite Systems 45

4.2 The Conjugate Gradient (CG) Method 46

4.3 The Conjugate Residual (CR) Method 47

5.1 The Preconditioned Variational Method for Positive Definite Systems 59

5.2 The Preconditioned Conjugate Gradient (PCG) Method 64

5.3 The Preconditioned Conjugate Residual (PCR) Method 65

7.1 The Reduced Preconditioned Conjugate Gradient (RPCG) Method for Two-Cyclic Systems 127

7.2 Reid's Method 129

7.3 The Modified Reid's Method 134

7.4 The Reduced Conjugate Gradient Method with Two-Cyclic Preconditioning 140

9.1 The Lanczos Method for Generation of Orthonormal Vectors.. 154

9.2 The SYMMBK Method 166

9.3 Luenberger's Method of Hyperbolic Pairs 176

ALGORITHM		PAGE
10.1	The Conjugate Gradient Method for Normal Equations.....	200
11.1	The Variational Method for Indefinite Systems.....	224
11.2	The Modified Conjugate Residual (MCR) Method.....	226
11.3	The Preconditioned Variational Method for Indefinite Systems.....	229
11.4	The Preconditioned Modified Conjugate Residual Method.....	230

PART I

THE SYMMETRIC PROBLEM

CHAPTER 1

INTRODUCTION

In this dissertation, we consider the solution of linear systems

$$Ax = f, \tag{1.1}$$

where A is a large sparse nonsingular symmetric matrix. Such systems occur frequently in the world of scientific computing, and often their solution involves a large portion of the total computational effort required for the numerical calculations in which they arise.

Methods for solving linear systems can essentially be categorized as either direct or iterative. Except for roundoff errors, direct methods yield the exact solution of the system after a finite number of numerical operations. Iterative methods yield sequences of approximate solutions which approach the exact solution asymptotically. Historically, solving large sparse systems has been the domain of iterative methods.

Direct methods typically use some form of symmetric Gaussian elimination [32]. When the matrix A is positive definite, it is factored into the product

$$A = U^T DU, \tag{1.2}$$

where U is a unit upper triangular matrix and D is a positive diagonal matrix, and the solution is obtained by solving

$$U^T y = f, \quad Dz = y, \quad \text{and} \quad Ux = z.$$

Direct methods are well-suited for small dense problems. But, with the development of sparse storage schemes and efficient orderings for a large class of grid problems, direct methods are now competitive with iterative methods on large sparse problems.

The main difficulties in the solution of large sparse systems of linear equations are storage of the coefficient matrix and the computational effort for the solution process. Direct methods all involve some type of transformation of the coefficient matrix. For example, during the factorization of the matrix A as in (1.2), "fill-in" occurs, i.e., zeros in A get changed to nonzeros, and more storage is required for U than for A in its original form. Thus the distribution of the nonzeros in the matrix is of importance, and both the storage requirements as well as the computational effort are strongly affected by the ordering of the equations, i.e., by the sequence in which the

unknowns are eliminated.

Iterative methods are better suited than direct for the solution of large sparse systems since they require much less storage and little computational effort at each iteration. Moreover, the actual positions of the nonzeros in the coefficient matrix is not important for the efficiency of the iteration, since the matrix is not modified during the iteration process. Generally the matrix is only used to compute matrix-vector products. Thus it does not even have to be available in assembled form, and it may be more economical to regenerate the nonzeros instead of storing them (see [31]). Moreover, in some cases certain sub-blocks may repeat themselves leading to additional efficiencies in storage and work.

A comparison of direct and iterative methods in terms of the total work depends upon the desired accuracy of the approximate solution. Often, for example when the system is derived from physical experiments and may contain experimental error, the solution is not desired to great accuracy, and a relatively small number of iterations may suffice. However, if many systems with the same coefficient matrix are to be solved, direct methods should probably be used since the most expensive part of the solution procedure, the factorization, has to be done only once.

A difficulty in using iterative methods is that the behavior of the iteration process is dependent upon the initial approximation to the solution. However, this may be an advantage in problems for which a good initial estimate is available. For example, implicit finite-difference approximations to time-dependent problems involve a system at each time step and the solution at one time step provides a good starting guess for the solution at the next time step.

Another difficulty attributed to iterative methods is that there is no way of knowing the error at any stage in the iteration process. This can cause either premature termination or additional work even though the desired accuracy has been obtained. However, the residual vector usually serves as a fairly efficient error predictor.

In this dissertation, we investigate a class of variational iterative methods including the conjugate gradient (CG) method, a powerful iterative method, for solving symmetric positive definite systems of linear equations. In 1952, Hestenes and Stiefel [39] proposed the CG method as a direct method which would, in the absence of round-off error, yield the exact solution of an N x N system in at most N steps. But even for small systems (say N < 100), roundoff error can seriously contaminate the approximation. Thus the CG method never became popular as a direct method. Its addition to the ranks of popular iterative methods is comparatively recent.

In 1959, Engeli, Ginsburg, Rutishauser, and Stiefel [27] gave extensive results on the performance of some gradient methods and compared them with other iterative methods such as SOR. Their conclusion was that gradient methods, particularly the CG method (as an iterative method), should be used when nothing is known about the eigenvalues of the coefficient matrix, in which case the iteration parameters required by methods like Chebyshev acceleration and successive overrelaxation are not known a priori and have to be estimated. Moreover, for ill-conditioned problems, overrelaxation methods were found to be extremely sensitive to the iteration parameter. This work was reinforced by a paper in 1970 by Reid [53]. In numerical experiments, he showed that the CG method compared favorably with methods like Chebyshev acceleration and SOR for large sparse systems. Subsequently, the CG method has been recognized as a powerful method for solving large sparse linear systems of equations.

Experience with the CG method has shown it to be a remarkably practical method which is efficient and easy to implement. In 1973, Kostem and Schultchen [42] presented extensive numerical tests on some of the positive definite linear systems arising in finite element analysis. Their results show the CG method to be the best iterative method for this class of problems. Reid [54] has shown that if the coefficient matrix is two-cyclic, then the work required for the CG method may be approximately halved with a small saving in storage. In

Chapter 7, we propose an even more efficient method for two-cyclic systems. Concus and Golub [13] have proposed a generalized CG method for solving nonsymmetric linear systems in which the symmetric part of the coefficient matrix corresponds to an easily solvable system of equations. Several authors (e.g., Wachspress [61], O'Leary [49]) have used the CG method in combination with other iterative methods.

Evans [28] proposed a preconditioning technique for increasing the convergence rate of the CG method for any symmetric positive definite system. In Chapter 6, we have extended Evans's idea to block-preconditioning, which gives faster convergence on a variety of problems. Meijerink and van der Vorst [48] proposed another preconditioning for linear systems arising from the discretization of elliptic differential equations. A factorization procedure due to Dupont, Kendall, and Rachford [20] is particularly simple to implement and gives remarkably good results for these problems.

One of the important reasons for the popularity of the CG method as a practical method is that it does not require estimation of parameters. It is well known that SOR relies heavily on a good choice of an iteration parameter ω, which, if not known (as in most real problems), requires considerable effort to compute [62]. Other iterative methods like ADI and Chebyshev iteration also require estimation of parameters that have to be determined a priori. Furthermore, the CG method requires fewer restrictions on the coefficient matrix for optimal

behavior than methods like SOR. A final attractive feature is that the 2-norm of the error vector decreases monotonically at each iteration [39].

Some of these attractive properties of the CG method carry over to a more general class of methods. In this dissertation, we define a family of variational iterative methods for solving symmetric linear systems that include the CG method as its simplest case. These variational methods minimize different error-functionals over subspaces of increasing dimension and possess good convergence properties. We develop the general theory and computational aspects, and discuss a few important special cases.

We have three main objectives. The first is to extend the domain of applicability of the CG type of method, the second is to investigate the use of preconditioning as a means to enhance the rate of convergence of the variational methods, and the third is to attempt to improve the efficiency of the iteration procedures.

We will use two examples to illustrate our results. The first, perhaps the simplest nontrivial example of large sparse systems in the world of scientific computing, arises from the application of the simplest finite difference approximations to second order self-adjoint elliptic partial differential equations. We call this the "generalized model problem". The second example is a special case of the generalized

model problem. It arises from the simplest finite difference approximations to Poisson's equation in a p-dimensional cube and has a very special structure. There are fast direct methods ([3], [17]) that exploit this special structure and solve the linear systems more efficiently than we can using the methods in this dissertation. However, the results that we get using these problems are applicable to less specialized cases for which special purpose methods are not available.

This dissertation is divided into three parts. The three parts correspond to the cases in which A is symmetric, symmetric positive definite, and symmetric indefinite, respectively.

In Chapter 2, we deal with preliminaries like notation and definitions, precisely define the generalized model and the model problems, and give a brief description of the procedure adopted for numerical experiments that are spread over the rest of the dissertation. In Chapter 3, we describe a class of variational iterative methods for solving symmetric linear systems. Moreover, we derive some important properties of these methods and obtain general error bounds.

Part II of this dissertation considers the application of these methods to symmetric positive definite problems. In Chapter 4, we give a computational version of these methods and discuss two important special cases - the Conjugate Gradient (CG) and the Conjugate Residual

(CR) methods. In Chapter 5, we motivate and apply the idea of using preconditioning to accelerate the rate of convergence of these methods.

In Chapter 6, we describe some preconditioning techniques with special reference to the generalized model problem, and give theoretical and numerical results comparing them for the model problem. Some of these techniques are derived from other iterative methods like SSOR [62] and ADI [60], others from the approximate factorization schemes due to Meijerink and van der Vorst [48], Dupont, Kendall, and Rachford [20], and Stone [6]. We prove that for the model problem certain preconditioning techniques can improve the asymptotic rate of convergence by an order of magnitude. Our numerical results show that preconditioning can be extremely effective in improving the convergence and reducing the cost of the iteration procedures. We also investigate the dependence of the preconditioned methods on parameters.

In Chapter 7, we propose a preconditioning which can be used with the CG method in the case when A is two-cyclic. We compare the resulting method, referred to as the Reduced Preconditioned Conjugate Gradient (RPCG) method, with a method due to Reid [54] which is known to be efficient for solving the two-cyclic problem. We show that RPCG is more effective with respect to work requirements and does not require additional storage. Based on the RPCG method, we define "two-cyclic preconditioning", which can be applied to any system with a symmetric positive definite matrix. In Chapter 8, we give a comparison of the CG

and PCG methods with some competing direct and iterative methods.

The CG and CR methods may break down if applied directly to problems that are symmetric indefinite. In Part III we give ways of extending these methods to symmetric indefinite problems. In Chapter 9 we present a new method, SYMMBK, based on a factorization procedure of Bunch and Kaufman [8], and two existing methods, SYMMLQ [49] and Luenberger's method of hyperbolic pairs [46]. Essentially, they all try to apply CG to the indefinite problem and use different techniques to avoid the numerical instability that may result. We show that, assuming exact arithmetic, these methods are mathematically equivalent. Since Luenberger's method is unstable in the presence of roundoff, and the SYMMLQ iteration is more expensive than the SYMMBK iteration, we recommend the latter.

It has been suggested, eg., [46], [51], that these methods can be used to efficiently solve the linear least squares problem. In Chapter 10, we show that using any of the above methods to solve this problem to any desired accuracy is less efficient than using the CG method on the normal equations, assuming exact arithmetic.

In Chapter 11, we consider the application of the variational methods of Chapter 3 to the symmetric indefinite problem and obtain error bounds similar to those for the positive definite problem. We derive the modified conjugate residual (MCR) method as the simplest

special case of this class of methods. MCR minimizes the 2-norm of the residual over subspaces of increasing dimension. It can be viewed as an extension of the CR method to the symmetric indefinite problem, and also as a stabilized version of the method of Luenberger [45]. We give results of numerical experiments comparing MCR with the methods in Chapter 9. These results show MCR to be an extremely powerful and efficient method for the indefinite problem.

CHAPTER 2

PRELIMINARIES

2.1: Introduction

As far as possible, standard mathematical notation has been used throughout. In Section 2, we introduce some additional notation and concepts, the meaning of which may not be obvious from the context. In Section 3, we define two model problems which will be referred to repeatedly in later chapters, and in Section 4, we describe the numerical experiments that were carried out to illustrate our theoretical results and conjectures.

2.2: Definitions and Notation

Denote the spectral radius of a matrix A by $\rho(A)$.

For any matrix A, its condition number is given by

$$\kappa(A) = \|A\| \ \|A^{-1}\|$$

where $\|.\|$ denotes the spectral norm. Note that $\kappa(A) \geq 1$ for any matrix A. Furthermore, if A is symmetric with eigenvalues λ_i, then

$$\kappa(A) \equiv (\max_i |\lambda_i|) \,/\, (\min_i |\lambda_i|).$$

For a symmetric positive definite matrix A, we define the A-norm of a vector v by

$$\|v\|_A \equiv (v, Av)^{1/2}.$$

We now give some definitions regarding iterative methods [62]. In general, an iterative method for solving a linear system $Ax = f$ can be written as

$$x_0 = \phi_0(A,f)$$
$$x_1 = \phi_1(x_0; A,f)$$
$$\cdot$$
$$\cdot$$
$$x_{i+1} = \phi_{i+1}(x_0,x_1,\ldots,x_i; A,f)$$

where $x_0,x_1,\ldots$ is the sequence of iterates and $\phi_0(A,f),\phi_1(x_0; A,f),\ldots$ are functions.

The iterative method is linear if for each i, ϕ_i is a linear function of $x_0,x_1,\ldots,x_{i-1}$. The iterative method is stationary if for some integer $s > 0$, ϕ_i is independent of i for all $i \geq s$. The degree of a stationary method is d for $d \leq s$, if, for $i \geq s-1$, x_{i+1} depends on $x_i,x_{i-1},\ldots,x_{i-d+1}$ but not on x_k for $k < i-d+1$. An iterative method is

consistent if the following holds: if for some i, x_i is a solution, say $\bar{x}$ of $Ax = f$, then $x_{i+1} = x_{i+2} = \ldots = \bar{x}$.

Now let x_m be the approximation to the solution x obtained at the m^{th} iteration of an iterative method, and let B be a symmetric positive definite matrix. Then the relative B-norm of the error at the m^{th} iteration is defined as

$$e_m \equiv \frac{\|x_m - x\|_B}{\|x_0 - x\|_B}.$$

The average rate of convergence of the iterative method is defined by

$$R_m \equiv -\frac{1}{m} \log e_m$$

and the (asymptotic) rate of convergence by

$$R \equiv \lim_{m\to\infty} R_m.$$

For the gradient methods presented in this dissertation, the error bounds obtained are of the form

$$e_m \leq C\beta^m,$$

where $C > 0$ is a constant independent of m. We slightly abuse the above definitions and refer to

$$\lim_{m\to\infty} \left(-\frac{1}{m} \log(C\beta^m)\right)$$

as the rate of convergence. Therefore, in this context,

$$R \equiv \lim_{m\to\infty} \left(-\frac{1}{m}\log(C\beta^m)\right) = -\log \beta.$$

A linear stationary iterative method of the first degree has an iteration of the form

$$x_{m+1} = G\,x_m + k$$

for some matrix G and vector k [62]. Then, for this method, $e_m \le \|G^m\|_2$. Moreover, if $\rho(G) < 1$, $\lim_{m\to\infty} (\|G^m\|_2)^{1/m} = \rho(G)$. Thus, for this method, the rate of convergence with respect to the 2-norm is given by

$$R = -\log(\rho(G)).$$

2.3: The Model Problems

Consider the self-adjoint elliptic boundary-value problem

$$L(u) \equiv \sum_{i=1}^{p} \frac{\partial}{\partial x_i}\left(\alpha_i(x)\frac{\partial}{\partial x_i}u\right) + \sigma(x)u = \phi(x), \quad x \in D,$$

$$u = \gamma(x), \quad x \in \partial D, \tag{2.1}$$

where D is the interior of the unit cube in p dimensions and ∂D denotes the boundary of D. We assume that the coefficients α_i, ϕ, and σ are smooth functions with $\alpha_i(x) \ge m$ for some positive constant m.

To solve (2.1), we cover the domain with a mesh, uniform and equal in all p directions, with mesh-width $h \equiv 1/(n+1)$, and seek a function $U(i_1, i_2, \ldots, i_p)$ which is an approximation to $u(i_1h, i_2h, \ldots, i_ph)$ in D. We replace the differential equation by a finite difference analog. The discretized version of the problem (assuming natural ordering of the unknowns) will be referred to as the <u>generalized model problem in p dimensions</u>. We will be mainly interested in problems in two and three dimensions.

In <u>two dimensions</u> we replace the differential operators in (2.1) by

$$\frac{\partial}{\partial x_1}\left(\alpha_1(x)\frac{\partial}{\partial x_1}u\right) \sim h^{-2}\left[\alpha_1\left(x_1+\tfrac{h}{2}, x_2\right)\{u(x_1+h, x_2) - u(x_1, x_2)\} - \alpha_1\left(x_1-\tfrac{h}{2}, x_2\right)\{u(x_1, x_2) - u(x_1-h, x_2)\}\right] \tag{2.2a}$$

and

$$\frac{\partial}{\partial x_2}\left(\alpha_2(x)\frac{\partial}{\partial x_2}u\right) \sim h^{-2}\left[\alpha_2\left(x_1, x_2+\tfrac{h}{2}\right)\{u(x_1, x_2+h) - u(x_1, x_2)\} - \alpha_2\left(x_1, x_2-\tfrac{h}{2}\right)\{u(x_1, x_2) - u(x_1, x_2-h)\}\right] \tag{2.2b}$$

Substituting in the differential equation, we note that the differential operator L has been replaced at every point (i_1h, i_2h) in D by a linear combination of the values of u at (i_1h, i_2h) and four adjacent points:

$$a_0\,U(i_1, i_2) - a_1U(i_1+1, i_2) - a_2U(i_1, i_2+1) - a_3U(i_1-1, i_2) - a_4U(i_1, i_2-1) = -h^2\,\phi(i_1, i_2), \quad (i_1h, i_2h) \in D, \tag{2.3}$$

where

$$\phi(i_1, i_2) = \phi(i_1h, i_2h),$$

$$U(i_1, i_2) = \gamma(i_1h, i_2h) \quad \text{for } (i_1h, i_2h) \in \partial D,$$

$$a_1 = \alpha_1(x_1 + h/2, x_2),$$

$$a_2 = \alpha_2(x_1,\ x_2 + h/2),$$

$$a_3 = \alpha_1(x_1 - h/2,\ x_2),$$

$$a_4 = \alpha_2(x_1,\ x_2 - h/2),$$

and $$a_0 = a_1 + a_2 + a_3 + a_4 - \sigma(x_1,\ x_2)h^2.$$

The problem then reduces to solving an N x N linear system

$$Au = f \tag{2.4}$$

where $N = n^2$ is the number of mesh-points in D and u is the vector of unknowns $U(i_1,\ i_2)$ in some order. Terms of (2.3) which refer to points on ∂D are brought to the right hand side since their value is known. Together with the vector $-h^2\phi(i_1,\ i_2)$, they form the vector f. A is symmetric [62], and each row of A contains at most five non-zero entries.

We will assume that the mesh-points are ordered using the "natural ordering," i.e., row by row from left to right, starting at the bottom row (see Figure 2.1).

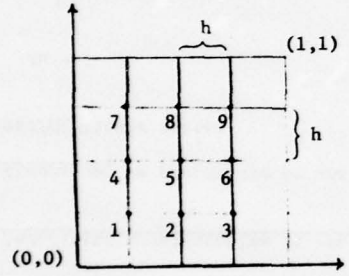


Figure 2.1: Natural ordering for points on a 3 x 3 mesh on the unit square (n = 3, N = 9, h = 1/4)

The resulting matrix A is block-tridiagonal,

$$A \equiv \begin{bmatrix} T_1 & B_1 & & & \\ B_1 & \ddots & \ddots & & \\ & \ddots & \ddots & B_{n-1} & \\ & & B_{n-1} & T_n \end{bmatrix}, \tag{2.5}$$

where the B_i's are n x n diagonal matrices and the T_i's are n x n tridiagonal matrices. The system (2.4) with this definition of A defines the generalized model problem in two dimensions. If $\sigma(x) \leq 0$ in D, then A is positive definite. Otherwise A may be indefinite.

If we take $\alpha_1 = \alpha_2 = 1$, σ = constant, and $\gamma = 0$ in (2.1), we obtain the model problem in two dimensions. In that case the B_i's and T_i's in (2.5) are given by

$$B_i = -I, \qquad 1 \leq i \leq n-1,$$

and

$$T_i = \begin{bmatrix} 4-\sigma h^2 & -1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 4-\sigma h^2 \end{bmatrix}, \quad 1 \leq i \leq n,$$

where I is the identity matrix. For $\sigma = 0$, A is positive-definite and we obtain the symmetric positive-definite model problem in two dimensions. For σ sufficiently positive, A is indefinite and we obtain

the symmetric indefinite model problem in two dimensions.

In three dimensions we proceed in an analogous way. The matrix A of the corresponding linear system

(2.6) $\qquad Au = f$

is an N x N ($N = n^3$) symmetric matrix with at most seven non-zero entries in any row. We use the "natural ordering" of the unknowns: i.e., we start at one end-face of the mesh, number the points on the face as in two dimensions and then proceed to the next adjacent face. The resulting matrix has its non-zeroes restricted to seven diagonals at distances zero (the main diagonal), 1, n+1, and n^2+1 from the main diagonal. It is block-tridiagonal,

$$A \equiv \begin{bmatrix} D_1 & C_1 & & & \\ C_1 & \ddots & \ddots & & \\ & \ddots & \ddots & C_{n-1} & \\ & & C_{n-1} & D_n \end{bmatrix},$$

where the D_i's are n^2 x n^2 matrices of the form

$$D_i \equiv \begin{bmatrix} E_{i,1} & F_{i,1} & & & \\ F_{i,1} & \ddots & \ddots & & \\ & \ddots & \ddots & F_{i,n-1} & \\ & & F_{i,n-1} & E_{i,n} \end{bmatrix}, \qquad i = 1, 2, \ldots n,$$

the $E_{i,j}$'s are n x n tridiagonal matrices, the $F_{i,j}$'s are n x n diagonal matrices, and the C_i's are n^2 x n^2 diagonal matrices. If $\sigma(x) \leq 0$ in D, then A is positive definite and we obtain the positive definite generalized model problem in three dimensions. For ν sufficiently positive, we obtain the symmetric indefinite generalized model problem in three dimensions.

For the three-dimensional model problem, we take $\alpha_1 = \alpha_2 = \alpha_3 = 1$, σ = constant, and $\gamma = 0$ in (2.1). Then

$$C_i = -I, \qquad 1 \leq i \leq n-1,$$

$$F_{i,j} = -I, \qquad 1 \leq i \leq n, \quad 1 \leq j \leq n-1,$$

and

$$E_{i,j} = \begin{bmatrix} 6-\sigma h^2 & -1 & & & \\ -1 & \ddots & \ddots & & \\ & \ddots & \ddots & -1 & \\ & & -1 & 6-\sigma h^2 \end{bmatrix}, \qquad 1 \leq i,j \leq n,$$

where the I's denote identity matrices of suitable order. For $\sigma = 0$, we obtain the symmetric positive definite model problem in three dimensions, and for σ sufficiently positive, we obtain the symmetric indefinite model problem in three dimensions.

For the positive definite model problem, the condition number of A is $\kappa(A) = O(n^2)$ [62]. For the indefinite model problem, the eigenvalues of A are given by

$$\lambda_i - \sigma h^2, \qquad i = 1, 2, \ldots N,$$

where $\{\lambda_i\}_{i=1}^{N}$ are the eigenvalues of the matrix corresponding to the positive definite model problem. If a few, say m, eigenvalues of the indefinite matrix are negative and the rest lie in an interval [a,b], where a, b > 0, then

$$\frac{b}{a} = \frac{\lambda_N - \sigma h^2}{\lambda_{m+1} - \sigma h^2} = O(n^2).$$

2.4: Numerical Experiments

We carried out numerical tests to demonstrate and compare the efficiency of different methods. To make the differences in performance more obvious, we chose fine meshes. All computations (unless otherwise mentioned) were in single precision on a PDP-10 (word-length 36 bits).

For a given N x N test matrix A, we set up the system Ax = f by choosing the solution vector x to be a vector of N random numbers and computing the right-hand side f. In certain tests we wanted to compare the performance of an iterative method for the same problem over a sequence of increasingly fine meshes. In such cases we took the solution to be

(2.7a) $\quad u(x,y) = 3e^x e^y (x - x^2)(y - y^2)$

in two dimensions and

(2.7b) $\quad u(x,y,z) = 3e^x e^y e^z (x - x^2)(y - y^2)(z - z^2)$

in three dimensions.

Iterative methods characteristically require an initial guess x_0 to the solution. However, since we did not want to select an x_0 that would bias our tests by enhancing the performance of certain solution procedures, we always chose x_0 to be the zero vector.

To compare different solution procedures, we started each iteration process with the same initial guess x_0, and at each iteration computed the relative 2-norm of the error, $\frac{\|x - x_i\|_2}{\|x - x_0\|_2}$. For the test examples, x was known and so the error could be computed easily. We stopped iterating when the error was reduced below 10^{-6}. We then used two measures, the number of iterations and the number of multiplications (including divisions and square roots), to compare the procedures. In counting the number of multiplications, we ignored multiplications by +1 and -1. Thus for the model problem, in which the off-diagonal non-zero elements of A are -1's, computing a matrix-vector product took only N multiplications.

Since A is symmetric, we need store only the non-zeroes of the upper triangular part of A. For the model problem, we assume that A is "known" so that no storage is required for it. However, for the generalized model problem, we do not assume any special values for the non-zero elements. Thus for example, for the generalized model problem in two dimensions, $N + 2(N - n)$ locations are required to store A and $N + 4(N - n)$ multiplications are required to compute a matrix-vector

product. If A is positive definite, we usually use a diagonal scaling (discussed in Section 6.2) to make the diagonal terms unity. Thus for the symmetric positive definite generalized model problem in two dimensions, a matrix vector product requires 4(N - n) multiplications and 2(N - n) locations are sufficient to store A.

The storage requirements of the methods considered in this dissertation are essentially limited to the non-zeroes in the coefficient matrix and a few additional N-vectors. The storage required by various methods differs by only a few vectors. Hence, though we note the storage required for each procedure, we do not make it an important basis for comparing them. Often it happens that there are alternative ways of computing a quantity, the trade-off being between work and storage. At such times, we make the choice that minimizes the computational effort.

CHAPTER 3

VARIATIONAL ITERATIVE METHODS

3.1: Introduction

Consider the linear system of equations

$$Ax = f, \tag{3.1}$$

where A is an N x N symmetric matrix. Let μ be a positive integer constant, where we restrict μ to be even if A is indefinite. Then the error functional

$$E_\mu(\tilde{x}) \equiv (x-\tilde{x}, A^\mu(x-\tilde{x})) \tag{3.2}$$

is the square of the A^μ-norm of $x-\tilde{x}$ and attains its unique minimum when $\tilde{x} = x$. Thus solving (3.1) is equivalent to minimizing (3.2).

In this chapter, we present a class of iterative methods for the solution of (3.1). These methods minimize the square of the A^μ-norm of the error (or equivalently, the error functional (3.2)) over subspaces of increasing dimension. These methods are optimal with respect to the

error functional, among all linear iterative methods. Corresponding to $\mu = 1,2,3,\ldots$ (where μ is even if A is indefinite), we get the different methods belonging to this class. The Conjugate Gradient [39], the Conjugate Residual [38], and the Modified Conjugate Residual [9] methods belong to this class.

In Section 2, we introduce the methods and derive their properties, and, in Section 3, we obtain general error bounds. In Section 4, we show that it may be possible to reduce the work involved in some iterations of the iteration process. We will discuss the application of these methods to solve (3.1) when A is positive definite, in Chapter 4, and, when A is indefinite, in Chapter 10.

3.2: Definition and Properties

These methods minimize $E_\mu(\tilde{x})$ over subspaces of increasing dimension. To approximate x, they compute a sequence of iterates $x_0,x_1,x_2,\ldots$ by moving along a sequence of directions $p_0,p_1,p_2,\ldots$, minimizing $E_\mu(\tilde{x})$ along each successive direction. More precisely, at the i^{th} step, x_{i+1} is computed as $x_{i+1} = x_i + a_i p_i$, where a_i is chosen to minimize $E_\mu(x_{i+1})$. The direction vectors are chosen by a method based on the Lanczos algorithm [50]. Due to this special way of choosiing the direction vectors, the uni-directional minimizations correspond to minimization in the whole subspace spanned by the p's, so that the x_{i+1} obtained actually minimizes the error functional on the affine subspace

$x_0 + \{p_0,p_1,p_2,\ldots,p_i\}$.

Algorithm 3.1: The Lanczos Method for Generation of Orthogonal Vectors

Step 1: Define $p_{-1} = \underline{0}$.

Choose p_0.

Step 2: Compute

$$\delta_i = (Ap_i,A^\mu p_{i-1})/(p_{i-1},A^\mu p_{i-1}),$$
$$\gamma_i = (Ap_i,A^\mu p_i)/(p_i,A^\mu p_i),$$

and

$$p_{i+1} = Ap_i - \gamma_i p_i - \delta_i p_{i-1}. \tag{3.3}$$

Step 3: Set i = i + 1 and go to Step 2.

Theorem 3.1: The direction vectors are A^μ-orthogonal, i.e.,

$$(p_i,A^\mu p_j) = 0,\ i \neq j.$$

Proof: Since A^μ is symmetric, it suffices to prove that

$$(p_i,A^\mu p_j) = 0,\quad i > j. \tag{3.4}$$

The proof is by induction on i. Since by Algorithm 3.1,

$$(p_1,A^\mu p_0) = (Ap_0,A^\mu p_0) - \gamma_0(p_0,A^\mu p_0) = 0,$$

(3.4) holds for i = 1. Assume that it holds for $i \leq k$.

Then

$$(p_{k+1},A^\mu p_j) = (Ap_k,A^\mu p_j) - \gamma_k(p_k,A^\mu p_j) - \delta_k(p_{k-1},A^\mu p_j). \tag{3.5}$$

By Algorithm 3.1 and the induction hypothesis,

$(p_{k+1}, A^{\mu} p_j) = 0$ for $j = k, k-1$.

Again using the induction hypothesis, the last two terms on the right-hand side of (3.5) vanish for $j < k-1$, and

$$\begin{aligned}(p_{k+1}, A^{\mu} p_j) &= (Ap_k, A^{\mu-1}(Ap_j)) \\ &= (Ap_k, A^{\mu-1}(p_{j+1} + \gamma_j p_j + \delta_j p_{j-1})) \\ &= (p_k, A^{\mu}(p_{j+1} + \gamma_j p_j + \delta_j p_{j-1})) \\ &= 0 \quad \text{for } j < k-1.\end{aligned}$$

Q.E.D.

Using Algorithm 3.1 for generating the direction vectors, we now consider the following gradient method for computing the approximations x_i to x.

Algorithm 3.2: The Variational Method

Step 1: Choose an initial approximation x_0 to x.

Compute $r_0 = f - Ax_0$.

Set $p_0 = r_0$

and $i = 0$.

Step 2: Compute

$$a_i = (r_i, A^{\mu-1} p_i)/(p_i, A^{\mu} p_i),$$

$$x_{i+1} = x_i + a_i p_i,$$

$$r_{i+1} = r_i - a_i A p_i,$$

$$\delta_i = (Ap_i, A^{\mu} p_{i-1})/(p_{i-1}, A^{\mu} p_{i-1}),$$

$$\gamma_i = (Ap_i, A^{\mu} p_i)/(p_i, A^{\mu} p_i),$$

and $p_{i+1} = Ap_i - \gamma_i p_i - \delta_i p_{i-1}$.

Step 3: If x_{i+1} is sufficiently close to x, terminate the iteration process;

else set $i = i + 1$ and go to Step 2.

Theorem 3.2: The following relations hold for the variational method:

(3.6a) $Ap_i \in \{p_0, p_1, \ldots, p_{i+1}\}$;

(3.6b) $r_i \in \{p_0, p_1, \ldots, p_i\}$;

(3.6c) $\{p_0, p_1, \ldots, p_i\} = \{p_0, Ap_0, \ldots, A^i p_0\} = \{r_0, Ar_0, \ldots, A^i r_0\}$;

(3.6d) $(r_i, A^{\mu-1} p_j) = 0, \quad j < i$;

(3.6e) $(r_i, A^{\mu-1} r_j) = 0, \quad i \neq j$;

(3.6f) $(r_i, A^{\mu-1} p_j) = (r_0, A^{\mu-1} p_j), \quad i \leq j$;

(3.6g) if $r_i \neq 0$, then $p_i \neq 0$.

Proof: The inclusion (3.6a) follows directly from (3.3). Next we prove (3.6b) - (3.6d) by induction. Since $p_0 = r_0$, (3.6b) - (3.6d) hold for $i = 0$. Assume that they hold for $i \leq k$.

Since $r_{k+1} = r_k - a_k A p_k$ and $Ap_k \in \{p_0, p_1, \ldots, p_{k+1}\}$ by (3.6a), we have that $r_{k+1} \in \{p_0, p_1, \ldots, p_{k+1}\}$. Thus (3.6b) holds for $i = k+1$.

By Algorithm 3.2 and the induction hypothesis, $p_{k+1} \in \{p_0, Ap_0, \ldots, A^{k+1} p_0\}$. Again using the induction hypothesis, this implies that

$$\{p_0, p_1, \ldots, p_{k+1}\} \subseteq \{p_0, Ap_0, \ldots, A^{k+1} p_0\}.$$

Because the p's are A^{μ}-orthogonal, they are linearly independent

and hence

$$\{p_0,p_1,\ldots,p_{k+1}\} = \{p_0,Ap_0,\ldots,A^{k+1}p_0\}.$$

Moreover,

$$\{p_0,Ap_0,\ldots,A^{k+1}p_0\} = \{r_0,Ar_0,\ldots,A^{k+1}r_0\},$$

since $p_0 = r_0$. Thus (3.6c) holds for $i = k+1$.

By the definition of r_{k+1} in Algorithm 3.2,

$$(r_{k+1}, A^{\mu-1}p_j) = (r_k, A^{\mu-1}p_j) - a_k(Ap_k, A^{\mu-1}p_j).$$

If $j = k$, then $(r_{k+1}, A^{\mu-1}p_j) = 0$ by the definition of a_k. If $j < k$, then the two terms on the right-hand side are zero by the induction hypothesis and Theorem 3.1. Thus (3.6d) holds for $i = k+1$, and (3.6b) - (3.6d) follow by induction.

To prove (3.6e), note that by (3.6b),

$$A^{\mu-1}r_j \in \{A^{\mu-1}p_0, A^{\mu-1}p_1,\ldots, A^{\mu-1}p_j\}.$$

The identity (3.6d) then implies that

$$(r_i, A^{\mu-1}r_j) = 0, \quad j < i,$$

and (3.6e) follows from the symmetry of A.

The identity (3.6f) holds since

$$r_i = r_0 - \sum_{k=0}^{i-1} a_k Ap_k$$

and Ap_k is orthogonal to $A^{\mu-1}p_j$, $j \neq i$, by Theorem 3.1.

To prove (3.6g), assume that $r_i \neq \underline{0}$ but $p_i = \underline{0}$. Then by (3.6a) and (3.6b),

$$r_i \in \{p_0,p_1,\ldots,p_{i-1}\}$$

and

$$Ar_i \in \{Ap_0,Ap_1,\ldots,Ap_{i-1}\} \subseteq \{p_0,p_1,\ldots,p_{i-1}\}.$$

Hence by (3.6d),

$$(A^{\mu}r_i,r_i) = (A^{\mu-1}r_i,Ar_i) = 0$$

and $r_i = \underline{0}$, which is a contradiction.

Q.E.D.

We also have the following result:

Theorem 3.3: For each i, x_{i+1} minimizes $E_\mu(\tilde{x})$ over the affine subspace $x_0 + \{p_0,p_1,\ldots,p_i\}$.

Proof: Let $\tilde{x} = x_0 + \sum_{j=0}^{i} s_j p_j$, where $\{s_j\}_{j=0}^{i}$ are scalars. Then

$$E_\mu(\tilde{x}) = (x - x_0 - \sum_{j=0}^{i} s_j p_j \ , \ A^{\mu}(x - x_0 - \sum_{j=0}^{i} s_j p_j))$$

$$= (r_0 - \sum_{j=0}^{i} s_j Ap_j \ , \ A^{\mu-2}(r_0 - \sum_{j=0}^{i} s_j Ap_j)).$$

Since the p's are A^{μ}-orthogonal,

$$E_\mu(\tilde{x}) = E(x_0) - \sum_{j=0}^{i} [2s_j(Ap_j, A^{\mu-2}r_0) - s_j^2(p_j,A^{\mu}p_j)].$$

The necessary and sufficient conditions for a minimum are that

$$(Ap_j, A^{\mu-2}r_0) - s_j(p_j,A^{\mu}p_j) = 0, \quad j=0,1,\ldots,i.$$

Hence by (3.6f),

$$s_j = (r_0, A^{\mu-1}p_j)/(p_j,A^{\mu}p_j) = (r_j, A^{\mu-1}p_j)/(p_j,A^{\mu}p_j) = a_j,$$

and the minimum point is

$$\tilde{x} = x_0 + \sum_{j=0}^{i} a_j p_j \equiv x_{i+1}.$$

Q.E.D.

3.3: Error Bounds

The fundamental minimization property of Theorem 3.3 allows us to prove that these methods are "optimal" among all linear iterative methods.

Using property (3.6c), we can write

$$x_i = x_0 + \sum_{j=0}^{i-1} s_j A^j r_0$$

$$= x_0 + P_{i-1}(A)\, r_0, \tag{3.7}$$

where $\{s_j\}_{j=0}^{i-1}$ are scalars and $P_{i-1}(A)$ is a polynomial of degree at most i-1 in A. Any consistent linear iterative method can be written in the form (3.7) [16]. Thus, since the methods defined in the preceding section minimize E_μ over all iterates of the form (3.7), we have the following result.

Theorem 3.4: For $k \geq 0$, the k^{th} approximation to x generated by the variational method coincides with that generated by a consistent linear iterative method which is optimal with respect to E_μ.

We now obtain error bounds for these methods by a generalization of the procedure used for conjugate gradients [16]. From (3.7),

$$x - x_i = x - x_0 - P_{i-1}(A) r_0,$$

so that

$$r_i = r_0 - AP_{i-1}(A) r_0 = (I - AP_{i-1}(A)) r_0$$

and

$$E_\mu(x_i) = (r_i, A^{\mu-2} r_i)$$
$$= (R_i(A) r_0, A^{\mu-2} R_i(A) r_0),$$

where $R_i(A) \equiv I - AP_{i-1}(A)$.

Define $\bar{R}_m$ to be the set of polynomials $R_m(y)$ of degree at most m in y satisfying $R_m(0) = 1$. Then, by Theorem 3.3, at the i^{th} iteration these methods choose the particular polynomial $R_i \in \bar{R}_i$ that minimizes the functional E_μ, i.e.,

$$E_\mu(x_i) = \min_{R_i \in \bar{R}_i} (R_i(A) r_0, A^{\mu-2} R_i(A) r_0) \tag{3.8}$$

Since A is symmetric, it has N orthonormal eigenvectors $\{v_j \mid 1 \leq j \leq N\}$, i.e.,

$$Av_j = \lambda_j v_j, \quad j = 1, 2, \ldots, N,$$

where $\{\lambda_j\}_{j=1}^N$ are the eigenvalues of A.

Since

$$r_0 = \sum_{j=1}^N t_j v_j \quad \text{for some scalars } \{t_j\}_{j=1}^N,$$

we have

$$R_i(A) r_0 = \sum_{j=1}^N t_j R_i(A) v_j = \sum_{j=1}^N t_j R_i(\lambda_j) v_j$$

and from (3.8),

$$E_\mu(x_i) = \min_{R_i \in \bar{R}_i} \left(\sum_{j=1}^N t_j R_i(\lambda_j) v_j, A^{\mu-2} \sum_{j=1}^N t_j R_i(\lambda_j) v_j \right)$$

$$= \min_{R_i \in \bar{R}_i} \left(\sum_{j=1}^{N} t_j R_i(\lambda_j) v_j, \sum_{j=1}^{N} t_j R_i(\lambda_j)\, \lambda_j^{\mu-2} v_j \right)$$

$$= \min_{R_i \in \bar{R}_i} \sum_{j=1}^{N} t_j^2 R_i^2(\lambda_j)\, \lambda_j^{\mu-2}$$

$$\leq \min_{R_i \in \bar{R}_i} \left(\max_{1 \leq j \leq N} |R_i(\lambda_j)| \right)^2 \left(\sum_{j=1}^{N} t_j^2\, \lambda_j^{\mu-2} \right)$$

$$\leq \min_{R_i \in \bar{R}_i} \left(\max_{1 \leq j \leq N} |R_i(\lambda_j)| \right)^2 \left(\sum_{j=1}^{N} t_j v_j, \sum_{j=1}^{N} t_j\, \lambda_j^{\mu-2} v_j \right)$$

$$= \min_{R_i \in \bar{R}_i} \left(\max_{1 \leq j \leq N} |R_i(\lambda_j)| \right)^2 \left(\sum_{j=1}^{N} t_j v_j, A^{\mu-2} \sum_{j=1}^{N} t_j v_j \right)$$

$$= \min_{R_i \in \bar{R}_i} \left(\max_{1 \leq j \leq N} |R_i(\lambda_j)| \right)^2 (r_0, A^{\mu-2} r_0)$$

$$= Q_i^2\, E_\mu(x_0),$$

where

$$Q_i = \min_{R_i \in \bar{R}_i} \max_{1 \leq j \leq N} |R_i(\lambda_j)|. \tag{3.9}$$

We then have the following result (see [27] for the case $\mu = 1$).

Theorem 3.5: For any $i \geq 0$, the iterates of the variational method satisfy

$$\| x - x_i \|_{A^\mu} \leq Q_i \| x - x_0 \|_{A^\mu}$$

where Q_i is as defined in (3.9).

Note that Q_i is independent of μ and depends only on the eigenvalues of A. Depending on what is known about the spectrum of A, we can use different techniques to get bounds on Q_i (see Sections 4.2 and 10.2).

Using Theorem 3.5, we can prove the following result on the maximum number of iterations required to obtain the exact solution (see [12] for $\mu = 1$).

Corollary 3.1: If r_0 has non-zero components along only $k \leq N$ eigenvectors of A, corresponding to only $m \leq k$ distinct eigenvalues, then the methods converge in at most m iterations to the unique solution of $Ax = f$.

Proof: Assume without loss of generality that r_0 has non-zero components only along the eigenvectors $v_1, v_2, \ldots, v_k$ of A. Then

$$r_0 = \sum_{j=1}^{k} t_j v_j \quad \text{for some scalars } \{t_j\}_{j=1}^{k},$$

and proceeding as in the proof of Theorem 3.5,

$$\| x - x_i \|_{A^\mu} \leq Q_i' \| x - x_0 \|_{A^\mu}$$

where

$$Q_i' = \min_{R_i \in \bar{R}_i} \max_{1 \leq j \leq k} |R_i(\lambda_j)|.$$

Now assume, again without loss of generality, that of the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_k$, only $\lambda_1, \lambda_2, \ldots, \lambda_m$, are distinct. Then

$$Q_i' \equiv \min_{R_i \in \bar{R}_i} \max_{1 \leq j \leq m} |R_i(\lambda_j)|.$$

Clearly, the polynomial

$$R_m(y) = \prod_{j=1}^{m} \left(\frac{y - \lambda_j}{\lambda_j}\right)$$

belongs to $\bar{R}_m$. Moreover, it vanishes at $\{\lambda_j\}_{j=1}^m$. Thus, $Q_m' = 0$, which implies that

$$\| x - x_i \|_{A^\mu} = 0, \quad \text{for some } i \leq m.$$

Q.E.D.

3.4: Alternative Generation of Direction Vectors

In this section, we show that in certain cases we can reduce the work per iteration by using a less expensive process than Algorithm 3.1 for generating the new direction vectors.

We prove the following identity.

Theorem 3.6: If

$$p_{i+1} = r_{i+1} + b_i p_i, \tag{3.10a}$$

where

$$b_i = (-r_{i+1}, A^\mu p_i)/(p_i, A^\mu p_i), \tag{3.10b}$$

and p_{i+1} is given by Algorithm 3.1, then

$$\tilde{p}_{i+1} = -a_i p_{i+1}.$$

Proof: By the definition of r_{i+1},

$$\tilde{p}_{i+1} = r_i - a_i A p_i + b_i p_i, \tag{3.11}$$

so that using (3.6b),

$$\tilde{p}_{i+1} = u - a_i A p_i \tag{3.12}$$

where $u \in \{p_0, p_1, \ldots, p_i\}$.

Moreover, substituting (3.10b) in (3.10a) and taking the inner product with $A^\mu p_i$,

$$(\tilde{p}_{i+1}, A^\mu p_i) = 0.$$

For $j < i$,

$$(\tilde{p}_{i+1}, A^\mu p_j) = (r_{i+1}, A^\mu p_j) + b_i (p_i, A^\mu p_j) = 0$$

since the first term vanishes by (3.6d), and the second term vanishes because the p's are A^μ-orthogonal. Hence

$$(\tilde{p}_{i+1}, A^\mu p_j) = 0, \quad j < i+1. \tag{3.13}$$

Also by Algorithm 3.1,

$$p_{i+1} = v + A p_i \tag{3.14}$$

where $v \in \{p_0, p_1, \ldots, p_i\}$ and by Theorem 3.1,

(3.15) $\quad (p_{i+1}, A^{\mu} p_j) = 0, \quad j < i+1.$

Hence

$$\tilde{p}_{i+1} + a_i p_{i+1} \in \{p_0, p_1, \ldots, p_i\},$$

and from (3.13) and (3.15),

$$(\tilde{p}_{i+1} + a_i p_{i+1},\ A^{\mu}(\tilde{p}_{i+1} + a_i p_{i+1})) = 0,$$

which implies that

$$\tilde{p}_{i+1} + a_i p_{i+1} = 0.$$

Q.E.D.

Thus if $a_i \neq 0$, then equations (3.10) generate the same direction vectors as the more expensive Algorithm 3.1. Only the normalization is different, but since that does not effect the validity of the orthogonality relations, it is clear that all the results of Sections 2 and 3 hold.

We note that since the lengths of $\tilde{p}_{i+1}$ and p_{i+1} are different, the step $\tilde{a}_{i+1}$ taken along $\tilde{p}_{i+1}$ will be different from the step a_{i+1} that would have been taken along p_{i+1}. However, the iterate x_{i+2} obtained is the same in both cases. More precisely, defining

$$\tilde{a}_{i+1} = \frac{(r_{i+1},\ A^{\mu-1}\tilde{p}_{i+1})}{(\tilde{p}_{i+1},\ A^{\mu}\tilde{p}_{i+1})},$$

we have

$$(3.16) \quad \tilde{a}_{i+1} = \frac{(r_{i+1},\ -A^{\mu-1}a_i p_{i+1})}{(-a_i p_{i+1},\ -A^{\mu}a_i p_{i+1})} = -\frac{1}{a_i} a_{i+1}$$

since $a_i \neq 0$, so that

$$\tilde{a}_{i+1}\tilde{p}_{i+1} = (-\frac{1}{a_i} a_{i+1})(-\ a_i p_{i+1}) = a_{i+1} p_{i+1}.$$

Thus

$$x_{i+2} = x_{i+1} + a_{i+1} p_{i+1} = x_{i+1} + \tilde{a}_{i+1}\tilde{p}_{i+1}.$$

Moreover, from (3.16) we have that

$$\tilde{a}_{i+1} = 0 \quad \text{iff} \quad a_{i+1} = 0.$$

Therefore, whenever $a_i \neq 0$, instead of using Algorithm 3.1 to generate p_{i+1}, we can use (3.10) and set

$$p_{i+1} = \tilde{p}_{i+1},$$

and

$$a_{i+1} = \tilde{a}_{i+1},$$

thereby reducing the cost of the iteration.

PART II

THE SYMMETRIC POSITIVE DEFINITE PROBLEM

CHAPTER 4

THE CONJUGATE GRADIENT AND CONJUGATE RESIDUAL METHODS

4.1: Introduction

Consider the linear system

$$Ax = f, \tag{4.1}$$

where A is an N x N symmetric positive definite matrix. In Chapter 3, we developed a class of variational methods for solving symmetric linear systems. Now we apply them to systems that are, in addition, positive definite.

In Section 2, we give error bounds, and prove that for the model problem these methods require O(n log n) iterations to reduce the initial error by a factor of n^{-p}, for $p > 0$. In Section 3, we present the simplifications to the general algorithm of Chapter 3 due to the positive-definiteness of A.

Two existing methods, the conjugate gradient method and the conjugate residual methods, are special cases of this class of methods. For these two methods, the iteration is particularly simple. In Sections 4 and 5, we present their computational forms, and in Section 6, we give results of some numerical experiments on the model problem.

4.2: Error Bounds

In Section 3.2, we obtained error bounds for our class of variational methods for symmetric A. In (3.9), $R_i(\lambda)$ is the polynomial of degree i of least deviation from zero on the eigenvalue spectrum of A, normalized so that $R_i(0) = 1$. For general eigenvalue distributions, we cannot find the minimum polynomial, and cannot evaluate Q_i. However, upper bounds on Q_i can be obtained.

For positive definite A, Engeli, Ginzburg, Rutishauser, and Stiefel [27] used the following approach. They let [a,b], a,b>0, be an interval known to contain all the eigenvalues of A. Then to obtain a bound on Q_i, they chose $R_i(\lambda)$ to be the unique polynomial in $\bar{R}_i(\lambda)$ that minimizes the deviation from zero on [a,b], viz., the normalized Chebyshev polynomial on [a,b]. i.e.,

$$R_i(\lambda) = \frac{T_i(\frac{-2\lambda + a + b}{b - a})}{T_i(\frac{b + a}{b - a})},$$

where $T_k(z) = \cos(k \arccos z)$ is the k^{th} Chebyshev polynomial in z. This choice gives ([27] and [16]),

$$Q_i \leq 2 \left(\frac{1 - \sqrt{\alpha}}{1 + \sqrt{\alpha}}\right)^i, \quad \text{for } i \geq 0,$$

where $\alpha \equiv a/b$. Noting that $\kappa(A) = b/a$ and substituting in (3.9), we have the following result (see [27] for the case $\mu = 1$).

Theorem 4.1: The iterates x_i, $i \geq 0$, satisfy

$$\|x - x_i\|_{A^\mu} \leq 2 \left(\frac{1 - \sqrt{\alpha}}{1 + \sqrt{\alpha}}\right)^i \|x - x_0\|_{A^\mu},$$

where $\alpha = 1/\kappa(A)$.

Other assumptions may be made about the distribution of the eigenvalues of A. For example, Axelsson [1] obtains bounds on Q_i by assuming that the eigenvalues of A are distributed over two intervals on the positive real axis and uses results of Lebedev [44] to obtain error bounds. Stewart [55] considers the case when one simple eigenvalue of A lies outside an interval which contains the remaining eigenvalues.

We now apply Theorem 4.1 to obtain the rate of convergence for the positive definite model problem defined in Chapter 2 (see [12] for the case $\mu = 1$).

Theorem 4.2: For the positive definite model problem, $O(n \log n)$ iterations suffice to reduce the 2-norm of the initial error by a factor of n^{-p}, for $p > 0$. The corresponding number of multiplications is $O(n^3 \log n)$ in two dimensions and $O(n^4 \log n)$ in three dimensions. The rate of convergence with respect to $\|\cdot\|_{A^\mu}$ is given by $R \sim \pi h$, as $h \to 0$.

Proof: By Theorem 4.1,

$$\| x - x_i \|_2 \leq 2 \sqrt{\kappa(A^\mu)} \left(\frac{1 - \sqrt{\alpha}}{1 + \sqrt{\alpha}}\right)^i \| x - x_0 \|_2$$

where $\alpha = c_1^2/n^2$, $\sqrt{\kappa(A^\mu)} = c_2 n^q$, and $c_1, c_2, q > 0$ are constants independent of n (see Section 2.3).

Suppose it takes k iterations to reduce this error bound by a factor of n^{-p}, $p > 0$. Then

$$2c_2 n^q \left(\frac{1 - c_1/n}{1 + c_1/n}\right)^k \leq \frac{1}{n^p}$$

or

$$2c_2 n^{q+p} \leq \left(\frac{1 + c_1/n}{1 - c_1/n}\right)^k.$$

Since the log function is monotonically increasing on $[1,\infty)$, k must satisfy

$$k \geq (c_3 \log n) / \log \left(\frac{1 + c_1/n}{1 - c_1/n}\right)$$

where $c_3 > 0$ is a constant independent of n. Now since $c_1 << n$,

$$\log \left(\frac{1 + c_1/n}{1 - c_1/n}\right) \approx \frac{2c_1}{n}$$

and hence $O(n \log n)$ iterations suffice. Since each iteration requires $O(n^2)$ multiplications in two dimensions and $O(n^3)$ multiplications in three dimensions, the first part of the Theorem follows.

For the model problem, it is easy to show that $\alpha = \pi^2 h^2/4$, where $h=1/(n+1)$ is the mesh-width [34]. Thus

$$\frac{1 - \sqrt{\alpha}}{1 + \sqrt{\alpha}} = \frac{1 - \pi h/2}{1 + \pi h/2} \sim 1 - \pi h, \quad \text{as } h \to 0,$$

and the rate of convergence is given by

$$R = -\log (1 - \pi h) \sim \pi h, \quad \text{as } h \to 0.$$

Q.E.D.

4.3: Computational Aspects

In Section 3.4, we showed that if $a_i \neq 0$ on the i^{th} iteration of the variational method (Algorithm 3.2), then the next direction vector can be generated by (3.10) instead of Algorithm 3.1, resulting in a more efficient iteration. If A is positive definite, then a_i *is never zero* so that (3.10) can be used to generate the direction vectors at every iteration. There are also alternative formulae for a_i and b_i.

Theorem 4.3: If A is positive definite, then for each $i \geq 0$,

$$(4.2) \qquad a_i = \frac{(r_i, A^{\mu-1} r_i)}{(p_i, A^\mu p_i)} \neq 0$$

and

$$(4.3) \qquad b_i = \frac{(r_{i+1}, A^{\mu-1} r_{i+1})}{(r_i, A^{\mu-1} r_i)}.$$

Proof: The proof of (4.2) is by induction on i. Since $p_0 = r_0$ and $(r_0, A^{\mu-1} r_0) \neq 0$, (4.2) clearly holds for $i = 0$. Assume that it holds for $i \leq k$. Then by (3.10a) and (3.6d),

$$(p_{k+1}, A^\mu p_{k+1})\, a_{k+1} = (r_{k+1}, A^{\mu-1} p_{k+1})$$

$$= (r_{k+1}, A^{\mu-1} r_{k+1}) + b_k (r_{k+1}, A^{\mu-1} p_k)$$

$$= (r_{k+1}, A^{\mu-1} r_{k+1}) \neq 0,$$

and (4.2) has been proved by induction.

To prove (4.3), note that by Algorithm 3.2, and by (3.6e),

$$(r_{i+1}, A^{\mu-1} r_{i+1}) = (r_i, A^{\mu-1} r_{i+1}) - a_i (Ap_i, A^{\mu-1} r_{i+1})$$
$$= - a_i (Ap_i, A^{\mu-1} r_{i+1}).$$

Hence, applying (4.2),

$$b_i = \frac{(-r_{i+1}, A^{\mu} p_i)}{(p_i, A^{\mu} p_i)} = \frac{(r_{i+1}, A^{\mu-1} r_{i+1})}{a_i (p_i, A^{\mu} p_i)}$$
$$= \frac{(r_{i+1}, A^{\mu-1} r_{i+1})}{(r_i, A^{\mu-1} r_i)},$$

which proves (4.3).

Q.E.D.

Using (4.2) and (4.3) and generating direction vectors by (3.10), we have the following version of the Algorithm 3.2 for the case when A is positive definite:

Algorithm 4.1: The Variational Method for Positive Definite Systems

Step 1: Choose an initial approximation x_0 to x.

Compute $r_0 = f - Ax_0$.

Set $p_0 = r_0$

and $i = 0$.

Step 2: Compute

$a_i = (r_i, A^{\mu-1} r_i)/(p_i, A^{\mu} p_i)$,

$x_{i+1} = x_i + a_i p_i$,

$r_{i+1} = r_i - a_i A p_i$,

$b_i = (r_{i+1}, A^{\mu-1} r_{i+1})/(r_i, A^{\mu-1} r_i)$,

and $p_{i+1} = r_{i+1} + b_i p_i$.

Step 3: If x_{i+1} is sufficiently close to x, terminate the iteration process;

else set $i = i + 1$ and go to Step 2.

4.4: The Conjugate Gradient (CG) Method

The best-known member of this class of methods is the Conjugate Gradient (CG) method. There are essentially two versions of the CG method for solving (4.1) (see [53]). We present the original algorithm due to Hestenes and Stiefel [39] which can be obtained from Algorithm 4.1 by setting $\mu = 1$.

Algorithm 4.2: The Conjugate Gradient Method

Step 1: Choose x_0.

Compute $r_0 = f - Ax_0$.

Set $p_0 = r_0$

and $i = 0$.

Step 2: Compute

$a_i = (r_i,r_i)/(p_i,Ap_i)$,

$x_{i+1} = x_i + a_ip_i$,

$r_{i+1} = r_i - a_iAp_i$,

$b_i = (r_{i+1},r_{i+1})/(r_i,r_i)$,

and $p_{i+1} = r_{i+1} + b_ip_i$.

Step 3: If x_{i+1} is sufficiently close to x, terminate the iteration process;

else set $i = i + 1$ and go to Step 2.

The number of multiplications per iteration for Algorithm 4.2 is $5N + 2$ plus a matrix-vector product. Apart from the matrix A, storage is required only for the four vectors x, r, p, and Ap. Work and storage requirements for the model and generalized model problems are given in Tables 4.1 and 4.2 at the end of this chapter.

Rutishauser [27] considered a version of the conjugate gradient algorithm involving a three-term recurrence for x_i and r_i obtained by eliminating the vector p_i. This version is not as satisfactory from a

practical point of view as Algorithm 4.2 since it requires an additional vector of storage and an additional scalar-vector product per iteration [53].

4.5: The Conjugate Residual (CR) Method

Setting $\mu = 2$ in Algorithm 4.1, we obtain the Conjugate Residual (CR) method, first described by Stiefel [56].

Algorithm 4.3: The Conjugate Residual (CR) Method

Step 1: Choose x_0.

Compute $r_0 = f - Ax_0$.

Set $p_0 = r_0$

and $i = 0$.

Step 2: Compute

$a_i = (r_i,Ar_i)/(Ap_i,Ap_i)$,

$x_{i+1} = x_i + a_ip_i$,

$r_{i+1} = r_i - a_iAp_i$,

$b_i = (r_{i+1},Ar_{i+1})/(r_i,Ar_i)$,

$p_{i+1} = r_{i+1} + b_ip_i$,

and $Ap_{i+1} = Ar_{i+1} + b_iAp_i$.

Step 3: If x_{i+1} is sufficiently close to x, terminate the iteration process;

else set $i = i + 1$ and go to Step 2.

Each iteration of Algorithm 4.3 requires 6N + 2 multiplications plus a matrix-vector product, Ar_{i+1}. In addition to A, storage is required for the five vectors x, r, Ar, p, and Ap. Work and storage requirements for the model and generalized model problems are given in Tables 4.1 and 4.2.

There is a correspondence between the iterates given by the CG and CR methods. As pointed out by Hestenes and Stiefel (see [39] and [42]), the following relation exists between the iterates x_i^{CG} of CG and x_i^{CR} of CR:

$$x_{i+1}^{CR} = \frac{1}{c_{i+1}}(x_{i+1}^{CG} + b_i^{CG} c_i x_i^{CR})$$

and

$$r_{i+1}^{CR} = \frac{1}{c_{i+1}} p_{i+1}^{CG},$$

where

$$c_0 = 1, \quad c_{i+1} = 1 + c_i b_i^{CG}, \quad i \geq 0.$$

For any $i \geq 0$, the CG and the CR iterates minimize the A-norm of the error and the 2-norm of the residual respectively over the same i-dimensional subspace. Hence

$$\| r_i^{CR} \|_2 \leq \| r_i^{CG} \|_2$$

and

$$\| x - x_i^{CG} \|_A \leq \| x - x_i^{CR} \|_A. \tag{4.4}$$

Moreover ([39], Theorem 7.5),

$$\| x - x_i^{CG} \|_2 \leq \| x - x_i^{CR} \|_2. \tag{4.5}$$

If the measure of the error in the iteration process is based on the norms in (4.4) or (4.5), then the CG method will require fewer iterations than the CR method to reduce the error by the same factor. If the measure is based on the 2-norm of the residual, then the CR method will require fewer iterations. In practice, the exact solution x of the linear system is not known. The residual is often used as an error measure and the iteration process is terminated when the residual is small. Therefore Cline [11] suggests that it makes more sense to choose the method that minimizes the 2-norm of the residual, i.e., the CR method. However, it should be noted that the CR method requires one more scalar-vector product per iteration than the CG method.

4.6: Numerical Results

We present results of numerical experiments comparing the performance of the conjugate gradient and conjugate residual methods on the model problem. Our results show that the two methods exhibit very similar convergence behaviour. This is also reflected in the results obtained by Engeli, Ginsburg, Rutishauser, and Stiefel [27] and by Kostem and Schultchen [42] on a variety of test problems.

Tables 4.3 and 4.4 give results of numerical tests on the two- and three-dimensional model problems respectively for various mesh-widths h. Figure 4.1 shows the almost identical convergence behavior of CG and CR. In Figures 4.2a and 4.2b, we have plotted the number of iterations

required by CG to reduce the 2-norm of the error by a factor of $1/n^2$ against (n log n), where the solution was chosen according to (2.7). The graphs are straight lines which illustrates the O(n log n) convergence result for the model problems obtained in Theorem 4.2.

Method	Storage reqd.	No. of mults./ iteration
CG	4N	6N+2
CR	5N	7N+2

Table 4.1: Storage and work requirements for N x N model problem in two and three dimensions

Method	Two dimensions ($N = n^2$)		Three dimensions ($N = n^3$)	
	Storage reqd.	No. of mults./ iteration	Storage reqd.	No. of mults./ iteration
CG	6N-2n	9N-4n+2	$7N-3n^2$	$11N-6n^2+2$
CR	7N-2n	10N-4n+2	$8N-3n^2$	$12N-6n^2+2$

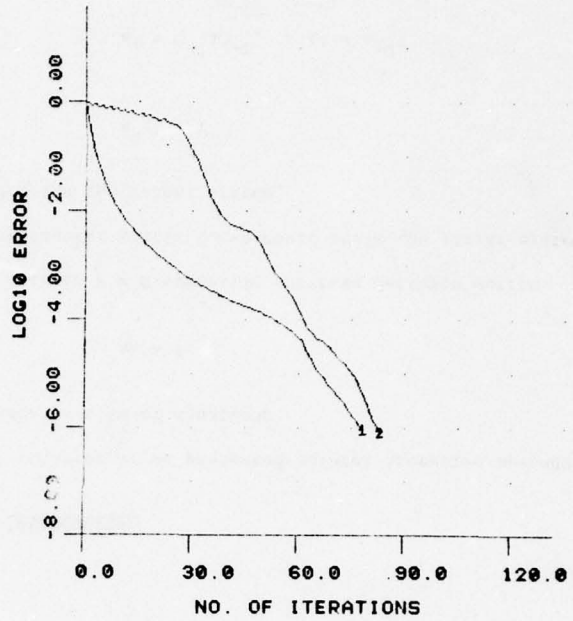
Table 4.2: Storage and work requirements for N x N generalized model problem

Mesh-width h	CG			CR		
	No. of iters.	mults./ iter.	Total no. of mults.	No. of iters.	mults./ iter.	Total no. of mults.
1/16	41	1,352	55,432	42	1,577	66,234
1/32	78	5,768	449,904	83	6,729	558,507
1/64	160	23,816	3,810,560	174	27,785	4,834,590

Table 4.3: Work required by the CG and CR methods to reduce the 2-norm of the error by 10^{-6} for the model problem in two dimensions

Mesh-width h	CG			CR		
	No. of iters.	mults./ iter.	Total no. of mults.	No. of iters.	mults./ iter.	Total no. of mults.
1/4	7	164	1,148	7	191	1,337
1/8	23	2,060	47,380	24	2,403	57,672
1/16	47	20,252	951,844	50	23,627	1,181,350

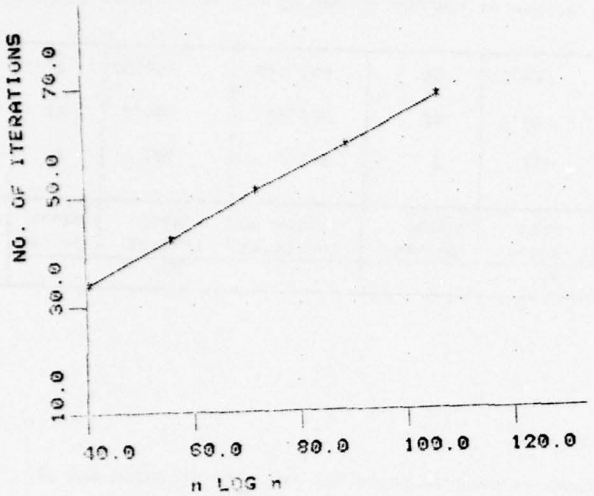
Table 4.4: Work required by the CG and CR methods to reduce the 2-norm of the error by 10^{-6} for model problem in three dimensions



1 Conjugate Gradient Method
2 Conjugate Residual Method

Figure 4.1: Error reduction by the CG and CR methods for model problem in two dimensions (h = 1/32)

(a) Two dimensions



(b) Three dimensions

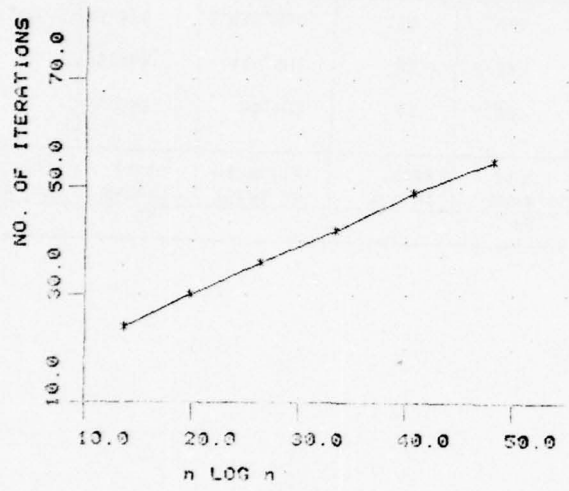


Figure 4.2: Graphs illustrating O(n log n) convergence result for the CG method on the model problem

CHAPTER 5

THE PRECONDITIONED CONJUGATE GRADIENT AND PRECONDITIONED CONJUGATE RESIDUAL METHODS

5.1: Introduction

In Chapter 4, we presented several iterative methods for solving the linear system of equations

(5.1) $$Ax = f,$$

where A is an N x N symmetric positive definite matrix. However, for any nonsingular matrix Q, we could scale the linear system (5.1) and consider the equivalent system

(5.2) $$A'x' = f'$$

where

(5.3) $$A' = Q^{-1}AQ^{-T}, \qquad f' = Q^{-1}f,$$

and

(5.4) $$x' = Q^{T}x.$$

Clearly, A' is symmetric and positive definite, so that the methods of

Chapter 4 could be used to find an approximation to x', from which an approximation to x could be obtained by solving (5.4). From Theorem 4.1 it follows that the lower bound on the rate of convergence is a monotone decreasing function of $\kappa(A)$. Therefore one way to possibly improve the rate of convergence is to choose Q to decrease the condition number of the iteration matrix. More precisely, if Q is chosen such that

$$\kappa(A') < \kappa(A) \tag{5.5}$$

then these algorithms may converge faster for the preconditioned problem (5.2) than for (5.1).

Since Q is nonsingular, $M \equiv QQ^T$ is symmetric positive definite. Conversely, any symmetric positive definite matrix M can be written as a product QQ^T, where Q is nonsingular [62]. Thus the question of choosing an appropriate nonsingular Q for scaling is equivalent to the question of choosing a splitting of the matrix A of the form

$$A = M - R \tag{5.6}$$

where M is an N x N symmetric positive definite matrix such that $\kappa(Q^{-1}AQ^{-T}) < \kappa(A)$.

Another motivation for considering preconditioning is provided by Theorem 3.5. Note that

$$A' = Q^{-1}AQ^{-T} = I - Q^{-1}RQ^{-T}.$$

Thus if R has only m distinct eigenvalues, then A' also has only m distinct eigenvalues. Applying Theorem 3.5, these methods will

require, assuming no roundoff error, at most m iterations to converge to the true solution of (5.2). Thus another way to decrease the number of iterations required is to choose the splitting so that R has only a few distinct eigenvalues as compared to the number of distinct eigenvalues of A.

Different choices of the splitting (5.6) correspond to different preconditionings. In Chapter 6, we present several splittings that, though applicable to a wide class of problems, are particularly suited to systems arising from finite-difference approximations to partial differential equations. In Chapter 7, we propose a preconditioning which can be applied if A is two-cyclic.

We now consider the work involved in applying the iterative methods of Chapter 4 to (5.2). We note that for the preconditioning to be effective, the additional cost of applying the preconditioning must be sufficiently small, so that the decrease in the number of iterations results in a decrease in the total computational effort to solve the problem. For several reasons, we generally do not want to compute A'. For an arbitrary matrix Q, computing A' and f' and then x from x' may be expensive. Also, A may have a nice sparsity structure and A' may not. This means that we may need more storage for A'. Then, since the major portion of the work per iteration is generally spent in forming matrix-vector products, if $A'u$ is much harder to compute than Au for some vector u, then the overall work per iteration may go up substantially. The savings in the total number of iterations for

convergence to any desired accuracy may not compensate for the increased storage and work requirements.

In Section 2, we define the preconditioned methods associated with the splitting (5.6). These methods are mathematically equivalent to applying the unmodified methods to solve (5.2), but do not require the computation of A' and directly give approximations to x rather than to x'. The only additional work introduced by the preconditioning is the solution of systems of the form

$$Mu = v, \tag{5.7}$$

or equivalently, $QQ^T u = v$, at each iteration.

The cases $\mu = 1$ and $\mu = 2$ give the preconditioned CG and the preconditioned CR methods respectively. Choosing M such that (5.5) holds or $Q^{-1}AQ^{-T}$ has few distinct eigenvalues, and (5.7) can be solved efficiently, gives fast convergence for these two methods without appreciably increasing the cost per iteration over the corresponding unpreconditioned methods. We give the computational forms of these two methods in Sections 3 and 4.

5.2: The Preconditioned Variational Methods

Let a'_k, x'_k, r'_k, b'_k, and p'_k denote the quantities computed in the k^{th} iteration of Algorithm 4.1 applied to solve (5.2). We make the following change of variables:

$$\begin{aligned}
&a_k \leftarrow a'_k,\\
&x_k \leftarrow Q^{-T}x'_k,\\
&r_k \leftarrow Qr'_k,\\
&\tilde{r}_k \leftarrow Q^{-T}r'_k = M^{-1}r_k,\\
&b_k \leftarrow b'_k,\\
&\text{and } p_k \leftarrow Q^{-T}p'_k.
\end{aligned} \tag{5.8}$$

Then the iteration can be written in the following form, which we refer to as the preconditioned variational method to solve (5.1) corresponding to the splitting (5.6).

Algorithm 5.1: The Preconditioned Variational Method for Positive Definite Systems

Step 1: Choose x_0.

Compute $r_0 = f - Ax_0$.

Solve $M\tilde{r}_0 = r_0$.

Set $p_0 = \tilde{r}_0$,

and $i = 0$.

Step 2: Compute

$$\begin{aligned}
a_i &= (r_i, (M^{-1}A)^{\mu-1}\tilde{r}_i)/(p_i, A(M^{-1}A)^{\mu-1}p_i),\\
x_{i+1} &= x_i + a_i p_i,\\
r_{i+1} &= r_i - a_i Ap_i, \qquad (= f - Ax_{i+1})\\
\tilde{r}_{i+1} &= \tilde{r}_i - a_i M^{-1}Ap_i, \quad (= M^{-1}r_{i+1})\\
b_i &= (r_{i+1}, (M^{-1}A)^{\mu-1}\tilde{r}_{i+1})/(r_i, (M^{-1}A)^{\mu-1}\tilde{r}_i),\\
\text{and } p_{i+1} &= \tilde{r}_{i+1} + b_i p_i.
\end{aligned}$$

Step 3: If x_{i+1} is sufficiently close to x, terminate the iteration

process;

else set $i = i + 1$ and go to Step 2.

The results of Chapters 3 and 4 hold for a_k', x_k', r_k', b_k', and p_k', the quantities that would be obtained if we applied the unpreconditioned method (Algorithm 4.1) to solve (5.2). By using the relations (5.8), we obtain the corresponding results for the a_k, x_k, r_k, b_k, and p_k in Algorithm 5.1. The next result follows directly from Theorem 4.1.

Theorem 5.1: The iterates x_i, $i \geq 0$, of the preconditioned variational method satisfy

$$\| x - x_i \|_{A(M^{-1}A)^{\mu-1}} \leq 2 \left(\frac{1 - \sqrt{\alpha}}{1 + \sqrt{\alpha}}\right)^i \| x - x_0 \|_{A(M^{-1}A)^{\mu-1}},$$

where $\alpha = 1/\kappa(Q^{-1}AQ^{-T})$. Moreover, x_i minimizes the $A(M^{-1}A)^{\mu-1}$-norm of the error among all approximations of the form

$$x_i = x_0 + P_{i-1}(M^{-1}A)\, M^{-1}A\, (x - x_0),$$

where $P_{i-1}(z)$ is any polynomial of degree at most i-1 in z.

We can also associate with the splitting (5.6), the linear stationary iterative method

$$Mx_{i+1} = Rx_i + f,$$

or, equivalently,

$$x_{i+1} = M^{-1}Rx_i + M^{-1}f. \tag{5.9}$$

By a simple analysis,

$$x_i = x_0 + \tilde{P}_{i-1}(M^{-1}A)\, M^{-1}A\, (x - x_0),$$

where $\tilde{P}_{i-1}(z)$ is a polynomial of degree at most i-1 in z. Thus we have the following comparison result.

Theorem 5.2: The preconditioned method corresponding to a particular splitting of the form (5.6) will converge at least as fast, with respect to the $A(M^{-1}A)^{\mu-1}$-norm of the error, as the linear stationary iterative method corresponding to the same splitting.

If $\rho(M^{-1}R) < 1$ so that (5.9) is convergent, then we can relate its rate of convergence to that of the preconditioned variational method for the same splitting of A.

Theorem 5.3: If $\rho(M^{-1}R) < 1$, then the iterates x_i, $i \geq 0$, of the preconditioned method satisfy

$$\| x_i - x \|_{A(M^{-1}A)^{\mu-1}} \leq 2\left(1 - \frac{2\sqrt{1-\rho(M^{-1}R)}}{\sqrt{1+\rho(M^{-1}R)} + \sqrt{1-\rho(M^{-1}R)}}\right)^i \| x_0 - x \|_{A(M^{-1}A)^{\mu-1}}.$$

Proof: Since $Q^{-1}AQ^{-T} = I - Q^{-1}RQ^{-T}$

and

$$\rho(Q^{-1}RQ^{-T}) = \rho(M^{-1}R) < 1,$$

we have

$$\kappa(Q^{-1}AQ^{-T}) \leq \frac{1 + \rho(M^{-1}R)}{1 - \rho(M^{-1}R)}. \tag{5.10}$$

Thus

$$\frac{\sqrt{\kappa(Q^{-1}AQ^{-T})}-1}{\sqrt{\kappa(Q^{-1}AQ^{-T})}+1} = 1 - \frac{2}{\sqrt{\kappa(Q^{-1}AQ^{-T})}+1}$$

$$\leq 1 - \frac{2\sqrt{1-\rho(M^{-1}R)}}{\sqrt{1+\rho(M^{-1}R)} + \sqrt{1-\rho(M^{-1}R)}}$$

by (5.10), and the result follows from Theorem 5.1.

Q.E.D.

We now show that the iterates of the preconditioned method are independent of a change in M by a scalar factor.

Theorem 5.4: In the preconditioned variational method, if we replace M by $M' = \alpha M$ where $\alpha \neq 0$ is a constant, then the sequence of iterates remains unchanged.

Proof: Denote the quantities obtained when using M' (instead of M) by primes. We prove by induction on i that

(5.11a) $a'_{i-1} = \alpha a_{i-1}$,

(5.11b) $x_i' = x_i$,

(5.11c) $r'_i = r_i$,

(5.11d) $\tilde{r}'_i = \frac{1}{\alpha}\tilde{r}_i$,

(5.11e) $b'_{i-1} = b_{i-1}$,

and

(5.11f) $p'_i = \frac{1}{\alpha}p_i$.

Clearly (5.11) holds for i = 0. Assume that it holds for i = k. Then

$$a'_k = (r'_k, ((\alpha M)^{-1}A)^{\mu-1}\tilde{r}'_k) \,/\, (p'_k, A((\alpha M)^{-1}A)^{\mu-1}p'_k)$$

$$= \frac{1}{\alpha}(r_k, (M^{-1}A)^{\mu-1}\tilde{r}_k) \,/\, (\frac{1}{\alpha^2}(p_k, A(M^{-1}A)^{\mu-1}p_k))$$

$$= \alpha\, a_k.$$

Hence (5.11a) holds for i = k+1.

Since $a'_k p'_k = a_k p_k$, (5.11b) and (5.11c) hold for i = k+1. Also,

$$\tilde{r}'_{k+1} = (\alpha M)^{-1} r'_{k+1} = \frac{1}{\alpha} M^{-1} r_{k+1} = \frac{1}{\alpha}\tilde{r}_{k+1}.$$

For i = k and k+1,

$$(r'_i, ((\alpha M)^{-1}A)^{\mu-1}\tilde{r}'_i) = (r_i, \frac{1}{\alpha}((\alpha M)^{-1}A)^{\mu-1}\tilde{r}_i)$$

$$= \frac{1}{\alpha^\mu}(r_i, (M^{-1}A)^{\mu-1}\tilde{r}_i).$$

and it follows that $b'_k = b_k$. Moreover,

$$p'_{k+1} = \tilde{r}'_{k+1} + b'_k p'_k$$

$$= \frac{1}{\alpha}\tilde{r}_{k+1} + b_k(\frac{1}{\alpha}p_k) = \frac{1}{\alpha}p_{k+1}.$$

Hence (5.11) holds for i = k+1.

Q.E.D.

5.3: The Preconditioned Conjugate Gradient (PCG) Method

The preconditioned variational method takes on its simplest form for $\mu = 1$. The iterates of the resulting method, the Preconditioned Conjugate Gradient (PCG) method, minimize the A-norm of the error over subspaces of increasing dimension. (Note that the CG method also minimizes the A-norm of the error, though over a different subspace.)

Algorithm 5.2: The Preconditioned Conjugate Gradient (PCG) Method

Step 1: Choose x_0.

Compute $r_0 = f - Ax_0$.

Solve $M\tilde{r}_0 = r_0$.

Set $p_0 = \tilde{r}_0$

and $i = 0$.

Step 2: Compute

$a_i = (r_i, \tilde{r}_i)/(p_i, Ap_i)$,

$x_{i+1} = x_i + a_i p_i$,

$r_{i+1} = r_i - a_i Ap_i$,

Solve $M\tilde{r}_{i+1} = r_{i+1}$,

$b_i = (r_{i+1}, \tilde{r}_{i+1})/(r_i, \tilde{r}_i)$,

and $p_{i+1} = \tilde{r}_{i+1} + b_i p_i$.

Step 3: If x_{i+1} is sufficiently close to x, terminate the iteration process;

else set $i = i + 1$ and go to Step 2.

Each iteration of Algorithm 5.2 requires one matrix-vector product, one solve of the form $Mu = v$, and $5N + 2$ multiplications for computing the scalar-vector products and vector inner-products. In addition to A and M, storage is required for the four vectors x, r, p, and Ap. The vector $\tilde{r}$ can share storage with Ap.

The PCG algorithm is a special case of the conjugate gradient algorithm presented by Daniel [16] (take $K = M$, $H = A^{-1}$ and $M = A$). Evans [28] considers the use of preconditioning to accelerate iterative methods in general, and presents a specific preconditioning to increase the rate of convergence of the CG method. Axelsson [1] presents a preconditioned version of Rutishauser's variant of the conjugate gradient method. Concus, Golub, and O'Leary [12] propose a generalized conjugate gradient algorithm which is equivalent to Algorithm 5.2.

5.4: The Preconditioned Conjugate Residual (PCR) Method

Setting $\mu = 2$ in Algorithm 5.1 gives the Preconditioned Conjugate Residual (PCR) method. The resulting iterates minimize the $AM^{-1}A$-norm of the error (or, equivalently, the M^{-1}-norm of the residual) over subspaces of increasing dimension.

Algorithm 5.3: The Preconditioned Conjugate Residual (PCR) Method

Step 1: Choose x_0.

Compute $r_0 = f - Ax_0$.

Solve $M\tilde{r}_0 = r_0$.

Set $p_0 = \tilde{r}_0$.

Compute $A\tilde{r}_0$.

Set $Ap_0 = A\tilde{r}_0$.

and $i = 0$.

Step 2: Solve $Mq_i = Ap_i$.

Step 3: Compute

$$a_i = (\tilde{r}_i, A\tilde{r}_i)/(Ap_i, q_i),$$

$$x_{i+1} = x_i + a_i p_i,$$

$$\tilde{r}_{i+1} = \tilde{r}_i - a_i q_i,$$

$$b_i = (\tilde{r}_{i+1}, A\tilde{r}_{i+1})/(\tilde{r}_i, A\tilde{r}_i),$$

$$p_{i+1} = \tilde{r}_{i+1} + b_i p_i,$$

and $$Ap_{i+1} = A\tilde{r}_{i+1} + b_i Ap_i.$$

Step 4: If x_{i+1} is sufficiently close to x, terminate the iteration process;

else set i = i + 1 and go to Step 2.

Each iteration of Algorithm 5.3 requires one matrix-vector product, $A\tilde{r}_{i+1}$, one solve of the form Mu = v, and 6N + 2 multiplications for computing scalar-vector products and vector inner-products. In addition to A and M, storage is required for the five vectors x, $\tilde{r}$, p, Ap, and $A\tilde{r}$. The vector q can share storage with $A\tilde{r}$. Thus, compared to PCG, the PCR method requires one additional vector of storage and N additional multiplications per iteration.

The true residual $r_{i+1} = f - Ax_{i+1}$ is not computed in the PCR method as it is in the PCG method. If required, it can be computed as the matrix-vector product

$$r_{i+1} = M\tilde{r}_{i+1}$$

or recursively as

$$r_{i+1} = r_i - a_i Ap_i.$$

CHAPTER 6

SOME PRECONDITIONINGS

6.1: Introduction

Consider the linear system of equations

$$Ax = f \tag{6.1}$$

where A is an N x N symmetric and positive definite matrix. In this Chapter, we consider some non-trivial examples of the splitting

$$A = M - R,$$

where M is symmetric and positive definite. Each such splitting gives rise to a preconditioning matrix. We will define some of these preconditionings with special reference to the systems of equations arising from the discretization of elliptic boundary-value problems. The associated preconditioned algorithms are particularly efficient and show good convergence rates when applied to such problems. Many of the results in this Chapter are presented for the PCG method. Similar results can be derived for the other preconditioned methods introduced in Chapter 5.

In Section 2, we discuss diagonal scaling, which is equivalent to choosing M to be a diagonal matrix. In Section 3, we give examples of problems for which the splitting can be chosen such that $M^{-1}A$ has only a few distinct eigenvalues and efficient algorithms are available for solving Mu = v.

From Theorem 5.1, we have that we would like the eigenvalues of $M^{-1}R$ to be as small as possible. From this standpoint the best choice of M is A itself. But solving Mu = v is then as difficult as solving the original system. Usually, the solve step requires factoring M into a product LU, where L and U are lower and upper triangular matrices respectively. We therefore look for matrices M that "resemble" A, can be decomposed into LU without much work and storage, and such that LUu = v can be solved quickly. We can also view this approach as one of "approximately" factoring A into LU, i.e.,

$$A = LU - R$$

such that LU is symmetric positive definite and "close" to A.

Several procedures for approximately factoring A have been suggested. When used as preconditionings, the approximate factorizations of Meijerink and van der Vorst [48] and Evans [28] have previously been shown to considerably improve the rate of convergence of the CG method (see [48], [28]).

In Section 4, we develop a particular approach of "approximately" factoring A for the generalized model problem. Included as special cases are the approximate factorizations of Meijerink and van der Vorst [48], Dupont, Kendall, and Rachford [20], and Stone [6]. We briefly outline their characteristics, domains of applicability and the differences between them. In Section 5, we describe the ADI preconditioning.

In Section 6, we generalize the (point-) SSOR preconditioning, proposed by Evans [28] and analyzed by Axelsson [2], to block-SSOR preconditioning. Using SSOR theory, we obtain a lower bound on the rate of convergence of the corresponding PCG method. For mesh problems, we discuss point and line preconditionings in two dimensions and point, line, and plane preconditionings in three dimensions, and obtain the corresponding rates of convergence for the PCG method. We also consider some questions related to the practical application of block-SSOR preconditionings.

Furthermore, we present examples of anisotropic problems which are special cases of the generalized model problem. We derive rates of convergence for the PCG method, and show that as the problem gets more anisotropic, there is an increase in the factor of improvement of the rate of convergence of the line-SSOR as compared to the point-SSOR preconditioning in two and three dimensions, and of the plane-SSOR as compared to the line-SSOR preconditioning in three dimensions.

In Section 7, we present the results of numerical experiments which compare the performance of the PCG method for the model and anisotropic problems for the different preconditionings.

6.2: Diagonal Scaling

Many preconditionings depend on additional properties of A besides symmetry and positive definiteness. Moreover, since each iteration of the PCG method requires the solution of $Mu = v$, the PCG method is more expensive per iteration than the CG method. Thus it may often not be economical to use these preconditionings. In such cases it may still be possible to improve the rate of convergence by scaling the system by a nonnegative diagonal matrix and then applying the CG method [12].

This is equivalent to the choosing M to be a diagonal and nonnegative matrix. Instead of using the PCG method and solving $Mu = v$ at each iteration, it may be more efficient to compute

$$(6.2) \qquad A' \equiv Q^{-1}AQ^{-T}, \quad f' \equiv Q^{-T}f, \quad \text{set } x_0' = Q^{T}x_0,$$

where $M \equiv QQ^{T}$, Q is also diagonal, x_0 is the initial approximation to x, and then solve $A'x' = f'$ by the CG method and obtain x from x' by

$$(6.3) \qquad x = Q^{-T}x'.$$

The extra processing in steps (6.2) and (6.3) is small since Q is diagonal, and no extra storage is required since the non-zero structure

of A′ is the same as that of A. In fact, we need not even store the diagonal entries of the scaled matrix since they are all unity.

The rate of convergence now depends on $\kappa(A')$ instead of $\kappa(A)$. As pointed out in [12], Forsythe and Straus [33] have shown that if A is two-cyclic, (as are the matrices for the model and generalized model problems defined in Chapter 2), then the choice of M that minimizes $\kappa(Q^{-1}AQ^{-T})$ among all diagonal matrices is given by

$$(6.4)\quad M = QQ^T = \begin{bmatrix} a_{11} & & & \\ & a_{22} & & \\ & & \ddots & \\ & & & a_{NN} \end{bmatrix}.$$

An analogous result exists for matrices that are block-two-cyclic [25].

Even if A is not two-cyclic, this preconditioning cannot do too badly. In fact, applying a result of van der Sluis [58],

$$\kappa(Q^{-1}AQ^{-T}) \leq q \min_{D \in \bar{D}} \kappa(DAD^T),$$

where Q is as defined in (6.4), $\bar{D}$ is the class of all positive definite N x N diagonal matrices, and q is the maximum number of non-zero elements in any row of A. Moreover, since the diagonal entries of the scaled matrix are unity, we save N multiplications each time a matrix-vector product is performed.

6.3: Low Rank Remainder Preconditioning

As mentioned in Section 5.1, we would expect a preconditioning to decrease the total number of iterations required if $M^{-1}A$ has significantly fewer distinct eigenvalues than A. Since $M^{-1}A$ has the same number of distinct eigenvalues as R, we should look for splittings where R has few distinct eigenvalues and there are efficient algorithms to solve Mu = v.

There are classes of problems for which such splittings can be chosen quite naturally [49]. One example is M corresponding to a separable elliptic operator and R to a modification due to irregular geometry or special boundary conditions. Another is M corresponding to an approximation to a Hessian matrix and R to a set of updates in a quasi-Newton algorithm. In both cases, R has low rank (thus it has few distinct eigenvalues) and there exist efficient procedures for solving Mu = v [49].

6.4: Approximate Factorization Algorithms

6.4.1: Introduction

In this section we consider splittings of the form

$$A = LU - R,$$

where LU is symmetric positive definite, L is lower triangular, U is

unit upper triangular, and L and U have non-zeroes only in positions in which the lower and upper triangular parts of A respectively have non-zeroes, i.e.,

$$\ell_{ij} = 0 \quad \text{if } a_{ij} = 0 \quad \text{for } i > j$$
$$u_{ij} = 0 \quad \text{if } a_{ij} = 0 \quad \text{for } i < j.$$

Note that we look for LU factorizations, instead of symmetric $\tilde{L}\tilde{L}^T$ factorizations since the backsolve for the former requires slightly less work.

We apply this splitting to the matrix A for the two-dimensional generalized model problem defined in Section 2.3, i.e.,

(6.5) $A \equiv$

$$\begin{bmatrix} E_1 & D_2 & & B_{n+1} & & 0 \\ D_2 & E_2 & & & 0 & \\ & & & & & B_N \\ & & & & & \\ B_{n+1} & & 0 & & & \\ & & & & & D_N \\ 0 & & B_N & & D_N & E_N \end{bmatrix}$$

where $N = n^2$ and

$$D_{mn+1} = 0, \quad m = 1, 2, \ldots, n-1.$$

Define

(6.6a) $L \equiv$

$$\begin{bmatrix} d_1 & & & & & \\ c_2 & d_2 & & & 0 & \\ & & & & & \\ b_{n+1} & & 0 & & & \\ & & & & & \\ 0 & & b_N & & c_N & d_N \end{bmatrix}$$

and

(6.6b) $U \equiv$

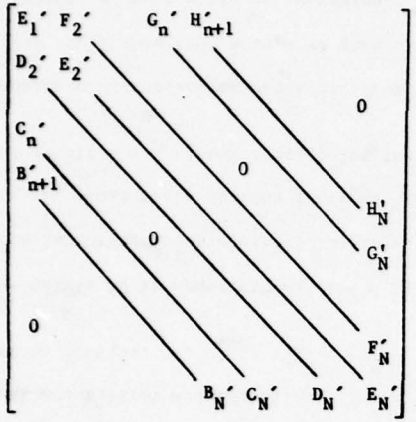
$$\begin{bmatrix} 1 & e_2 & & f_{n+1} & & 0 \\ & 1 & & & 0 & \\ & & & & & f_N \\ & & & & & \\ & & 0 & & & e_N \\ & & & & & 1 \end{bmatrix}$$

where

$$c_{mn+1} = e_{mn+1} = 0, \quad m = 1, 2, \ldots, n-1.$$

Then the product LU has the same sparsity structure as A except for two additional non-zero diagonals C_i' and G_i' adjacent to the extreme non-zero diagonals of A, i.e.,

(6.7) $LU =$



where

$$D'_{mn+1} = F'_{mn+1} = 0, \quad m = 1, 2, \ldots, n-1.$$

If we equate the non-zero elements in the i^{th} row of the matrix obtained by taking the product of L and U to the corresponding elements in (6.7), we obtain the following relations:

$$\begin{aligned}
&B'_i = b_i, && n+1 \le i \le N,\\
&C'_i = b_i e_{i-n+1}, && n \le i \le N,\\
&D'_i = c_i, && 2 \le i \le N,\\
&E'_i = d_i + c_i e_i + b_i f_i, && 1 \le i \le N,
\end{aligned} \tag{6.8}$$

and

$$\begin{aligned}
&F'_i = d_{i-1} e_i, && 2 \le i \le N,\\
&G'_i = c_{i-n+1} f_i, && n \le i \le N,\\
&H'_i = d_{i-n} f_i, && n+1 \le i \le N,
\end{aligned} \tag{6.9}$$

where we assume here and in the following sections that subscripted variables on the right-hand side have the value zero if not otherwise defined. For example, $E_1' = d_1$, since e_1 and f_1 are undefined .

Since we require that LU be symmetric, the diagonals F′, G′, and H′ are completely determined by B′, C′, and D′. Thus

$$\begin{aligned}
&B'_i = H'_i, && n+1 \le i \le N,\\
&C'_i = G'_i, && n \le i \le N,\\
&D'_i = F'_i, && 2 \le i \le N,
\end{aligned} \tag{6.10}$$

which imply respectively

$$\begin{aligned}
&b_i = d_{i-n} f_i, && n+1 \le i \le N,\\
&b_i e_{i-n+1} = c_{i-n+1} f_i, && n+1 \le i \le N,\\
&c_i = d_{i-1} e_i, && 2 \le i \le N.
\end{aligned} \tag{6.11}$$

Note that only two of the three relations in (6.11) are independent. In fact, the diagonals b and c are completely determined by d, e, and f. Thus, for each i, we have only three unknowns (d_i, e_i, and f_i) but four conditions in (6.8) to satisfy. Ideally we would like to set $C_i' = 0$, $B_i' = B_i$, $D_i' = D_i$, and $E_i' = E_i$. However, we cannot do this, since A cannot be factored into such simple factors. The next best thing is to specify the elements so that the resulting LU is as close to A as possible.

This is the point at which the various approximate factorizations use different approximations.

6.4.2: Meijerink and van der Vorst Factorization

The simple choice

$$B_i' = B_i, \quad D_i' = D_i, \quad E_i' = E_i \tag{6.12}$$

leads to the factorization algorithm

$$\begin{aligned} b_i &= B_i, \\ c_i &= D_i, \\ d_i &= E_i - D_i e_i - B_i f_i, \qquad i = 1,2,\ldots,N. \\ e_{i+1} &= D_{i+1}/d_i, \\ f_{i+n} &= B_{i+n}/d_i, \end{aligned} \tag{6.13}$$

This is a computational version of the first variant of Meijerink and van der Vorst's incomplete Cholesky decomposition [48] applied to the two-dimensional generalized model problem. The corresponding PCG method is referred to as ICCG(0).

Since the matrix LU is symmetric and the d_i are all positive [48], the approximate factorization process (6.13) is well-defined and LU is positive definite. This factorization requires 4(N-n) multiplications to compute and requires additional storage for the diagonals d,e, and f.

The approximate factorization procedure proposed by Meijerink and van der Vorst is applicable to a much more general class of matrices. For a general matrix A, we compute an approximate factorization $\tilde{L}\tilde{L}^T$, where $\tilde{L}$ is lower triangular, as follows. First choose the index set P

of those matrix entries which will be forced to be zero in $\tilde{L}$. Second compute an approximate factorization $\tilde{L}\tilde{L}^T$ of A by the Cholesky algorithm modified so that $\tilde{L}_{ij}$ is set to zero for $(i, j) \in P$. Thus the elements of $\tilde{L}$ corresponding to indices in P are neither calculated nor stored. The following result gives sufficient conditions for this algorithm to be well-defined.

Theorem 6.1: ([48]) Let A be a symmetric M-matrix. For each symmetric P, i.e., $(i, j) \in P \Rightarrow (j, i) \in P$, there exists a uniquely defined lower triangular matrix $\tilde{L}$ and a symmetric non-negative matrix R, with $\tilde{L}_{ij} = 0$ if $(i,j) \in P$ and $R_{i,j} = 0$ if $(i, j) \notin P$, such that the splitting $A = \tilde{L}\tilde{L}^T - R$ is a regular splitting.

Because this splitting is regular, the associated linear stationary iterative method

$$\tilde{L}\tilde{L}^T x_{m+1} = Rx_m + f$$

converges [60]. By Theorem 5.2, it follows that the PCG method using the approximate factorization will converge at least as fast. The simplest choice for P is

$$P = \{(i, j) \mid A_{ij} = 0, \quad 1 \le i,j \le N\}$$

in which case $R_{i,j} = 0$ if $A_{ij} \neq 0$, so that $\tilde{L}$ has the same non-zero structure as the lower triangular part of A.

Meijerink and van der Vorst [48] proposed a second variant for the two-dimensional generalized model problem. In this variant

$$P = \{(i, j) \mid |i - j| \neq 0, 1, 2, n-2, n-1, n\}$$

and the resulting PCG method is referred to as ICCG(3). The corresponding matrix $\tilde{L}$ has three additional non-zero diagonals as compared to the $\tilde{L}$ for ICCG(0), and each backsolve requires almost twice the amount of work. However, by allowing $\tilde{\tilde{L}}\tilde{\tilde{L}}^T$ to have more non-zero elements, we are making the decomposition "closer" to A, so that ICCG(3) should require fewer iterations than ICCG(0). Numerical results comparing the two are presented in Section 7.

Since the ICCG splitting is regular, the diagonal matrix $\tilde{D}$ consisting of the diagonal elements of $\tilde{L}$ is strictly positive [60]. Thus $U \equiv \tilde{D}^{-1} \tilde{L}^T$ is unit upper triangular, $L \equiv \tilde{\tilde{L}}\tilde{D}$ is lower triangular and

$$\tilde{\tilde{L}}\tilde{\tilde{L}}^T = LU.$$

In our numerical experiments we compute and store the LU factorization since solving $LUu = v$ requires N less multiplications than solving $\tilde{\tilde{L}}\tilde{\tilde{L}}^T u = v$. The factorization algorithm (6.13) corresponds to the LU factorization for ICCG(0) for the two-dimensional generalized model problem. It can be generalized easily to higher dimensions.

In Appendix A, we show that for the two- and three-dimensional model problems, the condition number of the ICCG(0) iteration matrix is $O(n^2)$, which is asymptotically the same as the condition number of A. Therefore, by Theorem 5.1, O(n log n) iterations are sufficient to reduce the 2-norm of the error by a factor of n^{-p}, for any $p > 0$. Since each iteration of ICCG(0) requires $O(n^2)$ multiplications in two and

$O(n^3)$ multiplications in three dimensions, we have the following result.

Theorem 6.2: For the model problem, O(n log n) iterations are sufficient for ICCG(0) to reduce the 2-norm of the error by a factor of n^{-p}, $p > 0$. The corresponding number of multiplcations is $O(n^3 \log n)$ in two dimensions and $O(n^4 \log n)$ in three dimensions respectively.

This theorem suggests that for the model problem, the improvement in the rate of convergence of ICCG(0) over the CG method may only be by a constant factor. Our numerical results in Section 7 show that this is actually true. However, very good results have been reported with ICCG(0) and ICCG(3) [41], [48], and it has been suggested [48] that the reason for the fast convergence of the ICCG algorithms may be a particularly favorable distribution of the eigenvalues of the iteration matrix. This conjecture is investigated in Section 7.

6.4.3: Dupont, Kendall, and Rachford Factorization

Dupont, Kendall, and Rachford [20] proposed the choice

$$B'_i = B_i, \quad D'_i = D_i, \quad E'_i = E_i(1 + \alpha) - C'_i - C'_{i+n-1}$$

where α is a parameter. With this definition of L and U,

$$(LU\,u)_i = B_i u_{i-n} + C'_i\,(u_{i-n+1} - u_i) + D_i u_{i-1} + E_i(1 + \alpha)\,u_i + D_{i+1} u_{i+1} + C'_{i+n-1}\,(u_{i+n-1} - u_i) + B_{i+n}\,u_{i+n}.$$

Compared with

$$(Au)_i = B_i u_{i-n} + D_i u_{i-1} + E_i u_i + D_{i+1} u_{i+1} + B_{i+n} u_{i+n}$$

we see that the Dupont, Kendall, and Rachford approximate factorization attempts to reduce the effect of the $C'_i u_{i-n+1}$ term by subtracting $C'_i u_i$, and the effect of the $C'_{i+n-1} u_{i+n-1}$ term by subtracting $C'_{i+n-1} u_i$. The iteration parameter α is added to accelerate convergence. This should be contrasted with the Meijerink and van der Vorst factorization procedure which makes no attempt to decrease the effect of the extra terms.

From (6.8) and (6.11), for $i = 1,2,\ldots,N$,

$$b_i = B_i,$$
$$c_i = D_i,$$
$$d_i = E_i(1 + \alpha) - b_i e_{i-n+1} - c_i f_{i+n-1} - c_i e_i - b_i f_i,$$
$$e_{i+1} = D_{i+1}/d_i,$$
$$f_{i+n} = B_{i+n}/d_i.$$

It is clear from these relations that we can compute b_i, c_i, d_i, e_{i+1}, and f_{i+n} (in that order) provided $d_i \neq 0$, $1 \leq i \leq N$. But for $0 \leq \alpha \leq 1$, the $\{d_i\}_{i=1}^N$ are strictly positive [19], and thus the resulting matrix LU is positive definite. Using (6.11) we can replace the equation for d_i by the two equations

$$g_i = c_i f_{i+n-1}$$
$$d_i = E_i(1 + \alpha) - g_{i-n+1} - g_i - c_i e_i - b_i f_i,$$

where it is assumed that $g_i = 0$ for $i < 1$. In this latter form, it is

clear that the Dupont, Kendall, and Rachford factorization procedure requires at most $N + 5(N-n)$ multiplications to compute, and additional storage is required for the diagonals d, e, and f.

Dupont [19] showed that if $\alpha = k_0 h^2$, where $h \equiv 1/(n + 1)$ and $k_0 \geq 0$ is a constant independent of h, and the coefficients $a_i(x)$ in equation (2.1) are uniformly Lipschitz continuous in each of the directions x_i, then the condition number of the preconditioned matrix is $O(h^{-1})$. Thus by Theorem 5.1, $O(n^{1/2} \log n)$ iterations of the PCG method are sufficient to reduce the 2-norm of the error by a factor of n^{-p}, $p > 0$. Since each iteration of the PCG method requires $O(n^2)$ and $O(n^3)$ multiplications in two and three dimensions respectively and $a_i(x) = 1$ for the model problem, we have the following result.

Theorem 6.3: For the model problem, for PCG with the Dupont, Kendall, and Rachford preconditioning (with $\alpha = k_0 h^2$, $k_0 \geq 0$ being a constant independent of h), $O(n^{1/2} \log n)$ iterations suffice to reduce the 2-norm of the error by a factor of n^{-p}, $p > 0$. The corresponding number of multiplications is $O(n^{5/2} \log n)$ in two dimensions and $O(n^{7/2} \log n)$ in three dimensions respectively.

The factorization algorithm can be generalized so that it is applicable to any symmetric matrix A. However, d_i may be zero, in which case the algorithm breaks down. Dupont [19] has proved factorizability for the nine-point discrete Laplacian, as well as for the discretized self-adjoint Neumann problem (with slightly different conditions on α). Moreover the condition number of the preconditioned matrix is $O(h^{-1})$ in

both cases. Hence a result similar to Theorem 6.3 can be stated for these problems. Factorizability conditions in the case when A is an M-matrix are presented in [4].

6.4.4: Stone Factorization

For the matrix in (6.5), Stone's factorization corresponds to setting

$$B_i' = B_i - \alpha C'_{i-1}, \quad D_i' = D_i - \alpha C'_{i-1}, \quad E_i' = E_i + 2\alpha C'_{i-1}$$

where α is a parameter [12]. With this definition of L and U,

$$\begin{aligned}(LU\,u)_i = {} & B_i u_{i-n} + (C_i' (u_{i-n+1} - \alpha u_{i+1}) - \alpha C'_{i-1} (u_{i-n} - u_i)) \\ & + D_i u_{i-1} + E_i u_i + D_{i+1} u_{i+1} \\ & + (C'_{i+n-1} (u_{i+n-1} - \alpha u_{i+n}) - \alpha C'_{i-1} (u_{i-1} - u_i)) + B_{i+n} u_{i+n}\end{aligned}$$

Comparing this with

$$(Au)_i = B_i u_{i-n} + D_i u_{i-1} + E_i u_i + D_{i+1} u_{i+1} + B_{i+n} u_{i+n}$$

we see that the success of the factorization depends on the approximations

$$\begin{aligned} & C_i'(u_{i-n+1} - \alpha u_{i+1}) \simeq \alpha C'_{i-1} (u_{i-n} - u_i) \\ & C'_{i+n-1} (u_{i+n-1} - \alpha u_{i+n}) \simeq \alpha C'_{i-1} (u_{i-1} - u_i)\end{aligned} \tag{6.14}$$

being good enough that the effect of the extra terms in $(LU\,u)_i$ as compared to $(Au)_i$ is reduced.

The convergence properties of this symmetric factorization procedure were studied by Bracha-Barak and Saylor [6]. Using the relations (6.8) and (6.11), they compute the diagonals b, c, d, e, and f of L and U, by

$$\begin{aligned} b_i &= B_i - g_{i-n}, \\ c_i &= D_i - g_{i-n}, \\ d_i &= E_i - 2g_{i-n} - c_i e_i - b_i f_i, \quad i=1,2,\ldots,N. \\ g_i &= \alpha c_i f_{i+n-1}, \\ e_{i+1} &= (D_{i+1} - g_{i-n+1})/d_i, \\ f_{i+n} &= (B_{i+n} - g_i)/d_i. \end{aligned} \tag{6.15}$$

Here again it is assumed that $g_i = 0$, $i < 1$.

For $0 \le \alpha \le 1$, the diagonal entries d_i are strictly positive [6] so that the factorization is well-defined and LU is positive definite. The factorization procedure requires at most $N + 5(N - n)$ multiplications and storage for the diagonals b, c, d, e, and f.

The factorization scheme of (6.15) can be generalized to more general symmetric matrices. Factorizability conditions when A is an M-matrix are derived in [6]. However no bounds are available for the eigenvalues of $(LU)^{-1}A$, even in the simple case of the model problem, and so we are unable to obtain the convergence rate for the PCG method using Stone's preconditioning. Numerical results showing that this

preconditioning does substantially increase the rate of convergence of CG for the model problem are given in Section 7.

6.5: ADI Preconditioning

Let

$$A = H_0 + V_0 + D$$

where D is a nonnegative diagonal matrix, and H_0, V_0, and D satisfy

(1) $H_0 + \alpha D + \beta I$ and $V_0 + \alpha D + \beta I$ are nonsingular for all $\alpha \geq 0$ and $\beta > 0$;

(2) for any $\alpha \geq 0$, $\beta > 0$ and any vectors u, v, the systems $(H_0 + \alpha D + \beta I)y = u$ and $(V_0 + \alpha D + \beta I)z = v$ can be efficiently solved.

Consider the splitting $A = M - R$, where

$$M \equiv \frac{1}{\gamma + \tilde{\gamma}}(H + \gamma I)(V + \tilde{\gamma} I),$$

$$H = H_0 + \tfrac{1}{2}D, \quad V = V_0 + \tfrac{1}{2}D$$

and $\gamma, \tilde{\gamma} > 0$ are parameters. Then the associated linear stationary iterative method is the Peaceman-Rachford alternating direction implicit (ADI) method [62].

In a typical situation, H and V would be tridiagonal, or could be made so by by a permutation of the rows and corresponding columns. For example, for the generalized model problem in two dimensions, H and V

correspond to the discrete analogs of $-h^2 u_{xx}$ and $-h^2 u_{yy}$ respectively. The LU factorization of the two tridiagonal matrices are computed and stored, and each solution of $Mu = v$ requires backsolves corresponding to the two tridiagonal systems.

In the Peaceman-Rachford method, γ and $\tilde{\gamma}$ may vary from iteration to iteration. However, for our preconditioned methods, we consider only the stationary case. For the two-dimensional model problem, the choice [60]

$$\gamma = \tilde{\gamma} = 2 \sin \pi h$$

gives

$$\rho(M^{-1}R) \sim 1 - 2\pi h, \quad \text{as } h \to 0,$$

and substituting in Theorem 5.3, we get the following bound.

Theorem 6.4: For the model problem in two dimensions, $O(n^{1/2} \log n)$ iterations of the PCG method with the ADI (Peaceman-Rachford) preconditioning (with $\gamma = \tilde{\gamma} = 2 \sin \pi h$) are sufficient to reduce the 2-norm of the initial error by a factor of n^{-p}, $p > 0$.

6.6: The Block-SSOR Preconditioning.

6.6.1: Introduction

Let $A^{(\pi)}$ be a partition of the matrix A of the form

$$A^{(\pi)} \equiv \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1q} \\ A_{21} & A_{22} & \cdots & A_{2q} \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ A_{q1} & \cdot & \cdot & A_{qq} \end{bmatrix},$$

where A_{ii} is a square matrix of order p_i. Also, for any vector $x = (x_1, x_2, \ldots x_N)^T$, define vectors $X_1, X_2, \ldots X_q$ corresponding to the partition π, where

$$(6.16) \qquad X_i = (x_{p+1}, x_{p+2}, \ldots x_{p+p_i})^T$$

and

$$p = \sum_{j=1}^{i-1} p_j.$$

In addition, we write

$$A^{(\pi)} = D^{(\pi)} - C^{(\pi)},$$

where

$$D^{(\pi)} \equiv \begin{bmatrix} A_{11} & & & \\ & A_{22} & & \\ & & \ddots & \\ & & & A_{qq} \end{bmatrix}$$

is block diagonal and

$$C^{(\pi)} \equiv C_L^{(\pi)} + C_U^{(\pi)},$$

where $C_L^{(\pi)}$ and $C_U^{(\pi)}$ are strictly lower and upper triangular respectively.

Since $D^{(\pi)}$ is symmetric positive definite, it follows that $(D^{(\pi)})^{-1}$ and $(D^{(\pi)})^{-1/2}$ exist and that $(D^{(\pi)})^{-1}$, $(D^{(\pi)})^{-1/2}$, and $\{A_{ii}^{(\pi)}\}_{i=1}^{q}$ are all symmetric and positive definite. Define

$$B^{(\pi)} = (D^{(\pi)})^{-1} C^{(\pi)} = (D^{(\pi)})^{-1} C_L^{(\pi)} + (D^{(\pi)})^{-1} C_U^{(\pi)},$$
$$\equiv L^{(\pi)} + U^{(\pi)}.$$

Then $B^{(\pi)}$ is the block-Jacobi matrix corresponding to the partition π [62].

Consider the splitting $A^{(\pi)} = M^{(\pi)} - R^{(\pi)}$, where

$$M^{(\pi)} = \frac{1}{\omega(2-\omega)} (D^{(\pi)} - \omega C_L^{(\pi)}) (D^{(\pi)})^{-1} (D^{(\pi)} - \omega C_U^{(\pi)})$$

and $0 < \omega < 2$ is a scalar parameter. Note that since $D^{(\pi)}$ is positive definite, it follows that $(D^{(\pi)} - \omega C_L^{(\pi)})$ and $(D^{(\pi)} - \omega C_U^{(\pi)})$ are nonsingular. Furthermore,

$$M^{(\pi)} = Q^{(\pi)} (Q^{(\pi)})^T,$$

where

$$Q^{(\pi)} = \frac{1}{\sqrt{\omega(2-\omega)}} (D^{(\pi)} - \omega C_L^{(\pi)}) (D^{(\pi)})^{-1/2}$$

so that $M^{(\pi)}$ is symmetric and positive definite.

The linear stationary iterative method associated with this splitting is the block-SSOR method. The rate of convergence of the block-SSOR method is a monotone decreasing function of $\rho(S_\omega^{(\pi)})$, where

$$S_\omega^{(\pi)} \equiv (M^{(\pi)})^{-1} R^{(\pi)}.$$

This rate of convergence is improved by seeking a value of the relaxation parameter ω which minimizes $\rho(S_\omega^{(\pi)})$. The analysis of Habetler and Wachspress [37] and Ehrlich [21] shows that a unique optimum value of ω exists. However, the determination of this value involves the solution of a highly implicit equation. Young [62] proved that in certain cases a "good" though not optimal value ω_1 of ω can be determined which gives an "asymptotically optimal" value for $\rho(S_{\omega_1}^{(\pi)})$. We summarize these results in the following theorem.

Theorem 6.5: ([62]) Let $A^{(\pi)}$ be symmetric and positive definite. Then

(a) there exists a unique value ω_0 of ω such that

$$0 < \omega_0 < 2$$

and unless $\omega = \omega_0$,

$$\rho(S_\omega^{(\pi)}) > \rho(S_{\omega_0}^{(\pi)})$$

and (b) if

$$\omega_1 \equiv \frac{2}{1+(1-2\rho(B^{(\pi)})+4\gamma)^{1/2}}$$

where

$$\gamma \equiv \max\,(\rho(L^{(\pi)}\,U^{(\pi)}),\ 1/4),$$

then

$$\rho(S_{\omega_1}^{(\pi)}) \leq \left[1 - \frac{1-\rho(B^{(\pi)})}{(1-2\rho(B^{(\pi)})+4\gamma)^{1/2}}\right] / \left[1 + \frac{1-\rho(B^{(\pi)})}{(1-2\rho(B^{(\pi)})+4\gamma)^{1/2}}\right].$$

We now consider a few particular choices of the partition π. Denote by π_0 the partition of A where each point constitutes a single block, i.e., the A_{ij}'s are single elements. The corresponding preconditioning will be referred to as point preconditioning. For mesh problems for which the natural ordering is used, let π_L denote the partition where points which lie in the same row in the mesh belong to the same block. The corresponding preconditioning will be referred to as line preconditioning. For mesh problems in three dimensions, let π_P denote the plane preconditioning in which all points on a plane lie in the same block.

For the model problem, it is easy to obtain asymptotic estimates of $\rho(B^{(\pi)})$ explicitly and to thus obtain $\rho(S_{\omega_1}^{(\pi)})$ by direct application of Theorem 6.5.

Theorem 6.6: ([62] and Appendix B) For the model problem in two dimensions,

$$\rho(S_{\omega_1}^{(\pi_0)}) \sim 1 - \pi h, \quad \text{as } h \to 0,$$

where

$$\omega_1 = 2/[1 + \sin(\tfrac{\pi h}{2})] \tag{6.17}$$

and

$$\rho(S_{\omega_1}^{(\pi_L)}) \sim 1 - 2^{1/2}\pi h, \quad \text{as } h \to 0,$$

where

(6.18) $$\omega_1 = 2/[1 + 2\,(\frac{1 - \cos(\pi h)}{2 - \cos(\pi h)})^{1/2}].$$

For the model problem in three dimensions,

$$\rho(S_{\omega_1}^{(\pi_\ell)}) \sim 1 - \pi h, \quad \text{as } h \to 0,$$

where

(6.19) $$\omega_1 = 2/[1 + \sin\frac{(\pi h)}{2}],$$

$$\rho(S_{\omega_1}^{(\pi_L)}) \sim 1 - 2^{-1/2}\,3^{1/2}\,\pi h, \quad \text{as } h \to 0,$$

where

(6.20) $$\omega_1 = 2/[1 + \{6\,\frac{(1-\cos(\pi h))}{1+\cos(\pi h)}\}^{1/2}]$$

and

$$\rho(S_{\omega_1}^{(\pi_P)}) \sim 1 - 3^{1/2}\,\pi h, \quad \text{as } h \to 0,$$

where

(6.21) $$\omega_1 = 2/[1 + \{6\,\frac{(1-\cos(\pi h))}{3-2\cos(\pi h)}\}^{1/2}].$$

Substituting for $\rho(S_{\omega_1}^{(\pi)})$ in Theorem 5.3, we obtain the following result on the rates of convergence of the PCG algorithm for the model problem.

Corollary 6.1: For the PCG method using block-SSOR preconditioning the rates of convergence with respect to $\|\cdot\|_A$ for the model problem, are given by

(a) in two dimensions,

$$R_{\omega_1}^{(\pi_\ell)} \sim 2^{1/2}\,(\pi h)^{1/2}, \quad \text{as } h \to 0, \quad \text{for } \omega_1 \text{ as in (6.17)}$$

$$R_{\omega_1}^{(\pi_L)} \sim 2^{3/4}\,(\pi h)^{1/2}, \quad \text{as } h \to 0, \quad \text{for } \omega_1 \text{ as in (6.18)}$$

(b) in three dimensions,

$$R_{\omega_1}^{(\pi_\ell)} \sim 2^{1/2}\,(\pi h)^{1/2}, \quad \text{as } h \to 0, \quad \text{for } \omega_1 \text{ as in (6.19)}$$

$$R_{\omega_1}^{(\pi_L)} \sim 2^{1/4}\,3^{1/4}\,(\pi h)^{1/2}, \quad \text{as } h \to 0, \quad \text{for } \omega_1 \text{ as in (6.20)}$$

$$R_{\omega_1}^{(\pi_P)} \sim 2^{1/2}\,3^{1/4}\,(\pi h)^{1/2}, \quad \text{as } h \to 0, \quad \text{for } \omega_1 \text{ as in (6.21)}$$

Since the rate of convergence for the CG method is O(h), there is an order of magnitude improvement in applying the block-SSOR preconditioning to the CG method. Also, in the rate of convergence, there is a factor of improvement of $2^{1/4}$ in going from point to line preconditioning in two dimensions, and a factor of improvement of $(3/2)^{1/4}$ and $2^{1/4}$ respectively in going from point to line and line to plane preconditioning in three dimensions. These observations are borne out by the numerical experiments in Section 7.

6.6.2: Computational Aspects

We now consider some questions related to the practical application of block-SSOR preconditioning. Suppose that we have chosen a partition π and a value for ω. Diagonally scaling the original system (6.1) before applying the PCG method with the block-SSOR preconditioning gives the same iterates as would be obtained by applying the preconditioning directly [2]. However, if the cost of the backsolve can be reduced in this way, it may be computationally advantageous to apply such a

scaling.

In general, we can always choose the scaling matrix as in (6.4) so that the diagonal elements of A are unity, and thus save N multiplications per iteration in computing the matrix-vector product.

Also instead of solving $M^{(\pi)}u = v$, we can save N multiplications by solving

$$(D^{(\pi)} - \omega C_L^{(\pi)})\,(D^{(\pi)})^{-1}\,(D^{(\pi)} - \omega C_U^{(\pi)})\,u = v,$$

where we have left off the scalar factor $1/\sqrt{\omega(2-\omega)}$. By Theorem 5.4, the sequence of iterates x_i remains the same.

This equation can also be written as

$$(D^{(\pi)} - \omega C_L^{(\pi)})w = v, \tag{6.22a}$$

$$(D^{(\pi)} - \omega C_L^{(\pi)})u = D^{(\pi)}w. \tag{6.22b}$$

Note that we do not have to compute $D^{(\pi)}w$ to solve (6.22b) since it is implicitly computed in the solution of (6.22a). This becomes more obvious when we write the above equations in the following equivalent form:

For $i = 1, 2, \ldots, q$, solve

$$A_{ii}\,W_i = V_i + \omega \sum_{j=1}^{i-1} A_{ij}\,W_i, \tag{6.23a}$$

and for $i = q, q-1, \ldots 2, 1$, solve

$$A_{ii}\,U_i = A_{ii}\,W_i + \omega \sum_{j=i+1}^{q} A_{ij}\,U_j, \tag{6.23b}$$

where the vectors V_i, W_i, and U_i are defined from v, w, and u respectively as in (6.16). Clearly, the vector $A_{ii}\,W_i$ is formed as the right-hand side while solving the i^{th} system in (6.23a), and can be used to compute the right-hand side while solving the i^{th} system in (6.23b).

We now derive work and storage costs for the model and generalized model problems to factor and solve (6.23) for different choices of π. We suppress the superscript π for the remainder of this subsection.

We first consider the point preconditioning. The matrices A_{ii} are single elements and solving (6.23) is trivial. For any general matrix, we scale the system Ax = f (see Section 6.2) so that the scaled matrix A′ has unit diagonal and solve the system A′x′ = f′. Then corresponding to (6.22), we have that each *solve* step consists of computing

$$w' = v' + (\omega C_L')w'$$

and

$$u' = w' + (\omega C_U')u'.$$

We compute and store the matrices C_U' and $\omega C_U'$ (and hence C_L' and $\omega C_L'$ which are just the respective transposes) and the diagonal scaling matrix. The overhead costs include N square root operations to compute the scaling matrix $D^{1/2}$ and the operations to obtain A′ from A, f′ from f, x_0' from x_0 and finally x_i from x_i' at the last step. The solve step clearly requires twice the number of multiplications as there are non-zeroes in $\omega C_U'$.

For the model problem we do not scale the system as we want to take advantage of the -1's in the off-diagonal non-zero positions. We can write $D = \alpha I$, where $\alpha > 0$ is a constant which is 4 in two and 6 in three dimensions and I is the identity matrix, so that

$$(D - \omega C_L)\, D^{-1}\, (D - \omega C_U) = \alpha\, (I - \tilde{\omega} C_L)\, (I - \tilde{\omega} C_U)$$

where $\tilde{\omega} = \omega/\alpha$. Moreover, if we used $(I - \tilde{\omega} C_L)\, (I - \tilde{\omega} C_U)$ as the preconditioning matrix M, then the same sequence of iterates would be obtained as by using $\alpha(I - \tilde{\omega} C_L)\, (I - \tilde{\omega} C_U)$ (see Theorem 5.4). Thus, for the model problem, we solve at each iteration

$$w' = v' + \tilde{\omega} C_L w'$$

and

$$u' = w' + \tilde{\omega} C_U u'.$$

Since all the non-zero elements in C_L and C_U have modulus unity, this only requires 2N multiplications.

We next consider line-SSOR preconditioning. The blocks in this case are n x n and the matrices A_{ii} are n x n, symmetric, positive definite and tri-diagonal. A technique due to Cuthill and Varga [14] can be incorporated into the PCG method so that the solve step essentially takes the same amount of work as when the A_{ii} are diagonal. There are some overhead operations, but these are not significant since we will solve many systems with the same matrix A_{ii}.

Let

$$A_{ii} = \begin{bmatrix} b_1 & c_1 & & & \\ c_1 & b_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & c_{n-1} & b_n \end{bmatrix}.$$

Then

$$A_{ii} = S_i\, T_i^T\, T_i\, S_i \tag{6.24a}$$

where S_i and T_i are diagonal are upper bi-diagonal respectively and of the form

$$S_i \equiv \begin{bmatrix} s_1 & & & \\ & s_2 & & \\ & & \ddots & \\ & & & s_n \end{bmatrix}, \qquad T_i \equiv \begin{bmatrix} 1 & e_1 & & \\ & 1 & e_2 & \\ & & \ddots & e_{n-1} \\ & & & 1 \end{bmatrix}, \tag{6.24b}$$

where

$$s_1 = b_1^{1/2},\ s_k = [b_k - (e_{k-1}/d_{k-1})^2]^{1/2}, \quad k = 2, 3, \ldots n,$$
$$\text{and } e_k = c_k/(d_k d_{k+1}), \quad k = 1, 2, \ldots n-1. \tag{6.24c}$$

This decomposition takes 3(n-1) multiplications and n square root operations for each A_{ii}. For the model problem, the matrices A_{ii} for all i, are the same, so that the above decomposition has to be done for just one A_{ii} and requires only (2n - 1) storage. However, for a general

problem where the A_{ii} are all different, the decomposition has to be done for each i and the storage requirements for all the S_i's and T_i's is 2N - n.

We then apply the scaling procedure to (6.1) obtaining $A'x' = f'$ by choosing the scaling matrix

$$S = \begin{bmatrix} S_1 & & & \\ & S_2 & & \\ & & \ddots & \\ & & & S_q \end{bmatrix} \tag{6.25}$$

where $q = N/n$ and the S_i's are defined in (6.24). We compute A', x_0', and f' from A, x_0, and f respectively, and note that since S is diagonal,

$$A'_{i,j} = S_i^{-1} A_{i,j} S_j^{-1}$$

so that

$$A_{ii}' = T_i^T T_i.$$

Then, at each iteration, solving $M'u' = v'$ consists of the following sequence of operations.

For i = 1, 2, ... q solve

$$T_i^T T_i W_i' = V_i' + \sum_{j=1}^{i-1} (\omega A'_{i,j}) W_j' \tag{6.26a}$$

and for i = q, q-1, ... 1 solve

$$T_i^T T_i U_i' = T_i^T T_i W_i' + \sum_{j=i+1}^{q} (\omega A'_{i,j}) U_j' \tag{6.26b}$$

where V_i', W_i' and U_i' are vectors defined from v', w' and u' respectively as in (6.16). The vectors $T_i^T T_i W_i'$ in the equations in (6.26b) do not have to be computed since they are exactly the right-hand sides obtained while solving the equations in (6.26a). We compute and store $\omega C_U'$ and hence the $\omega A'_{i,j}$ in the above equations. Then computing the right-hand sides in the above equations require 2(N-n) multiplications in two dimensions and $4(N-n^2)$ multiplications in three dimensions for a generalized model problem. Since each n x n system takes 2(n-1) multiplications to solve and there are 2n and $2n^2$ such systems in two and three dimensions respectively, the whole solve step of computing $u' = (M')^{-1}v'$ requires 6(N-n) in two and $8(N-n^2)$ in three dimensions. Note that since the diagonal of the matrix A' is not unity, the matrix-vector product takes N more multiplications over those required for τ_0 for the generalized model problem.

For τ_L for the model problem, A' does not have any special values for its non-zeroes as was the case for point preconditioning. Hence the cost per iteration for the model problem is the same as that for the generalized model problem. There is some saving in storage and the overhead cost since the S_i and T_i are the same for all i and so have to be computed and stored for just one i. For two-line preconditioning in two dimensions where points that lie in two adjacent rows lie in the same block, the A_{ii}'s are two-cyclic, and there is a similar technique [37] that makes systems with them easy to solve.

In three dimensions plane preconditioning gives subsystems with five non-zero diagonals that may be solved using fast direct methods ([17], [3]).

6.6.3: Preconditioning for Anisotropic Problems

In this subsection, we discuss a class of problems for which line and plane SSOR-preconditioning is especially efficient. In particular, we consider anisotropic difference approximations to the generalized model problem. By anisotropic we mean that the variables have stronger linkage in some coordinate directions than in others. For example, in two dimensions, assume that either the mesh-spacing in the y-direction is wider than in the x-direction or else that α_1 in equation (2.1) is larger than α_2. Then $u_{i,j}$ is more closely linked to $u_{i+1,j}$ and $u_{i-1,j}$ than to $u_{i,j-1}$ and $u_{i,j+1}$.

For ease of analysis, we choose a mesh which is equal and uniform in both directions and let $\alpha_1(x)$, $\alpha_2(x)$, and $\sigma(x)$ be constant functions with

(6.27a) $\alpha_1(x) = P > 1, \quad \alpha_2(x) = 1, \quad \sigma(x) = 0.$

The five-point finite-difference scheme approximates the problem with the N x N system $Ax = f$, where the non-zero structure of A is as in (2.5). The diagonal entries are all equal to $2(1 + P)$, the non-zeroes on the sub- and super-diagonals are equal to -P, and the furthest

diagonals are equal to -1.

Similarly in three dimensions, let $\alpha_1(x)$, $\alpha_2(x)$, $\alpha_3(x)$, and $\sigma(x)$ be constant functions with

(6.27b) $\alpha_1(x) = P > 1, \quad \alpha_2(x) = Q > 1, \quad \alpha_3(x) = 1, \quad P > Q, \quad \sigma(x) = 0.$

The seven-point finite-difference scheme reduces the problem to solving the N x N system $Ax = f$, where A has the same non-zero structure as in (2.6). The entries on the main diagonal are all equal to $2(P + Q + 1)$, and the non-zeroes on the diagonals at distances 1, n + 1, and $n^2 + 1$ from the main diagonal are equal to -P, -Q, and -1 respectively. Thus the linkage between the unknowns at the grid-points is stronger in the x- and y-directions than in the z-direction. Furthermore, since P > Q, the linkage in the x-direction is stronger than in the y-direction. Note that for P = Q = 1, we have the model problem in which the unknown at each grid-point is linked equally strongly to its neighbors in each of the coordinate directions.

Proceeding as in Section 6.6.1, we can obtain $\rho(B^{(\pi)})$ explicitly for the anisotropic problems for $\pi = \pi_0$ and π_L in two dimensions, and $\pi = \pi_0$, π_L, and π_P in three dimensions. As in Corollary 6.1, we can prove the following result on the rate of convergence of the resulting PCG method.

Theorem 6.7: For the PCG method using block-SSOR preconditioning, the rates of convergence for the anisotropic problems of (6.27) are given by

(a) in two dimensions,

$$\text{for } \omega_1 = 2/[1 + 2\sin(\pi h/2)],$$
$$R_{\omega_1}^{(\pi_e)} \sim 2^{1/2}(\pi h)^{1/2}, \quad \text{as } h \to 0;$$

$$\text{for } \omega_1 = 2/[1 + \{2(1 - \frac{\cos(\pi h)}{1+P(1-\cos(\pi h))}\}^{1/2}],$$
$$R_{\omega_1}^{(\pi_L)} \sim 2^{1/2}(P+1)^{1/4}(\pi h)^{1/2}; \quad \text{as } h \to 0,$$

(b) in three dimensions,

$$\text{for } \omega_1 = 2/[1 + 2\sin(\pi h/2)],$$
$$R_{\omega_1}^{(\pi_e)} \sim 2^{1/2}(\pi h)^{1/2}, \quad \text{as } h \to 0;$$

$$\text{for } \omega_1 = 2/[1 + \{2(1 - \frac{(Q+1)\cos(\pi h)}{(P+Q+1)-P\cos(\pi h)})\}^{1/2}],$$
$$R_{\omega_1}^{(\pi_L)} \sim 2^{1/2}(\frac{P+Q+1}{Q+1})^{1/4}(\pi h)^{1/2}, \quad \text{as } h \to 0;$$

$$\text{for } \omega_1 = 2/[1 + \{2(1 - \frac{\cos(\pi h)}{(P+Q+1) - (P+Q)\cos(\pi h)})\}^{1/2}],$$
$$R_{\omega_1}^{(\pi_P)} \sim 2^{1/2}(P+Q+1)^{1/4}(\pi h)^{1/2}, \quad \text{as } h \to 0.$$

It follows from these results that in two dimensions,

$$\frac{R_{\omega_1}^{(\pi_L)}}{R_{\omega_1}^{(\pi_e)}} \sim (P+1)^{1/4}, \quad \text{as } h \to 0,$$

and in three dimensions,

$$\frac{R_{\omega_1}^{(\pi_L)}}{R_{\omega_1}^{(\pi_e)}} \sim (\frac{P+Q+1}{Q+1})^{1/4}, \quad \text{as } h \to 0,$$

and

$$\frac{R_{\omega_1}^{(\pi_P)}}{R_{\omega_1}^{(\pi_1)}} \sim (Q+1)^{1/4}, \quad \text{as } h \to 0.$$

Thus, for suitable values of P and Q, the number of iterations required to reduce the error by a fixed amount can be quite large and can be greatly reduced by going from point to line and line to plane preconditioning. Note that since the cost of a PCG iteration is essentially the same for point and line preconditioning, there is considerable computational advantage in going from point to line preconditioning for even relatively small values of P and Q. Similarly, for three-dimensional problems with Q sufficiently large, the additional cost of going from line to plane preconditioning may be justified.

6.7: Numerical Results

In this section, we present the results of numerical experiments on the model and anisotropic problems. We compare the performance of PCG with the different preconditionings, investigate the dependence of the preconditionings on parameters, and provide experimental evidence to illustrate the results of the preceding sections.

In Tables 6.1 and 6.2 we give the cost per iteration for the generalized model and model problems with different preconditionings. We indicate briefly how these expressions were obtained. From the definition of the PCG algorithm in Chapter 5, it follows that the cost per iteration is 5N + 2 multiplications plus a matrix-vector product plus the cost of solving $LU\bar{r} = r$. Prior to applying the PCG method, problems are usually scaled so that the diagonal elements are unity.

Thus the cost of a matrix-vector product is 4(N - n) and 6(N - n) for the two- and three-dimensional generalized model problems respectively. For the model problems, no scaling is done since it is computationally advantageous to retain -1's as the non-zero off-diagonal terms. Thus the cost of a matrix-vector product is N for model problems. The only exception (as noted in Section 6.6.2) is the line-SSOR preconditioning for the model problem in which the -1's are not retained due to a scaling operation, and the cost of an iteration is the same as for the corresponding generalized model problem.

The cost of the backsolves for the two-dimensional problems have been noted under the respective preconditionings in the preceding sections. The costs for three-dimensional problems can be obtained in an analagous way.

In addition to the iteration cost, there is overhead consisting of the computation of the factorization at the start of the iteration process and any associated scaling. Since this overhead cost is less than the cost of a single iteration, it does not affect the conclusions of our experiments and, therefore, we choose to neglect it.

Figures 6.1a and 6.1b show the error versus number of iterations for the CG and PCG methods with the various preconditionings for the two- and three-dimensional model problems. Tables 6.3a and 6.3b give the corresponding number of multiplications required to reduce the error by a factor of 10^{-6}. It is clear that the PCG method is much more efficient than the CG method. Note that though using the line-SSOR

preconditioning requires fewer iterations than the point-SSOR preconditioning , the total work required is greater. However, the comparison depends on the fact that for the model problem, τ_0 can take advantage of the special values of the non-zeroes in A whereas τ_L cannot.

In Table 6.4 we make the same comparison considering the model problem to be a generalized model problem. As expected, there is a factor of improvement of $\simeq 2^{1/4}$ in the number of iterations in going from point to line preconditioning in two dimensions and a corresponding factor of $\simeq(3/2)^{1/4}$ in three dimensions. However since this improvement is smaller than the increase in the cost per iteration in going from point to line preconditioning, we conclude that it does not pay to use line-SSOR preconditioning for isotropic problems.

In general, we are interested in determining how sensitive the rate of convergence of the PCG method is to the choice of parameters. Figures 6.2(a) - 6.2(d) show the error after 20 iterations for different values of the parameters. For the Dupont, Kendall, and Rachford preconditioning, we found $\alpha = 0$ to be optimal. For Stone's preconditioning, it seems best to choose α slightly less than one. Bracha-Barak and Saylor [6] suggest taking

$$\alpha_{max} = 1 - h^2$$

and then varying α over a cycle of p parameters, where

$$1 - \alpha_i = (1 - \alpha_{max})^{i/(p-1)}, \quad i = 0, 1, \ldots p-1.$$

Since the factorization has to be recomputed for each α, we did not find this scheme to be as efficient. We found that α_{opt} was quite close to the α_{max} defined above and so we chose $\alpha = \alpha_{max}$ in our experiments. For the point and line-SSOR preconditionings, the graphs indicate that it is better to underestimate rather than overestimate ω_{opt}. The experiments with block-SSOR preconditioning indicate that ω_1 is a very good approximation to ω_{opt}, the value of ω (obtained experimentally) that required the minimum number of iterations to reduce the error to 10^{-6}. The results for the two-dimensional model problem are given in Table 6.5. We obtained similar results in three dimensions.

The ICCG algorithms do not depend on any parameters. From Table 6.1, the cost of the ICCG(0) iteration for the generalized model problem in two dimensions is $\simeq$ 14N. Since the L and U in ICCG(3) each have 3 extra diagonals, the corresponding work per iteration is $\simeq$ 20N. Thus ICCG(3) requires about 3/2 as much work per iteration as ICCG(0). The numbers in Table 6.6 show that ICCG(0) requires about twice the number of iterations as ICCG(3) to reduce the error by the same factor. Hence ICCG(3) seems to be more efficient than ICCG(0) in terms of the total work involved.

We next tried to estimate, numerically, the asymptotic rates of convergence for the PCG method with different preconditionings for the model problem. To reduce the error by a factor of n^{-p}, $p > 0$, $O(n \log n)$ iterations suffice for CG and ICCG(0) while $O(n^{1/2} \log n)$ iterations suffice for the PCG method with Dupont, Kendall, and Rachford preconditioning and the SSOR preconditionings. No such theoretical results are available for ICCG(3) or for the PCG method with Stone's preconditioning.

Hence we tried the following numerical experiments. For the two-dimensional model problem, we chose the solution as in (2.7a), and for increasing n, determined the number of iterations required to reduce the error by a factor of 10^{-6}. The graphs in Figure 6.3 indicate that the number of iterations required for ICCG(0) and for the PCG method with the Dupont, Kendall, and Rachford preconditioning are $O(n)$ and $O(n^{1/2})$ respectively. For ICCG(3), the number of iterations required seem to be less than $O(n)$ but not quite $O(n^{1/2})$. Also, for the PCG method, the point- and line-SSOR and Stone's preconditionings required $O(n^{1/2})$ iterations.†

For various n, we computed the condition number of the ICCG(3) iteration matrix. The graph in Figure 6.4 clearly indicates that the condition number is $O(n^2)$. The error bounds for the PCG method (which depend only on the extreme eigenvalues of the iteration matrix) then indicate that $O(n \log n)$ iterations should suffice for ICCG(3). However as shown in Figure 6.5, most of the eigenvalues of the ICCG(3) matrix are bunched around unity. In fact, for N = 100, 93 of the 100 eigenvalues lie in the interval [0.95, 1.05]. Thus, even though the actual condition number may be large, after a few iterations some of the extreme eigenvalues have been annihilated. Thus the effective condition

† We used 10^{-6} instead of $1/n^2$ or $1/n^3$ since in the latter two cases too few iterations were required to make any deductions. Results similar to Theorems 6.2 and 6.3 can be stated showing that $O(n)$ and $O(n^{1/2})$ iterations are sufficient for ICCG(0) and PCG with Dupont, Kendall, and Rachford preconditioning respectively, to decrease the 2-norm of the error by a constant factor.

number is small so that the iteration process shows improved convergence.

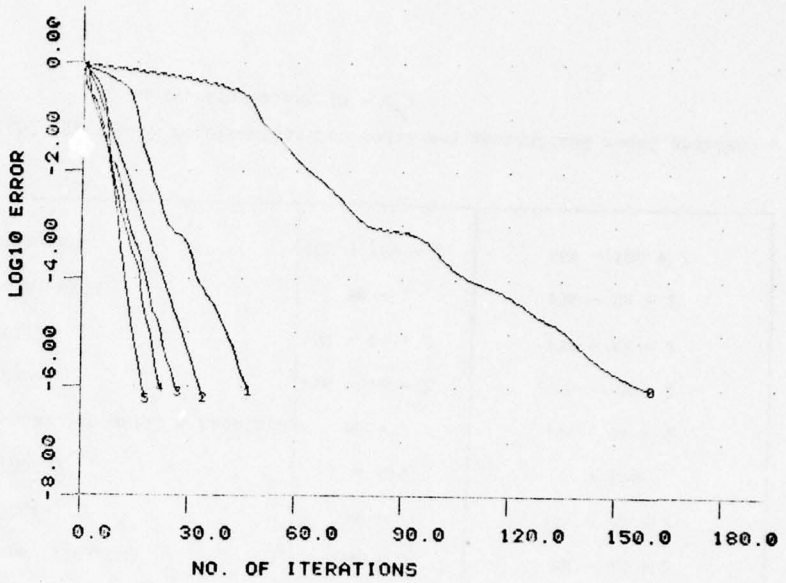
In Table 6.7, we compare point- and line-SSOR preconditionings for anisotropic problems. In two dimensions, we chose P = 100 so that we expect an improvement of a factor of approximately $(101)^{1/4} \approx 3.2$ in going from point to line preconditioning as $h \to 0$. In three dimensions for P = 100, Q = 1, we expect a corresponding improvement of $(51)^{1/4} \approx 2.7$ as $h \to 0$. From Table 6.1, the cost per iteration in going from point- to line-SSOR preconditioning increases by only 1.23 in two and 1.18 in three dimensions for the generalized model problem. Thus as is evident from Figures 6.6a and 6.6b, *the line-SSOR preconditioning is* more efficient in terms of the total work involved for such anisotropic problems.

Preconditioning	No. of mults./iteration	
	Model Problem	Generalized Model Problem
None (i.e., CG)	6N + 2	9N - 4n + 2
ICCG(0)	8N + 2	14N - 8n + 2
ICCG(3)	≈ 17N	≈ 20N
Dupont, Kendall, & Rachford	8N + 2	14N - 8n + 2
Stone	11N - 4n + 2	14N - 8n + 2
ADI	10N - 2n + 2	15N - 8n + 2
Point-SSOR	8N + 2	13N - 8n + 2
Line-SSOR	16N - 10n + 2	16N - 10n + 2

Table 6.1: Work requirements for model and generalized model problems in two dimensions ($N = n^2$)

Preconditioning	No. of mults./iteration	
	Model Problem	Generalized Model Problem
None (i.e., CG)	$6N + 2$	$11N - 6n^2 + 2$
Dupont, Kendall, & Rachford	$8N + 2$	$18N - 12n^2 + 2$
ICCG(0)	$8N + 2$	$18N - 12n^2 + 2$
Point-SSOR	$8N + 2$	$17N - 12n^2 + 2$
Line-SSOR	$20N - 14n^2 + 2$	$20N - 14n^2 + 2$

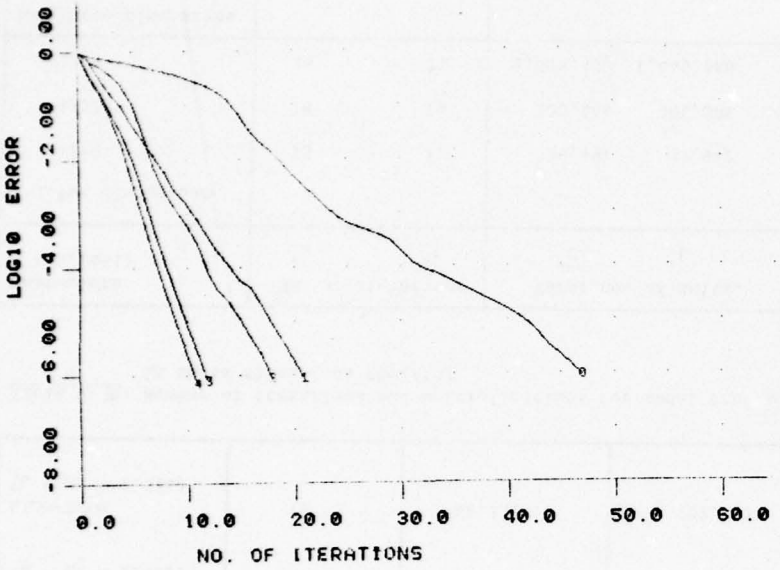
Table 6.2: Work requirements for model and generalized model problems in three dimensions ($N = n^3$)



0 CG
1 ICCG(0)
2 ADI Preconditioning
3† Dupont, Kendall, & Rachford Preconditioning; Stone's Preconditioning
4† ICCG(3); Line-SSOR Preconditioning
5 Point-SSOR Preconditioning

Figure 6.1a: Error reduction for the CG and PCG methods on model problem in two dimensions ($h = 1/64$)

† The plots corresponding to the two preconditionings were almo identical. Therefore, only one has been shown.



0 CG
1 Dupont, Kendall, & Rachford Preconditioning
2 ICCG(0)
3 Point-SSOR Preconditioning
4 Line-SSOR Preconditioning

Figure 6.1b: Error reduction for the CG and PCG methods on model problem in three dimensions (h = 1/16)

Preconditioning	No. of iters.	No. of mults./ iteration	Total no. of mults.
None (i.e., CG)	160	23,816	3,810,560
Meijerink and van der Vorst (ICCG(0))	47	31,754	1,492,438
Dupont, Kendall, & Rachford ($\alpha = 0$)	27	31,754	857,358
Stone ($\alpha = 1 - h^2 = .9998$)	27	43,409	1,172,043
ADI ($\gamma = 2 \sin \pi h = .098$)	34	39,566	1,345,244
Point-SSOR ($\omega = \omega_1 = 1.672$)	26	31,754	825,604
Line-SSOR ($\omega = \omega_1 = 1.870$)	23	62,876	1,446,148

Table 6.3a: Number of iterations and multiplications for model problem in two dimensions (h=1/64)

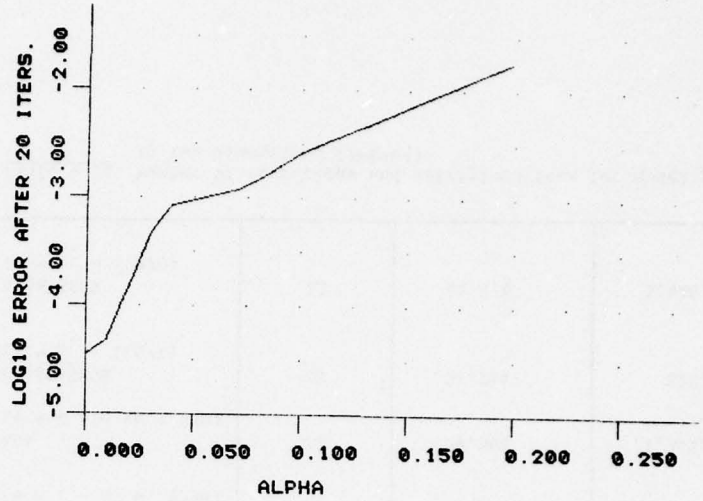
Preconditioning	No. of iters.	No. of mults./ iteration	Total no. of mults.
None (i.e., CG)	47	20,252	951,844
Meijerink and van der Vorst (ICCG(0))	18	27,002	486,036
Dupont, Kendall & Rachford ($\alpha = 0$)	21	27,002	567,042
Point-SSOR ($\omega = \omega_1 = 1.672$)	12	27,002	324,024
Line-SSOR ($\omega = \omega_1 = 1.614$)	11	64,352	707,872

Table 6.3b: Number of iterations and multiplications for model problem in three dimensions (h=1/16)

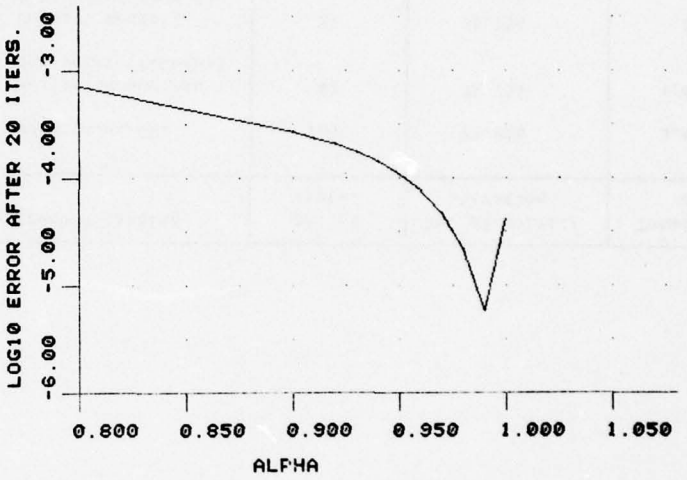
Mesh-width $h = 1/(n+1)$	No. of iterations τ_0	τ_L	Total no. of mults. τ_0	τ_L
(a) Two dimensions				
1/16	13	11	36,491	37,942
1/32	18	16	220,446	241,088
1/64	26	23	1,328,470	1,446,148
(b) Three dimensions				
1/4	6	5	2,118	2,080
1/8	9	8	47,205	49,408
1/16	12	11	656,124	707,872

Table 6.4: Comparison of point- and line-SSOR preconditionings for the generalized model problem ($\omega = \omega_1$).

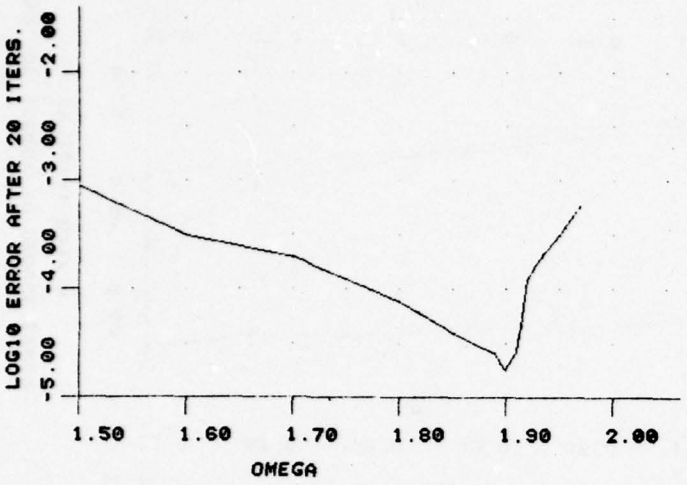
(a) Dupont, Kendall, and Rachford Preconditioning



(b) Stone Preconditioning



(c) Point-SSOR Preconditioning



(d) Line-SSOR Preconditioning

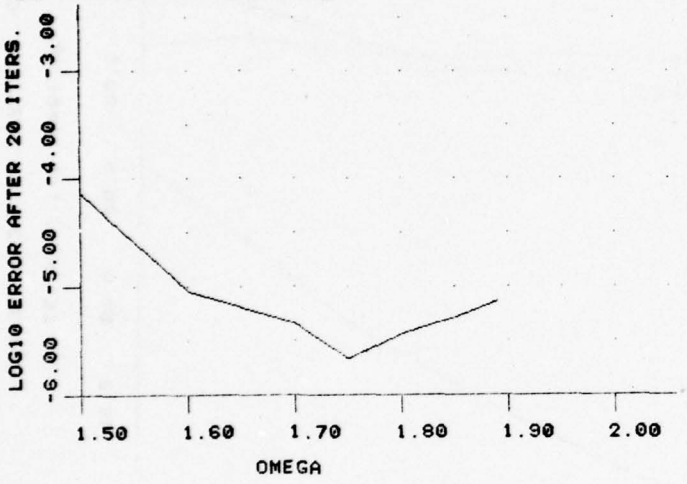


Figure 6.2: Graphs showing dependence of preconditionings on parameters for model problem in two dimensions (h = 1/64)

Preconditioning	Mesh-width h=1/(n+1)	$\omega=\omega_1$		$\omega=\omega_{opt}$	
		ω_1	No.of iters.	ω_{opt}	No. of iters.
Point-SSOR	1/16	1.672	13	1.620	12
	1/32	1.821	18	1.821	18
	1/64	1.906	26	1.906	26
Line-SSOR	1/16	1.569	11	1.569	11
	1/32	1.757	16	1.660	15
	1/64	1.870	23	1.750	21

Table 6.5: Comparison of ω_1 and ω_{opt} for model problem in two dimensions

Mesh-width h=1/(n+1)	No. of iterations ICCG(0)	ICCG(3)
1/16	15	7
1/32	26	13
1/64	47	22

Table 6.6: Comparison of ICCG(0) and ICCG(3) for model problem in two dimensions

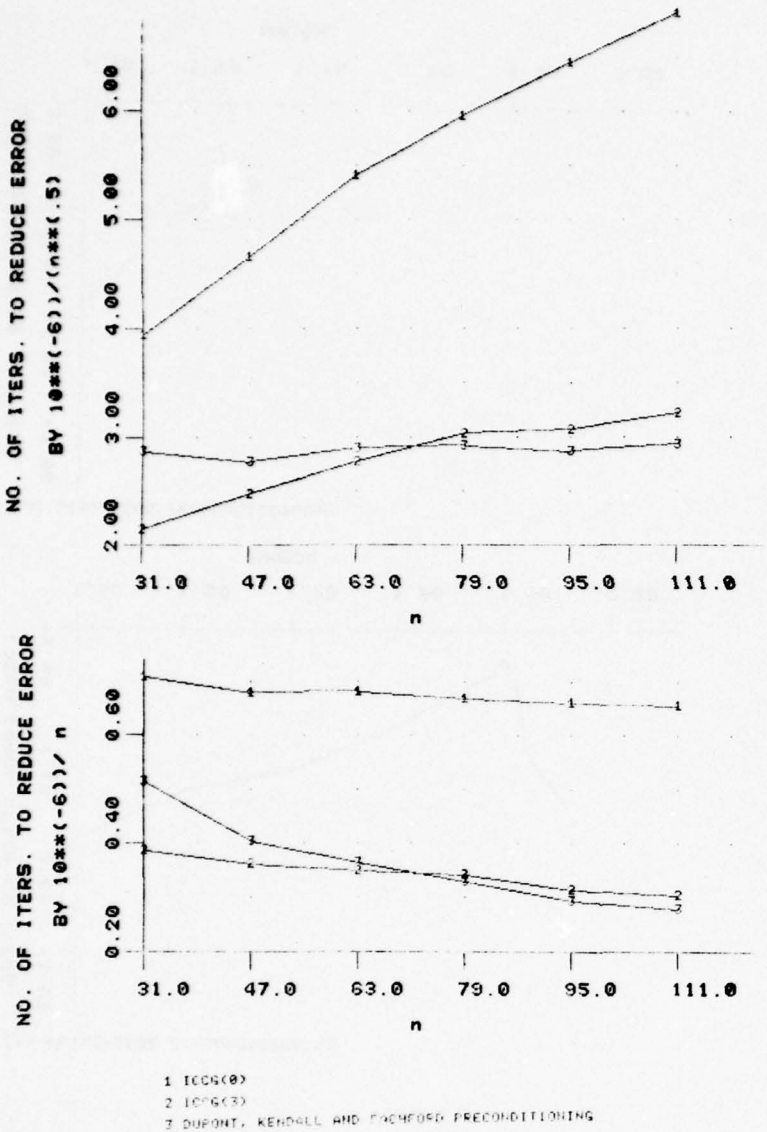


Figure 6.3: Graphs illustrating convergence behavior of the PCG method on model problem in two dimensions

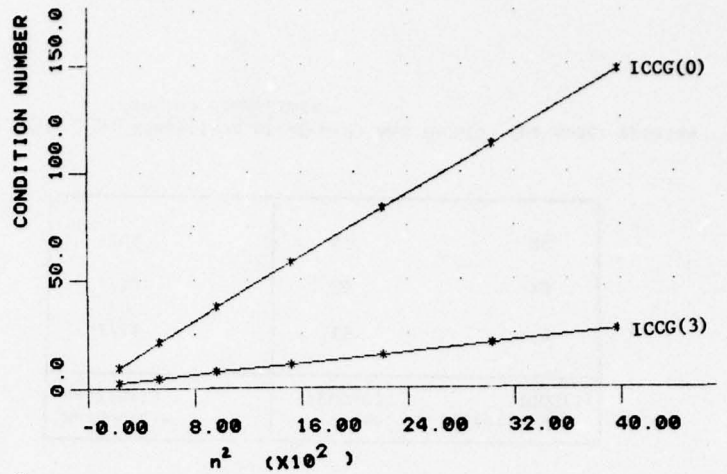


Figure 6.4: variation in condition number of ICCG(0) and ICCG(3) matrices with n^2 for model problem in two dimensions

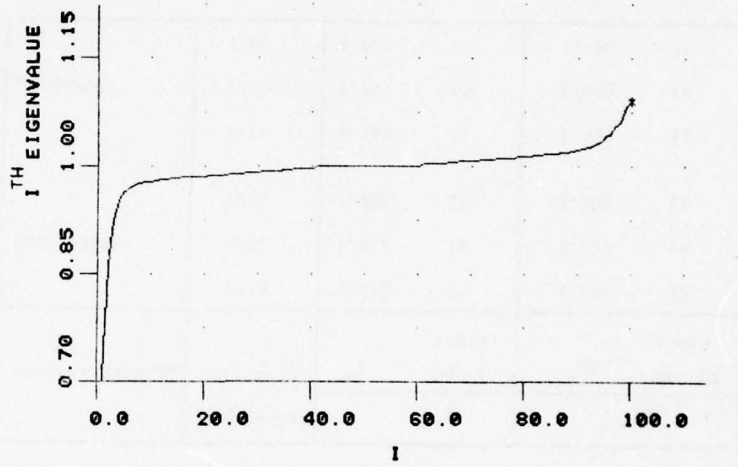


Figure 6.5: Distribution of eigenvalues of ICCG(3) matrix for model problem in two dimensions (h = 1/10)

Mesh-width h=1/(n+1)	No. of iterations π_0	π_L	Factor of improvement π_L over π_0
(a) Two dimensions			
1/16	10	4	2.5
1/32	16	5	3.2
1/64	27	7	3.9
(b) Three dimensions			
1/4	6	3	2.0
1/8	8	4	2.0
1/16	11	4	2.8

Table 6.7: Comparison of point- and line-SSOR preconditionings for anisotropic problems ($\omega = \omega_1$)

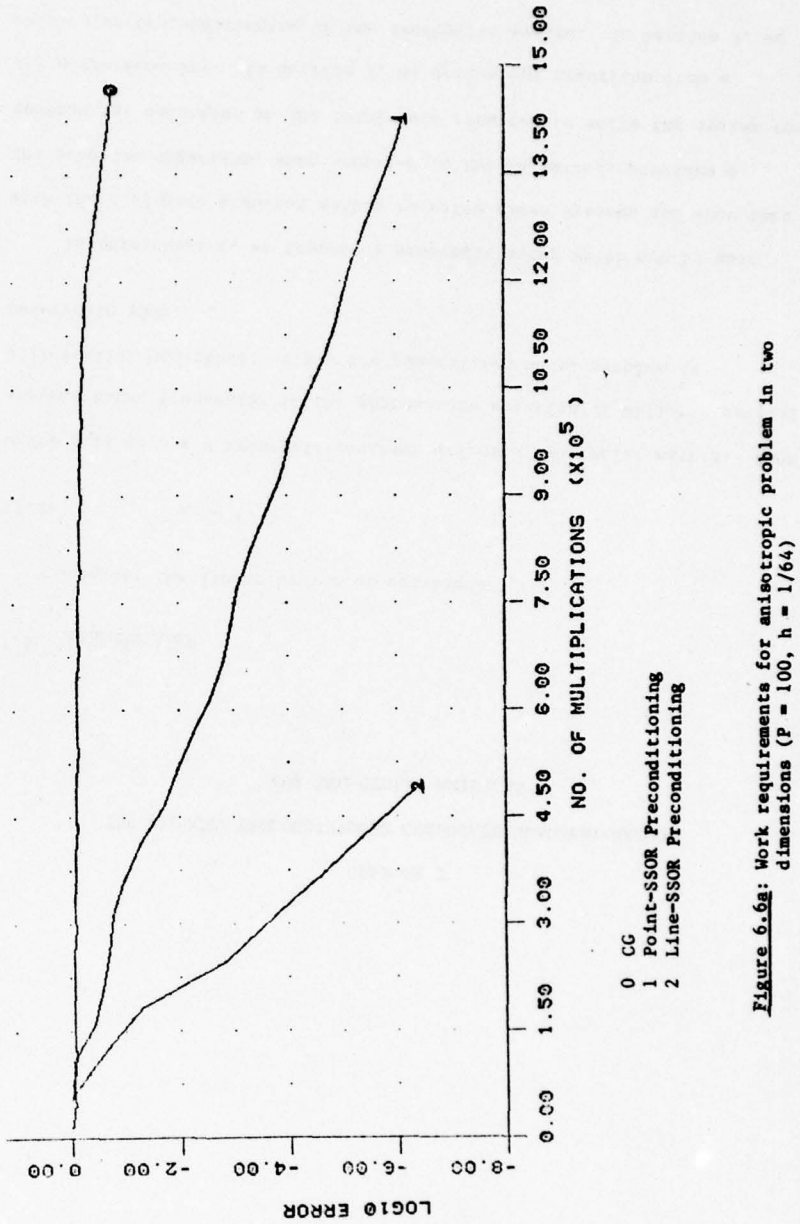
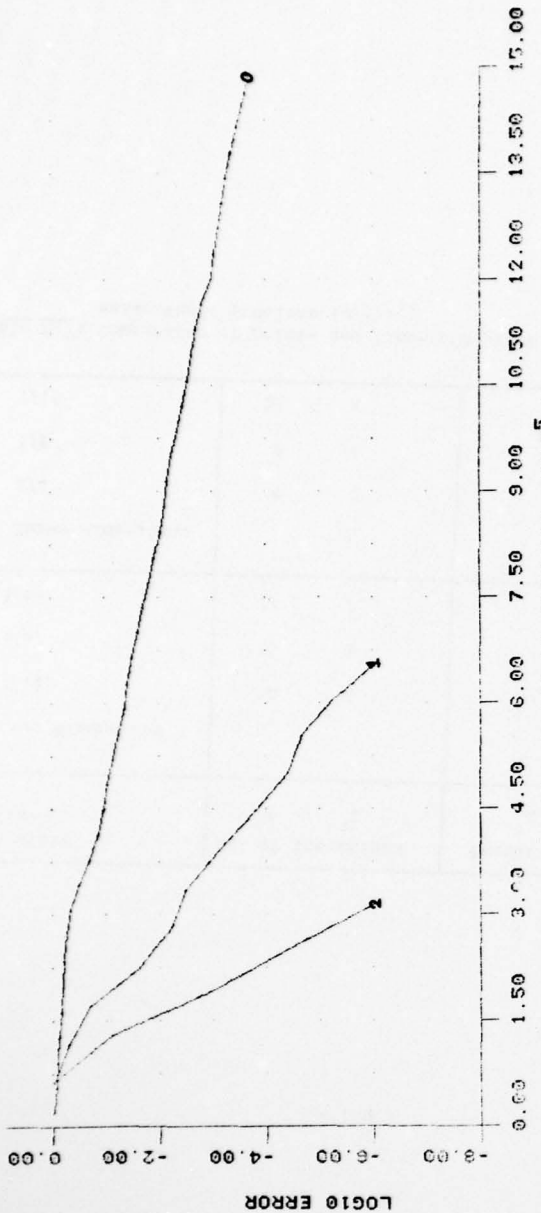


Figure 6.6a: Work requirements for anisotropic problem in two dimensions (P = 100, h = 1/64)



0 CG
1 Point-SSOR Preconditioning
2 Line-SSOR Preconditioning

Figure 6.6b: Work requirements for anisotropic problem in three dimensions (P = 100, Q = 1, h = 1/16)

CHAPTER 7

THE REDUCED PRECONDITIONED CONJUGATE GRADIENT METHOD

FOR TWO-CYCLIC MATRICES

7.1: Introduction

Consider the linear system of equations

(7.1) $Ax = f,$

where A is an N x N symmetric positive definite two-cyclic matrix. Such systems occur frequently in the approximate solution of elliptic partial differential equations; e.g., the generalized model problem is two-cyclic [60].

In this chapter, we propose a preconditioning which can be used with the conjugate gradient method to solve these systems and show that the proposed algorithm, when compared to the CG method, provides a substantial reduction in the total work required to solve the system to any given accuracy. In Section 2, we derive our algorithm from a particular preconditioning of the two-cyclic system. In Section 3, we

show that it is equivalent to a method due to Reid [54] in the sense that the two methods generate the same sequence of iterates. Since the proposed algorithm requires less work per iteration, it is more efficient than Reid's method. In particular, for the model problem, the proposed algorithm gives a 20% improvement in efficiency. In Section 4, we develop and analyze a two-cyclic preconditioning which can be applied to arbitrary symmetric positive definite matrices.

7.2: The Reduced Preconditioned Conjugate Gradient (RPCG) Algorithm

Assume without loss of generality that the variables have been ordered such that the diagonal blocks of A are diagonal matrices. Furthermore, assume that A has been symmetrically scaled so that its diagonal elements are unity. Then (7.1) has the form

$$\begin{bmatrix} I & -Q \\ -Q^T & I \end{bmatrix} \begin{bmatrix} x^{(1)} \\ x^{(2)} \end{bmatrix} = \begin{bmatrix} f^{(1)} \\ f^{(2)} \end{bmatrix}, \tag{7.2}$$

where Q is $n_1 \times n_2$ and $n_1 + n_2 = N$. We assume further that $n_1 \leq n_2$, for if not, we could consider the equivalent system

$$\begin{bmatrix} I & -Q^T \\ -Q & I \end{bmatrix} \begin{bmatrix} x^{(2)} \\ x^{(1)} \end{bmatrix} = \begin{bmatrix} f^{(2)} \\ f^{(1)} \end{bmatrix}$$

instead of (7.2).

The CG method can be used to solve (7.2). To accelerate its convergence, we wish to choose a preconditioning matrix $M \equiv QQ^T$ such that $\kappa(Q^{-1}AQ^{-T}) << \kappa(A)$. Clearly, the best choice for M^{-1} is A^{-1}, but since

the CG iteration requires matrix-vector products, this would require forming $A^{-1}y$, which is of course as hard as the original problem that we are trying to solve. Instead, we look for matrices M such that M^{-1} is, in some sense, "close" to A^{-1}, and systems of the form $Mu = v$ are easy to solve.

For the linear system (7.2), with $A = I - \tilde{Q}$ where

$$\tilde{Q} = \begin{bmatrix} 0 & Q \\ Q^T & 0 \end{bmatrix}$$

we can approximate $A^{-1} = (I - \tilde{Q})^{-1}$ by the first two terms $I + \tilde{Q}$ of the Neumann series. Choosing $M^{-1} = I + \tilde{Q}$, the preconditioned system is

$$(I - \tilde{Q}^2)x = (I + \tilde{Q})f \tag{7.3}$$

which is symmetric positive definite.

Theorem 7.1: $\kappa(I - \tilde{Q}^2) \leq \frac{\kappa(A)}{4}\left[1 + \frac{2}{\kappa(A)} + \frac{1}{(\kappa(A))^2}\right]$.

Proof: Since A is two-cyclic, $\tilde{Q}$ is weakly cyclic of index 2 [60], and thus if λ is an eigenvalue of $\tilde{Q}$, then so is $-\lambda$.

Let λ_i, $i=1,2,..N$, denote the eigenvalues of $\tilde{Q}$, and let

$$\delta = \max_i |\lambda_i|$$

and

$$\delta' = \min_i |\lambda_i|$$

Then since $A = I - \tilde{Q}$,

$$\kappa(A) = \frac{1 + \delta}{1 - \delta}$$

which gives

$$\delta = \frac{\kappa(A) - 1}{\kappa(A) + 1}.$$

Moreover,

$$\kappa(I - \tilde{Q}^2) = \frac{1 - \delta'^2}{1 - \delta^2} \leq \frac{1}{1 - \delta^2}$$

$$= \frac{1}{1 - \left(\frac{\kappa(A) - 1}{\kappa(A) + 1}\right)^2}$$

$$= \frac{\kappa(A)}{4}\left[1 + \frac{2}{\kappa(A)} + \frac{1}{(\kappa(A))^2}\right].$$

Q.E.D.

Thus if $\kappa(A)$ is not too small, then

$$\kappa(I - \tilde{Q}^2) = \frac{1}{4}\kappa(A).$$

A similar result has been proved by Dubois, Greenbaum and Rodrigue [18]. It follows from Theorem 4.1 that to solve (7.2) to any given accuracy, the upper bound on the number of iterations for the CG method applied to (7.3) is approximately one half the number for the CG method applied to (7.2)†. Therefore we would expect faster convergence with the latter approach.

Moreover, note that

$$I - \tilde{Q}^2 = I - \begin{bmatrix} 0 & Q \\ Q^T & 0 \end{bmatrix}\begin{bmatrix} 0 & Q \\ Q^T & 0 \end{bmatrix}$$

† In the next section we will show that exactly half the number of iterations are required if CG is applied to (7.3) rather than to (7.2).

$$= \begin{bmatrix} I - QQ^T & 0 \\ 0 & I - Q^TQ \end{bmatrix}$$

so that (7.3) is

$$\begin{bmatrix} I - QQ^T & 0 \\ 0 & I - Q^TQ \end{bmatrix}\begin{bmatrix} x^{(1)} \\ x^{(2)} \end{bmatrix} = \begin{bmatrix} f^{(1)} + Qf^{(2)} \\ Q^Tf^{(1)} + f^{(2)} \end{bmatrix}.$$

But the variables $x^{(1)}$ and $x^{(2)}$ in the above equation are decoupled. Therefore we could first solve

$$(I - QQ^T)x^{(1)} = f^{(1)} + Qf^{(2)} \tag{7.4}$$

and then obtain $x^{(2)}$ from $x^{(1)}$ by

$$x^{(2)} = f^{(2)} + Q^Tx^{(1)}. \tag{7.5}$$

Using the CG method (Algorithm 4.2) for solving (7.4), we have the following algorithm.

Algorithm 7.1: The Reduced Preconditioned Conjugate Gradient (RPCG) Method for Two-Cyclic Systems

Step 1: Choose $x_0^{(1)}$.

Compute $r_0^{(1)} = f^{(1)} + Qf^{(2)} - (I - QQ^T)x_0^{(1)}$.

Set $p_0^{(1)} = r_0^{(1)}$

and $i = 0$.

Step 2: Compute

$$a_i = (r_i^{(1)}, r_i^{(1)}) / ((I - QQ^T)p_i^{(1)}, p_i^{(1)}),$$

$$x_{i+1}^{(1)} = x_i^{(1)} + a_i p_i^{(1)},$$

$$r_{i+1}^{(1)} = r_i^{(1)} - a_i(I - QQ^T)\, p_i^{(1)},$$

$$b_i = (r_{i+1}^{(1)}, r_{i+1}^{(1)}) / (r_i^{(1)}, r_i^{(1)}),$$

and $p_{i+1}^{(1)} = r_{i+1}^{(1)} + b_i p_i^{(1)}$.

Step 3: If $x_{i+1}^{(1)}$ is sufficiently close to $x^{(1)}$, go to Step 4;

else set i = i+1, and go to Step 2.

Step 4: Terminate the iteration process and compute $x_{i+1}^{(2)}$ by

$$x_{i+1}^{(2)} = f^{(2)} + Q^T x_{i+1}^{(1)}.$$

Each iteration of Algorithm 7.1 requires two matrix-vector products $Q^T p_i^{(1)}$ and $Q(Q^T p_i^{(1)})$. Since all vectors are of length n_1, the rest of the iteration requires $5n_1$ multiplications. In addition to the matrix Q, storage is required for the five vectors $x^{(1)}$, $r^{(1)}$, $p^{(1)}$, $Q^T p^{(1)}$, and $(I - QQ^T)p^{(1)}$.

7.3: Comparison with Reid's Method

In Section 4.4, we mentioned the Rutishauser version of the conjugate gradient method. The iterates obtained are the same as those for the Hestenes and Stiefel version given in (4.5) [53]. For two-cyclic problems, Reid [54] presented a modification of Rutishauser's method which provides a significant improvement in efficiency.

Algorithm 7.2: Reid's Method

Step 1: Choose $x_0^{(1)}$.

Compute $x_0^{(2)} = f^{(2)} + Q^T x_0^{(1)}$

and $r_0^{(1)} = f^{(1)} + Qx_0^{(2)} - x_0^{(1)}$.

Set $r_0^{(2)} = 0$,

$x_{-2} = x_{-1} = x_0$,

$r_{-1} = r_0$,

$e_{-2} = e_{-1} = 0$,

and $m = 0$.

Step 2: Compute

$$q_{2m} = 1 - e_{2m-1},$$

$$r^{(1)}_{2m+1} = \underline{0},$$

$$r^{(2)}_{2m+1} = \frac{1}{q_{2m}} \left[Q^T r^{(1)}_{2m} - e_{2m-1}\, r^{(2)}_{2m-1} \right],$$

$$\text{and } e_{2m} = q_{2m} \frac{(r^{(2)}_{2m+1},\, r^{(2)}_{2m+1})}{(r^{(1)}_{2m},\, r^{(1)}_{2m})}.$$

Step 3: Compute

$$q_{2m+1} = 1 - e_{2m},$$

$$y^{(1)}_{2m+2} = \frac{1}{q_{2m} q_{2m+1}} \left[r^{(1)}_{2m} + e_{2m-1}\, e_{2m-2}\, y^{(1)}_{2m} \right], \tag{7.6}$$

$$x^{(1)}_{2m+2} = y^{(1)}_{2m+2} + x^{(1)}_{2m},$$

$$r^{(1)}_{2m+2} = \frac{1}{q_{2m+1}} \left[Q r^{(2)}_{2m+1} - e_{2m}\, r^{(1)}_{2m} \right],$$

$$r^{(2)}_{2m+2} = \underline{0},$$

$$\text{and } e_{2m+1} = q_{2m+1} \frac{(r^{(1)}_{2m+2},\, r^{(1)}_{2m+2})}{(r^{(2)}_{2m+1},\, r^{(2)}_{2m+1})}.$$

Step 4: If $x^{(1)}_{2m+2}$ is sufficiently close to $x^{(1)}$, go to Step 5; else set $m = m + 1$ and go to Step 2.

Step 5: Terminate the iteration process and compute $x^{(2)}_{2m+2}$ by

$$x^{(2)}_{2m+2} = f^{(2)} + Q^T x^{(1)}_{2m+2}.$$

In Algorithm 7.2, Steps 2 and 3 actually constitute a double iteration. Each double iteration requires one matrix-vector product of the form Qy, one of the form $Q^T y$, and $3N + 2n_1$ multiplications. The average cost for a single iteration is given in Table 7.1. In addition to Q, storage is required for the vectors $r^{(1)}$, $r^{(2)}$, $Q^T r^{(2)}$, $y^{(1)}$ and $x^{(1)}$. $Qr^{(1)}$ shares storage with $Q^T r^{(2)}$.

We will now show that Reid's iteration can be modified so that the quantity within the square brackets in the definition of $y^{(1)}_{2m+2}$ in (7.6) can be computed recursively and the $r^{(2)}$ never have to be computed.

Define

$$p_0 \equiv r_0,$$

$$a_{2m} \equiv \frac{1}{q_{2m} q_{2m+1}}, \tag{7.7}$$

$$p^{(1)}_{2m} \equiv r^{(1)}_{2m} + e_{2m-1}\, e_{2m-2}\, (x^{(1)}_{2m} - x^{(1)}_{2m-2}), \tag{7.8}$$

and

$$b_{2m} \equiv \frac{(r^{(1)}_{2m+2},\, r^{(1)}_{2m+2})}{(r^{(1)}_{2m},\, r^{(1)}_{2m})}. \tag{7.9}$$

Then for $m \geq 0$,

$$x^{(1)}_{2m+2} = x^{(1)}_{2m} + a_{2m} p^{(1)}_{2m}, \tag{7.10}$$

and

$$\begin{aligned} p^{(1)}_{2m+2} &= r^{(1)}_{2m+2} + e_{2m+1}\, e_{2m}\, a_{2m}\, p^{(1)}_{2m} \\ &= r^{(1)}_{2m+2} + q_{2m+1} \frac{(r_{2m+2}, r_{2m+2})}{(r_{2m+1}, r_{2m+1})} q_{2m} \frac{(r_{2m+1}, r_{2m+1})}{(r_{2m}, r_{2m})}\, a_{2m} p^{(1)}_{2m} \end{aligned} \tag{7.11}$$

$$= r^{(1)}_{2m+2} + b_{2m}\, p^{(1)}_{2m}.$$

Thus, generating the $p^{(1)}$'s does not explicitly require forming the $r^{(2)}$'s.

We show that the same is true for the $r^{(1)}$'s. For $m \geq 0$,

$$r_{2m+2} = f - Ax_{2m+2} = f - Ax_{2m} - A(x_{2m+2} - x_{2m})$$

$$= r_{2m} - \begin{bmatrix} I & -Q \\ -Q^T & I \end{bmatrix} \begin{bmatrix} x^{(1)}_{2m+2} - x^{(1)}_{2m} \\ x^{(2)}_{2m+2} - x^{(2)}_{2m} \end{bmatrix}$$

$$= \begin{bmatrix} r^{(1)}_{2m} - (x^{(1)}_{2m+2} - x^{(1)}_{2m}) + Q(x^{(2)}_{2m+2} - x^{(2)}_{2m}) \\ r^{(2)}_{2m} - Q^T(x^{(1)}_{2m+2} - x^{(1)}_{2m}) + (x^{(2)}_{2m+2} - x^{(2)}_{2m}) \end{bmatrix} \tag{7.12}$$

Since $r^{(2)}_{2m} = r^{(2)}_{2m+2} = \underline{0}$,

$$x^{(2)}_{2m+2} - x^{(2)}_{2m} = Q^T(x^{(1)}_{2m+2} - x^{(1)}_{2m}).$$

Substituting in (7.12) and using (7.10),

$$r^{(1)}_{2m+2} = r^{(1)}_{2m} - (I - QQ^T)(x^{(1)}_{2m+2} - x^{(1)}_{2m})$$

$$= r^{(1)}_{2m} - a_{2m}(I - QQ^T)\, p^{(1)}_{2m}, \tag{7.13}$$

so that the $r^{(1)}$'s can be generated without explicitly forming the $r^{(2)}$'s.

However, the definition of the a's depends on the q's, and the q's depend on the $r^{(2)}$'s for their computation. We now show that there is an alternative definition of the a's which gets around this problem.

For the CG method, the residual at any iteration is orthogonal to the subspace spanned by all the preceding residuals by (3.6e), i.e.,

$$(r_i, u) = 0 \quad \text{for } u \in \{r_0, r_1, \ldots r_{i-1}\}. \tag{7.14}$$

From $p^{(1)}_0 = r^{(1)}_0$ and the recurrence relation (7.11) for the p's, we have

$$p^{(1)}_{2m} \in \{r^{(1)}_0, r^{(1)}_2, \ldots r^{(1)}_{2m}\}, \quad m \geq 0.$$

Since $r^{(2)}_{2m+2} = \underline{0}$, it follows from (7.14) that

$$(r^{(1)}_{2m+2}, p^{(1)}_{2m}) = 0, \quad m \geq 0. \tag{7.15}$$

Taking the inner-product of (7.13) with $p^{(1)}_{2m}$ and using (7.15), we get for $m \geq 0$,

$$a_{2m} = (r^{(1)}_{2m}, p^{(1)}_{2m}) \,/\, ((I - QQ^T)\, p^{(1)}_{2m}, p^{(1)}_{2m}). \tag{7.16}$$

Thus the a's can be generated without forming the $r^{(2)}$'s.

Moreover, we can derive an alternate expression for the numerator of (7.16) by taking the inner product of (7.11) with $r^{(1)}_{2m}$. In particular,

$$(r^{(1)}_{2m}, p^{(1)}_{2m}) = (r^{(1)}_{2m}, r^{(1)}_{2m})$$

since the second term on the right-hand side vanishes by (7.15).

Making these substitutions in Algorithm 7.2, we get the following.

Algorithm 7.3: The Modified Reid's Method

Step 1: Choose $x_0^{(1)}$.

Compute $r_0^{(1)} = f^{(1)} + Q(f^{(2)} + Q^T x_0^{(1)}) - x_0^{(1)}$.

Set $p_0^{(1)} = r_0^{(1)}$,

and $m = 0$.

Step 2: Compute

$$a_{2m} = (r_{2m}^{(1)}, r_{2m}^{(1)}) / ((I - QQ^T)\, p_{2m}^{(1)}, p_{2m}^{(1)}),$$

$$x_{2m+2}^{(1)} = x_{2m}^{(1)} + a_{2m}\, p_{2m}^{(1)},$$

$$r_{2m+2}^{(1)} = r_{2m}^{(1)} - a_{2m}\, (I - QQ^T)\, p_{2m}^{(1)},$$

$$b_{2m} = (r_{2m+2}^{(1)}, r_{2m+2}^{(1)}) / (r_{2m}^{(1)}, r_{2m}^{(1)}),$$

and $p_{2m+2}^{(1)} = r_{2m+2}^{(1)} + b_{2m}\, p_{2m}^{(1)}$.

Step 3: If $x_{2m+2}^{(1)}$ is sufficiently close to $x^{(1)}$, go to Step 4; else set $m = m+1$ and go to Step 2.

Step 4: Terminate the iteration process and compute $x_{2m+2}^{(2)}$ by

$$x_{2m+2}^{(2)} = f^{(2)} + Q^T x_{2m+2}^{(1)}.$$

Comparing the Algorithms 7.1 and 7.3, it is clear that

$$x_{i+1} = x'_{2i+2}$$

where x and x' denote iterates obtained in Algorithms 7.1 and 7.3 respectively. The odd indexed vectors and scalars are not computed in Algorithm 7.3, so that the sequence of vectors and scalars actually computed in the two algorithms is the same, and the two algorithms are exactly equivalent.

Moreover, since the numbering of the iterates in Algorithm 7.3 corresponds to the number of iterations for the CG method applied to the original system (7.2), it is clear that the RPCG method actually requires half the number of iterations as the CG method when applied directly to the original problem (7.2).

Method	Work/Iteration		Storage
	Mults.	Products Qy (or $Q^T y$)	(excl. Q)
Hestenes and Stiefel-CG	5N	2	4N
Rutishauser-CG	6N	2	5N
Reid's Method	$1.5N+n_1$	1	$2N+n_1$
Reduced Preconditioned Conjugate Gradient (RPCG) Method†	$2.5n_1$	1	$N+3n_1$

Table 7.1: Work and Storage requirements for the two-cyclic problem

Table 7.1 compares the work and storage requirements for the various versions of the CG method for the two-cyclic problem (7.2). Since $n_1 \leq \frac{N}{2}$, it follows that the RPCG method is always faster than Reid's method. The smaller n_1 is compared to n_2, the greater is the improvement in efficiency. We note that for the model problem $n_1 = \frac{N}{2}$, and the products Qy (or $Q^T y$) take $\frac{N}{2}$ multiplications. Therefore from Table 7.1 it follows that the RPCG method gives a 20% improvement in efficiency over Reid's method.

† The work requirements for the RPCG method derived in Section 7.2 have been halved, since the RPCG method requires exactly half the number of iterations required by the other three algorithms.

7.4: Two-Cyclic Preconditioning

In this section we present a preconditioned version of the RPCG method which is applicable to solving any linear system

(7.17) $\quad Ax = f,$

for which

$$A = \begin{bmatrix} S & -Q \\ -Q^T & T \end{bmatrix}$$

is an N x N symmetric positive definite matrix and the systems $Su = v$ and $Ty = z$ are easy to solve.

As an example of this type of matrix structure, consider the block-tridiagonal system $Ax = f$ where

$$A = \begin{bmatrix} B_1 & C_1 & & & \\ C_1^T & & \ddots & & \\ & \ddots & & C_{n-1} & \\ & & C_{n-1}^T & B_n \end{bmatrix}$$

and the systems $B_i u = v$, $i = 1,\ldots,n$ are easy to solve. For simplicity, we assume that n is even. Then by a suitable reordering of the unknowns, $Ax = f$ can be transformed to the system $\tilde{A}\tilde{x} = \tilde{f}$ where

$$\tilde{A} = \begin{bmatrix} B_1 & & & & C_1 & & & \\ & B_3 & & & C_2^T & C_3 & & \\ & & \ddots & & & \ddots & \ddots & \\ & & & B_{n-1} & & & C_{n-2}^T & C_{n-1} \\ \hline C_1^T & C_2 & & & B_2 & & & \\ & C_3^T & \ddots & & & B_4 & & \\ & & \ddots & C_{n-2} & & & \ddots & \\ & & & C_{n-1}^T & & & & B_n \end{bmatrix}.$$

The choice

$$S = \begin{bmatrix} B_1 & & \\ & B_3 & \\ & & \ddots \\ & & & B_{n-1} \end{bmatrix}, \quad T = \begin{bmatrix} B_2 & & \\ & B_4 & \\ & & \ddots \\ & & & B_n \end{bmatrix},$$

yields systems that are easily solvable.

Matrices of this form often arise in discretizations of elliptic partial differential equations. For example, if A corresponds to the nine-point difference approximation [34] with the natural ordering of the unknowns, and all variables corresponding to points on the same horizontal (or vertical) line of the mesh belong to the same block, then the B_i's and the C_i's are tridiagonal matrices. The tridiagonal systems involving S and T may be solved using the technique of Cuthill and Varga [14] described in Section 6.6.2. If the B_i's are all the same, then only one B_j need be factored. Note that the nine-point problem is not two-cyclic [62], so that the RPCG method is not applicable.

Assume without loss of generality that S is $n_1 \times n_1$ and $n_1 \leq \frac{N}{2}$. Let

$$M = \begin{bmatrix} S & 0 \\ 0 & T \end{bmatrix}. \tag{7.18}$$

Then M can be written as $M = WW^T$ where

$$W \equiv \begin{bmatrix} U & 0 \\ 0 & V \end{bmatrix},$$

and $S = UU^T$ and $T = VV^T$, and U, V, and W are nonsingular matrices. Then premultiplying (7.17) by W^{-1}, we get

$$W^{-1}AW^{-T}\,(W^T x) = W^{-1} f$$

or

$$\begin{bmatrix} I & -U^{-1}QV^{-T} \\ -V^{-1}Q^TU^{-T} & I \end{bmatrix} \begin{bmatrix} U^T x^{(1)} \\ V^T x^{(2)} \end{bmatrix} = \begin{bmatrix} U^{-1} f^{(1)} \\ V^{-1} f^{(2)} \end{bmatrix}. \tag{7.19}$$

Since the coefficient matrix of (7.19) is two-cyclic, we can apply Algorithm 7.1 to solve this system. The reduced equation corresponding to (7.4) is

$$(I - U^{-1}QT^{-1}Q^TU^{-T})\,(U^T x^{(1)}) = U^{-1} f^{(1)} + U^{-1}QT^{-1} f^{(2)}.$$

By a simple transformation of variables, we can write the algorithm in the following form in which we are working directly with the variables $x^{(1)}$ and $x^{(2)}$ instead of $U^T x^{(1)}$ and $V^T x^{(2)}$.

Algorithm 7.4: The Reduced Conjugate Gradient Method with Two-Cyclic Preconditioning

Step 1: Choose $x_0^{(1)}$.

Compute $r_0^{(1)} = f^{(1)} + QT^{-1}(f^{(2)} + Q^T x_0^{(1)}) - Sx_0^{(1)}$.

Solve $S\tilde{r}_0^{(1)} = r_0^{(1)}$.

Set $\tilde{p}_0^{(1)} = r_0^{(1)}$,

$p_0^{(1)} = \tilde{r}_0^{(1)}$,

and $i = 0$.

Step 2: Compute

$$a_i = (r_i^{(1)}, \tilde{r}_i^{(1)}) / (\tilde{p}_i^{(1)} - QT^{-1}Q^T p_i^{(1)}, p_i^{(1)}),$$

$$x_{i+1}^{(1)} = x_i^{(1)} + a_i p_i^{(1)},$$

and $r_{i+1}^{(1)} = r_i^{(1)} - a_i(\tilde{p}_i^{(1)} - QT^{-1}Q^T p_i^{(1)})$.

Solve $S\tilde{r}_{i+1}^{(1)} = r_{i+1}^{(1)}$.

Compute

$$b_i = (r_{i+1}^{(1)}, \tilde{r}_{i+1}^{(1)}) / (r_i^{(1)}, \tilde{r}_i^{(1)}),$$

$$\tilde{p}_{i+1}^{(1)} = r_{i+1}^{(1)} + b_i \tilde{p}_i^{(1)},$$

and $p_{i+1}^{(1)} = \tilde{r}_{i+1}^{(1)} + b_i p_i^{(1)}$.

Step 3: If $x_{i+1}^{(1)}$ is sufficiently close to $x^{(1)}$, go to Step 4;

else set $i = i+1$ and go to Step 2.

Step 4: Terminate the iteration process and compute $x_{i+1}^{(2)}$ by

$$x_{i+1}^{(2)} = T^{-1}(f^{(2)} + Q^T x_{i+1}^{(1)}).$$

Note that it suffices, at each iteration, to solve one linear system of the form $Su = v$ and another of the form $Ty = z$. In addition, each iteration requires two matrix-vector products of the form Qy or $Q^T y$ and $6n_1$ multiplications. Storage is required for the matrix A, the four n_1-vectors $x^{(1)}$, $r^{(1)}$, $\tilde{p}^{(1)}$, and $p^{(1)}$, and for the two n_2-vectors required to compute and store $\tilde{p}^{(1)} - QT^{-1}Q^T p^{(1)}$. $\tilde{r}^{(1)}$ can share storage with one of these n_2-vectors.

The CG method can also be used to solve (7.19). The resulting method is the PCG method (Algorithm 5.1) corresponding to the splitting $A = M - R$ where M is the block-diagonal matrix defined in (7.18)). Also Reid's method can be applied to (7.19) and will require the same number of iterations as the PCG method. Reid [54] considered the above preconditioning for the case when S and T are diagonal matrices; and his method can be extended in a straightforward manner to general S and T. We refer to the corresponding method as the Preconditioned Reid's method.

Method	Work/Iteration						Storage (excl. A)
	Mults	Products			Solves		
		Qy or $Q^T y$	Sy	Ty	$S^{-1}y$	$T^{-1}y$	
PCG	5N	2	1	1	1	1	4N
Preconditioned Reid's Method	$1.5N+n_1$	1	0	0	.5	.5	3N
Reduced CG with 2-cyclic prec.†	$3n_1$	1	0	0	.5	.5	$N+4n_1$

Table 7.2: Work and storage requirements for an N x N symmetric positive definite problem

In Table 7.2 we compare the the storage and work requirements of the three preconditioned algorithms to solve (7.18). Since $n_1 \leq \frac{N}{2}$, the reduced CG method with two-cyclic preconditioning is always faster than the other two methods, and the smaller n_1 is compared to n_2, the greater is the improvement in efficiency in going from Reid's method to the reduced CG method with the two-cyclic preconditioning.

† The work requirements for reduced CG with two-cyclic preconditioning have been halved, since it requires exactly half the number of iterations required by the other two algorithms.

CHAPTER 8

COMPARISON WITH OTHER METHODS

8.1: Introduction

In this chapter, we compare the performance of some direct and iterative methods on the model problem. In addition to the CG and PCG methods, we consider sparse symmetric Gaussian elimination and the following iterative methods: successive overrelaxation (SOR), symmetric successive overrelaxation with semi-iteration (SSOR-SI), and the alternating direction implicit (ADI) methods. In Section 2, we present the comparison with iterative methods and in Section 3, with direct methods.

8.2: Comparison with Iterative Methods

Table 8.1 gives the rate of convergence and the work per iteration for the model problem assuming that any parameters (for SOR, SSOR-SI, ADI, and PCG) are optimally chosen. Only the highest order terms in N,

the order of the system being solved, are presented. Moreover, we do not take into account the overhead associated with the LU factorization of tridiagonal matrices in ADI and the Dupont, Kendall, and Rachford factorization of A in the PCG method.

For each of the methods, we assume that the system has been diagonally scaled so that the diagonal elements of the iteration matrix are all unity. In computing matrix-vector products, no advantage is taken of the special values of the non-zeroes in the matrix. Thus, for the model problem, we assume that it takes 4N multiplications to compute a matrix-vector product.

Method	Rate of Convergence	No. of Mults./Iter.
SOR	$O(h)$	5N
SSOR-SI (natural ordering)	$O(h^{1/2})$	12N
ADI (Peaceman-Rachford)	$O((\log h)^{-1})$	12N
CG	$O(h)$	9N
PCG (Dupont, Kendall, & Rachford)	$O(h^{1/2})$	14N

Table 8.1: Convergence rates and multiplications/iteration for the model problem on an n x n mesh ($N = n^2$, $h = 1/(n+1)$)

Mesh-width h	Method	No. of iters.	No. of mults.
1/10	SOR ($\omega = 1.54$)	28	11,340
	SSOR-SI ($\omega = 1.57$)	11	10,692
	ADI (m=5)	7	6,804
	CG	26	18,954
	PCG ($\alpha = 0$)	10	11,340
1/20	SOR ($\omega = 1.74$)	53	95,665
	SSOR-SI ($\omega = 1.76$)	17	73,644
	ADI (m=4)	11	47,652
	CG	53	172,197
	PCG ($\alpha = 0$)	15	75,810
1/40	SOR ($\omega = 1.86$)	117	889,785
	SSOR-SI ($\omega = 1.87$)	24	438,048
	ADI (m=5)	14	255,528
	CG	104	1,423,656
	PCG ($\alpha = 0$)	22	468,468
1/80	SOR ($\omega = 1.93$)	236	7,364,380
	SSOR-SI ($\omega = 1.94$)	34	2,546,328
	ADI (m=5)	18	1,348,056
	CG	201	11,289,969
	PCG ($\alpha = 0$)	34	2,970,716

Table 8.2: Work requirements for iterative methods on the model problem in two dimensions

Table 8.2 gives the work required to solve the model problem as obtained from numerical experiments with the five iterative methods.† In order to have our experiments conform with those in [5] and [21], in each case we took the solution vector to be the zero vector and

† The number of iterations required for ADI and SOR were taken from Birkhoff, Varga, and Young [5] and those for SSOR-SI from Ehrlich [21]. For SOR and SSOR-SI, the numbers correspond to the value of ω that required the fewest number of iterations. For PCG, no attempt was made to choose α optimally; $\alpha = 0$ was used throughout.

terminated the iteration when

$$\frac{\|x_i - x\|_\infty}{\|x_0 - x\|_\infty} \leq 10^{-6}.$$

The experiments in [5] and [21] take the initial approximation x_0 to x to be the unit vector. We did the same for the CG method. For the PCG method, this choice yields the exact solution in one iteration, so x_0 was chosen to be a vector of N random numbers.

The numbers in Table 8.2 lead to the following observations. First, even though SOR and CG both exhibit an O(h) rate of convergence, i.e., if the mesh-width is halved, the number of iterations required approximately doubles, SOR is clearly more efficient than CG in terms of the total number of multiplications required.. However, even with the optimal ω, the number of iterations required for SOR is never less than the number for CG. Furthermore, SOR is quite sensitive to the choice of ω and an inaccurate determination of ω may lead to many more iterations being required [62].

Second, the SSOR-SI and PCG methods are much more efficient than SOR. Also, although the numbers of iterations required by SSOR-SI and PCG are approximately the same, SSOR-SI is slightly more efficient in terms of the total number of multiplications. However, the optimal value of ω was used for SSOR-SI. Ehrlich [21] gives the number of iterations required for $\omega = \omega_1$, where ω_1 is a good approximation to the optimal ω and is predictable from the theory. For $\omega = \omega_1$, the number of iterations required for h = 1/10, 1/20, 1/40, and 1/80 are 19, 29, 43,

and 61 respectively. For this choice of ω, PCG is more efficient than SSOR-SI.

Finally, of the five iterative methods considered, ADI with the optimum parameters is clearly the most efficient for the model problem. However, the theory of ADI methods applies rigorously to only a small class of problems [62].

8.3: Comparison with Direct Methods

The advantages and disadvantages of direct and iterative methods were discussed in Chapter 1. In particular, the CG and PCG methods require much less storage than direct methods and the work per iteration is small. Moreover, the CG method involves no parameters and the PCG method has only a minimal dependence on parameters.

The question here is whether a direct method will be more efficient than the PCG method for solving the model problem. For the PCG method with the appropriate preconditioning, solving the model problem on an n x n mesh requires $O(n^2)$ storage, and $O(n^{1/2} \log n)$ iterations and $O(n^{5/2} \log n)$ work suffices to reduce the initial error by a factor of n^{-p}, $p > 0$. Among direct methods, sparse elimination [36] is more efficient for the model problem than the other variants of Gaussian elimination such as band elimination and profile elimination [26]. Sparse Gaussian elimination with the appropriate ordering requires $O(n^2 \log n)$ storage and $O(n^3)$ work [35]. So as n gets large, PCG

should emerge as the superior method.

Table 8.3 gives the results of numerical experiments to compare the two methods. Neither method took advantage of the actual values of the nonzero elements in the coefficient matrix. In the PCG method, the nonzero diagonals were stored as well as the approximate LU factorization for the Dupont, Kendall, and Rachford preconditioning (see Section 6.4.3) and the four vectors required in the iteration process. The storage required was 13N ($N = n^2$), and 14N multiplications were required per iteration. The initial guess and termination criterion were as described in the last section. No attempt was made to optimize the parameter required in the preconditioning, the value $\alpha = 0$ was used throughout.

The numbers for sparse elimination were obtained from Eisenstat [23]. The storage required includes overhead for pointer storage, and the work includes the work for computing the factorization and doing the backsolve. The near-optimal digonal nested dissection ordering [35] was used.

Mesh-width h	Sparse Symmetric Gaussian Elimination		PCG		
	Storage	No. of mults.	Storage (13N)	No. of iters.	Total no. of mults.
1/10	1,083	2,946	1,053	10	11,340
1/20	6,340	30,353	4,693	15	75,810
1/40	33,075	282,416	19,773	22	468,468
1/80	161,218	2,449,755	81,133	34	2,970,716
1/112	337,515	6,878,995	160,173	40	6,899,760
1/113	344,695	7,099,776	163,072	40	7,024,640
1/114	351,649	7,275,868	165,997	40	7,150,640

Table 8.3: Storage and work required for Sparse Symmetric Gaussian Elimination (diagonal nested-dissection ordering) and PCG (Dupont, Kendall and Rachford preconditioning with $\alpha = 0$)

For coarse mesh-widths, sparse elimination is better in terms of the total work required. As the mesh gets finer, the difference between the two methods narrows. For $h < 1/112$, the PCG method requires less work. In terms of storage, the PCG method is by far the more efficient, even for coarse meshes. For small h, sparse elimination requires considerably more storage, and at $h = 1/114$, sparse elimination requires 352K, more than double that required by PCG.

For the three-dimensional model problem on an n x n x n mesh, the PCG method requires $O(n^3)$ storage, and $O(n^{1/2} \log n)$ iterations and $O(n^{7/2} \log n)$ work suffices to reduce the initial error by a factor of n^{-p}, for $p > 0$. For the same problem, sparse elimination requires $O(n^4)$ storage and $O(n^6)$ work for any ordering [24]. Thus, except for small problems, sparse elimination should not be used in three dimensions.

PART III

THE SYMMETRIC INDEFINITE PROBLEM

CHAPTER 9

SOME METHODS FOR THE INDEFINITE PROBLEM

9.1 Introduction

To consider the problem of solving the linear system

$$Ax = f,$$

where A is an N x N, symmetric, and nonsingular matrix. The eigenvalues of A are real but they may be positive or negative since A is not necessarily positive definite. In each iteration of the conjugate gradient method (Algorithm 4.2), the scalar $a_i = (r_i,r_i)/(p_i,Ap_i)$ is computed. Unfortunately, if A is indefinite, the denominator (p_i,Ap_i) can vanish. Thus the CG method may break down if used directly in an attempt to solve the indefinite problem.

A common approach is to transform (9.1) into an equivalent system whose solution is the same and which has a symmetric positive definite coefficient matrix. The CG method is then used to solve this new system. Two ways to do this are the CG method applied to the normal

equations (see [39] and Section 10.2) and Craig's method [29]. However, since this causes a deterioration in the condition number of the original problem, these methods converge slowly.

In this Chapter, we present two known methods based on the CG method, for solving the indefinite problem: the SYMMLQ method of Paige and Saunders [52] and the method of hyperbolic pairs of Luenberger [46]. We also present a new method, which we call SYMMBK, which uses a factorization procedure of Bunch and Kaufman [8]. These three methods can be viewed as extensions of the CG method to indefinite problems. Like the CG method, these methods generate sequences of iterates $x_0,x_1,\ldots$ such that x_{i+1} corresponds to a stationary point of the error functional $(x-\bar{x},A(x-\bar{x}))$ in the subspace $\{r_0,Ar_0,\ldots,A^i r_0\}$. When A is positive definite, the stationary point given by the CG method is actually the point at which the error functional is minimized in the subspace. This fact leads to error bounds for CG (see Sections 3.3 and 4.4). However, when A is not positive definite, we know of no results about their rates of convergence, apart from the fact that assuming exact arithmecic, these methods give the solution in at most N iterations.

In Section 2, we develop the basis for the SYMMBK and SYMMLQ algorithms. In Section 3, we outline the Bunch and Kaufman factorization procedure that leads to the SYMMBK algorithm which we present in Section 4. In Section 5, we briefly describe SYMMLQ and show that it is mathematically equivalent to SYMMBK. Since SYMMBK requires

less work per iteration, we recommend it over SYMMLQ. In Section 6, we present Luenberger's method of hyperbolic pairs which is unstable in the presence of roundoff error and show that it is mathematically equivalent to SYMMLQ and SYMMBK under the assumption of exact arithmetic. In Section 7, we present results of numerical experiments on the indefinite model problem.

9.2: Development of SYMMBK and SYMMLQ

The development in this section follows Paige and Saunders [52]. Given an initial approximation x_0 to x and a set of linearly independent vectors $v_1, v_2, \ldots$, we consider computing successive *approximations* $x_1, x_2, \ldots$ to x of the form

$$x_k = x_0 + \sum_{i=1}^{k} \xi_{k,i} v_i,$$

where the $\xi_{i,m}$ are scalars. If we define

$$V_k \equiv [v_1, v_2, \ldots v_k] \quad \text{and} \quad y_k = (\xi_{k,1}, \xi_{k,2}, \ldots \xi_{k,k})^T,$$

then

$$x_k = x_0 + V_k y_k$$

and the functional

$$f_k(y_k) = (x - x_k, A(x - x_k)) \tag{9.6}$$
$$= (x - x_0 - V_k y_k, A(x - x_0 - V_k y_k))$$

has a stationary value at y_k iff

$$V_k^T A V_k y_k = V_k^T A(x - x_0),$$

i.e.,

$$V_k^T A V_k \, y_k = V_k^T r_0 \tag{9.7}$$

or

$$V_k^T r_k = \underline{0} \tag{9.8}$$

where $r_k \equiv f - Ax_k$. Since the v_i are linearly independent this clearly implies that for some $m \leq N$ we have $r_m = \underline{0}$ which implies $x_m = x$.

The Lanczos process [43] is a computationally efficient way of generating linearly independent vectors, which will lead to a computationally tractable linear system (9.7). Here we use a variant [50] to generate the v_i's.

Algorithm 9.1: The Lanczos Method for Generation of Orthonormal Vectors

Step 1: Choose an initial guess x_0 to x.

Compute $r_0 = f - Ax_0$.

Set $v_0 = \underline{0}$.

Compute $\beta_1 = \| r_0 \|_2$,

and $v_1 = r_0/\beta_1$. (9.9)

Set $j = 1$.

Step 2: Compute

$$\alpha_j = (Av_j, v_j),$$
$$v'_{j+1} = Av_j - \alpha_j v_j - \beta_j v_{j-1},$$
$$\beta_{j+1} = \| v'_{j+1} \|_2,$$

and $v_{j+1} = v'_{j+1}/\beta_{j+1}$.

Step 3: Set j = j+1.

Go to Step 2.

Note that $(v_j, v_j) = 1$ for every j. The process will be terminated at the first zero β_j, so from now on we assume that $\beta_j \neq 0$.

After k iterations of Algorithm 9.1, we have

$$AV_k = V_k T_k + \beta_{k+1} v_{k+1} e_k^T, \tag{9.10}$$

where

$$V_k^T V_k = I \equiv [e_1, e_2, \ldots, e_k], \quad V_k^T v_{k+1} = \underline{0}, \tag{9.11}$$

and T_k is the tridiagonal matrix

$$\begin{bmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \ddots & & \\ & \ddots & \ddots & \beta_k & \\ & & \beta_k & \alpha_k & \end{bmatrix}. \tag{9.12}$$

Multiplying (9.10) by V_k^T and using (9.11), we have $V_k^T A V_k = T_k$. From (9.9) and (9.11), $V_k^T r_0 = \beta_1 e_1$, so that (9.7) may be written as

$$T_k y_k = \beta_1 e_1 \tag{9.13a}$$

and

$$x_k = x_0 + V_k y_k \tag{9.13b}$$

Thus to solve (9.1) we successively generate vectors v_i by Algorithm 9.1

and solve (9.13) to obtain approximations to x.

Let us consider the special case when A is positive definite. Then, $T_k = V_k^T A V_k$ is also positive definite and the Cholesky decomposition $L_k D_k L_k^T$ of T_k, where L_k is unit lower bidiagonal and D_k is diagonal, exists and is numerically stable. Also, since T_{k+1} has T_k as its principal k x k submatrix, the decomposition for T_{k+1} can be developed easily from that of T_k. The x_k obtained as a solution to (9.13) yields the unique minimum of $(x - \tilde{x}, A(x - \tilde{x}))$ over all $\tilde{x}$ of the form $x_0 + q$ where $q \in S \equiv \{v_1, v_2, \ldots, v_k\}$. Now, since the v's are generated by Algorithm 9.1, they are vectors derived from a Krylov sequence [40] so that $S = \{v_1, Av_1, \ldots, A^{k-1}v_1\}$. Using the definition of v_1 from (9.9) we have that the x_k obtained from (9.13) uniquely minimizes $(x - \tilde{x}, A(x - \tilde{x}))$ over all $\tilde{x}$ of the form $x_0 + q$ where $q \in \{r_0, Ar_0, \ldots, A^{k-1}r_0\}$. In Section 4.4 we pointed out that the k^{th} iterate x_k^{CG} of the conjugate gradient algorithm has the same property. Hence, for the same initial guess x_0, $x_k = x_k^{CG}$. We thus have the following equivalence result.

Theorem 9.1: ([52]) If A is symmetric and positive definite, the solution x_k of (9.13) is mathematically equivalent to the approximation to x obtained at the k^{th} iteration of the conjugate gradient method.

If A is not positive definite, then, it is possible for some T_k to be singular or not have a Cholesky decomposition even if T_N is nonsingular. Moreover, if T_k is singular, the Cholesky decomposition of

subsequent T_j's, $j > k$, may be numerically unreliable. Hence, when A is indefinite, alternatives to the Cholesky decomposition have to be found in order to be able to solve (9.13). In the following sections we present two algorithms resulting from different decompositions of the T_k.

9.3: The Bunch and Kaufman Decomposition

In this section, we describe the use of a decomposition procedure due to Bunch and Kaufman [8] for solving (9.13). We start by observing that for any $k > 0$, T_k and T_{k+1} cannot both be singular. In fact if T_k and T_{k+1} were both singular, the theory of determinants says that all T_j's, $j \geq k$ would be singular [40]. Clearly, this is not possible since T_N is similar to A which is nonsingular. Hence, if T_{k+1} is singular, both T_k and T_{k+2} are nonsingular.

Suppose that we were computing the Cholesky decompositions of successive T_j's and that all the T_j's, $j \leq k$ were nonsingular. If T_{k+1} were singular, this would imply that the $(k+1)^{st}$ pivot element became zero and caused the break down of the decomposition process [8]. However, if we were to take a 2 x 2 pivot instead of a single element, we would have

$$\text{pivot} = \begin{bmatrix} 0 & \beta_{k+2} \\ \beta_{k+2} & \alpha_{k+2} \end{bmatrix}$$

and its determinant is $-\beta_{k+2}^2$ which would not be zero. Using this pivot yields the decomposition of T_{k+2} directly from that of T_k, both of which are nonsingular, without going through the singular T_{k+1}. The matrix D_k is no longer diagonal but is block diagonal with diagonal blocks of order either 1 or 2. The resulting decomposition can be written as

$$T_k = M_k D_k M_k^T, \tag{9.14}$$

where D_k is as defined above and M_k is a unit lower triangular matrix such that its non-zeroes are restricted to the diagonal and the two preceding subdiagonals and $M_{i+1,i} = 0$ iff $D_{i+1,i} \neq 0$.

To maintain numerical stability we must prevent large growth in the elements of the reduced matrices M_k and D_k generated during the decomposition process. We do this by using 2 x 2 pivots when the pivot element is small as well as zero.

Let T be any N x N nonsingular symmetric tridiagonal matrix. The Bunch and Kaufman procedure decomposes T into the product MDM^T where M and D are as defined above, by generating a sequence of tridiagonal matrices $T^{(k)}$ of order k, called reduced matrices. We show the first step which is typical. Without loss of generality we may assume that T is irreducible so that $T_{21} \neq 0$.

If $\max_{1 \leq j \leq N} |T_{1j}| \leq \delta$, let $\gamma > 0$ be such that $\gamma\delta < 1$.

If $|T_{11}| > \gamma T_{21}^2$, then T_{11} is used as a 1 x 1 pivot to generate $T^{(N-1)}$ and

$$T = \begin{bmatrix} 1 & 0 \cdots 0 \\ M_{21} & \\ 0 & I \\ \vdots & \\ 0 & \end{bmatrix} \begin{bmatrix} T_{11} & 0 \cdots 0 \\ 0 & \\ \vdots & T^{(N-1)} \\ 0 & \end{bmatrix} \begin{bmatrix} 1 & M_{21}\ 0 \cdots 0 \\ 0 & \\ \vdots & I \\ 0 & \end{bmatrix} .$$

If $|T_{11}| \leq \gamma T_{21}^2$, then $\begin{bmatrix} T_{11} & T_{21} \\ T_{12} & T_{22} \end{bmatrix}$ is used as a 2 x 2 pivot and

$$T = \begin{bmatrix} 1 & 0 & & \\ 0 & 1 & 0 & \\ M_{31} & M_{32} & & \\ 0 & & I & \\ \vdots & & & \\ 0 & & & \end{bmatrix} \begin{bmatrix} T_{11} & T_{21} & & \\ T_{12} & T_{22} & 0 & \\ 0 & & T^{(N-2)} & \\ \vdots & & & \\ 0 & & & \end{bmatrix} \begin{bmatrix} 1 & 0 & M_{31} & \\ 0 & 1 & M_{32} & 0 \\ 0 & & I & \\ \vdots & & & \\ 0 & & & \end{bmatrix}$$

It was shown in [8] that in both cases the reduced matrices are tridiagonal and hence the same procedure can be applied recursively. Bunch [7] gave the following bound on the growth of elements in the reduced matrices.

Lemma 9.1:

If $\max_{1\leq i,j\leq N} |T_{ij}| \leq \delta$ and $\gamma \equiv (\sqrt{5} - 1)/(2\delta)$,

then $\max_{1\leq i,j\leq k} |T_{ij}^{(k)}| \leq (3 + \sqrt{5})\,\delta/2.$

We now prove a result which we will use to bound element growth for the SYMMBK algorithm that uses this factorization.

Theorem 9.2: Let $\{\lambda_i\}_{i=1}^N$ be the eigenvalues of the matrix A and let $\max_i |\lambda_i| \leq \theta$. The choice

$$\gamma = (\sqrt{5} - 1)/(2\theta)$$

gives element growth in the Bunch and Kaufmann procedure that is bounded by $(3 + \sqrt{5})\theta/2$.

Proof: Since $(v_j, v_j) = 1$, $1 \leq j \leq N$, we have

$$\alpha_j = (v_j, Av_j) \leq \max_i |\lambda_i|. \tag{9.15}$$

From Algorithm 9.1, we have

$$\beta_{j+1} v_{j+1} = Av_j - \alpha_j v_j - \beta_j v_{j-1},$$

Taking the inner product with v_{j+1} and using the facts that $(v_{j+1}, v_i) = 0$, $i \neq j+1$, and $(v_{j+1}, v_{j+1}) = 1$, we have

$$\beta_{j+1} = (Av_j, v_{j+1}).$$

Hence by the Cauchy Schwartz inequality,

$$\begin{aligned} |\beta_{j+1}| = |(Av_j, v_{j+1})| &\leq (Av_j, Av_j)^{1/2} (v_{j+1}, v_{j+1})^{1/2} \\ &\leq (\max_i |\lambda_i^2|)^{1/2} \\ &= \max_i |\lambda_i|. \end{aligned} \tag{9.16}$$

From (9.15) and (9.16) we have

$$\max_{1\leq j\leq N} |\alpha_j|, \quad \max_{1\leq j\leq N-1} |\beta_j| \leq \max_i |\lambda_i| \leq \theta .$$

Applying Lemma 9.1, the Theorem follows.

Q.E.D.

9.4: SYMMBK

In this section we present an algorithm for the indefinite problem (9.1) that combines the development of Section 2 and the factorization of Section 3. We call the resulting algorithm SYMMBK, since it uses the Symmetric Bunch and Kaufman factorization.

As noted in the preceding section, we will not be able to obtain x_k for all k. If T_{k+1} is singular, or nearly so, we obtain factorizations for T_k and then T_{k+2} so that x_{k+1} is not computed. We also note that even though the decompositions are easily obtainable from the preceeding one, the vector y_k changes completely as k increases and cannot be accumulated. This would imply that to solve (9.13b) all the vectors v_i forming V_k would have to be available in order to compute x_k.

However, there is an easy way around this. Let us suppose that we have a decomposition $T_k = M_k D_k M_k^T$. We do not compute y_k. Instead we define $p_k \equiv M_k^T y_k$ and $C_k \equiv V_k M_k^{-T}$ and rewrite equations (9.13) as

$$M_k D_k p_k = \beta_1 e_1, \tag{9.17}$$

$$M_k C_k^T = V_k^T, \tag{9.18}$$

and

$$x_k = x_0 + C_k p_k, \tag{9.19}$$

where $p_k = (\eta_{k,1}, \eta_{k,2}, \ldots \eta_{k,k})^T$ is a vector of length k, V_k and $C_k = [c_1^{(k)}, c_2^{(k)}, \ldots c_k^{(k)}]$ are N x k matrices, and D_k and M_k are k x k matrices. We will show that if x_{k-1} was computed (i.e., if the present pivot is 1 x 1), p_{k-1} consists of the first (k-1) elements of p_k, and C_{k-1} is the first (k-1) columns of C_k. Hence, x_k is the sum of the two vectors x_{k-1} and $\eta_{k,k} c_k^{(k)}$. Similarly, if the present pivot is 2 x 2, p_k differs from p_{k-2} in its last two elements, C_k is just C_{k-2} concatenated with two additional columns $c_{k-1}^{(k)}$ and $c_k^{(k)}$ so that x_k can be obtained easily as the sum of x_{k-1}, $\eta_{k,k-1} c_{k-1}^{(k)}$ and $\eta_{k,k} c_k^{(k)}$. We now outline the procedure to solve (9.17)-(9.19).

Case 1: Suppose we have decided that the present pivot is to be 1 x 1. Then in general (9.17) can be written as

$$\left[\begin{array}{c|c} M_{k-1} & 0 \\ \hline 0 \cdots 0 \;\; m_1 \;\; m_2 & 1 \end{array}\right] \left[\begin{array}{c|c} D_{k-1} & 0 \\ \begin{array}{cc} d_1 & d_3 \\ d_2 & d_4 \end{array} & \\ \hline 0 & \alpha \end{array}\right] \left[\begin{array}{c} \eta_{k,1} \\ \eta_{k,2} \\ \vdots \\ \eta_{k,k-1} \\ \hline \eta_{k,k} \end{array}\right] = \left[\begin{array}{c} \beta_1 \\ 0 \\ \vdots \\ 0 \\ \hline 0 \end{array}\right], \tag{9.20}$$

where the α, m_i and d_i are known from the last iteration. If the last pivot was 1 x 1 then $m_1 = d_2 = d_3 = 0$.

Multiplying out the left-hand side of (9.20) we get

(9.21)
$$\left[\begin{array}{c|c} M_{k-1}D_{k-1} & 0 \\ \hline 0 \text{—} 0 & \alpha \end{array}\right] \left[\begin{array}{c} \eta_{k,1} \\ \eta_{k,2} \\ \vdots \\ \eta_{k,k-1} \\ \hline \eta_{k,k} \end{array}\right] = \left[\begin{array}{c} \beta_1 \\ 0 \\ \vdots \\ 0 \\ \hline 0 \end{array}\right]$$

$m_1d_1+m_2d_2$ $\quad$ $m_1d_3+m_2d_4$

Since we already have $M_{k-1}\, D_{k-1}\, P_{k-1} = \beta_1 e_1$, from (9.21) we get that

(9.22a) $\quad \eta_{k,i} = \eta_{k-1,i} \qquad i = 1,2, \ldots k-1$

and

(9.22b) $\quad \eta_{k,k} = -\left[(m_1d_1 + m_2d_2)\, \eta_{k,k-2} + (m_1d_3 + m_2d_4)\eta_{k,k-1}\right] / \alpha.$

Likewise (9.18) can be written as

(9.23)
$$\left[\begin{array}{c|c} M_{k-1} & 0 \\ \hline 0 \text{—} 0\ m_1\ m_2 & 1 \end{array}\right] \left[\begin{array}{c} c_1^{(k)T} \\ \vdots \\ c_{k-1}^{(k)T} \\ \hline c_k^{(k)T} \end{array}\right] = \left[\begin{array}{c} v_1^T \\ \vdots \\ v_{k-1}^T \\ \hline v_k^T \end{array}\right]$$

Since $M_{k-1}\, C_{k-1}^T = V_{k-1}^T$, from (9.23) we get that

(9.24a) $\quad c_i^{(k)} = c_i^{(k-1)}, \quad i = 1, 2, \ldots k-1,$

and

(9.24b) $\quad c_k^{(k)} = v_k - m_1 c_{k-2}^{(k)} - m_2 c_{k-1}^{(k)}$

Using (9.22a) and (9.24a), we get from (9.19) that

(9.25) $\quad x_k = x_0 + \sum_{i=1}^{k} \eta_{k,i}\, c_i^{(k)} = x_{k-1} + \eta_{k,k}\, c_k^{(k)}.$

Thus if the pivot is chosen to be 1 x 1, (9.22), (9.24) and (9.25) may be used to compute x_k from x_{k-1}.

Case 2: If the pivot chosen is 2 x 2, we can represent

$$M_k = \left[\begin{array}{c|cc} M_{k-2} & 0 & \\ \hline 0 \text{—} 0\ m_1\ m_2 & 1 & 0 \\ 0 \text{———} 0 & 0 & 1 \end{array}\right] \quad \text{and} \quad D_k = \left[\begin{array}{cc|cc} D_{k-2} & & 0 & \\ & & d_1 & d_3 \\ & & d_2 & d_4 \\ \hline & & \alpha & \beta \\ 0 & & \beta & \tilde{\alpha} \end{array}\right],$$

where the m_i, d_i, α, $\tilde{\alpha}$ and β are known. If the last pivot was 1 x 1, then $m_1 = d_2 = d_3 = 0$, and

$$\left[\begin{array}{c|cc} M_{k-2}D_{k-2} & 0 & \\ \hline 0 \text{—} 0 & \alpha & \beta \\ 0 \text{———} 0 & \beta & \tilde{\alpha} \end{array}\right] \left[\begin{array}{c} \eta_{k,1} \\ \vdots \\ \eta_{k,k-2} \\ \hline \eta_{k,k-1} \\ \eta_{k,k} \end{array}\right] = \left[\begin{array}{c} \beta_1 \\ 0 \\ \vdots \\ 0 \\ \hline 0 \\ 0 \end{array}\right]$$

$m_1d_1 + m_2d_2$ $\quad$ $m_1d_3 + m_2d_4$

Since $M_{k-2}\, D_{k-2}\, P_{k-2} = \beta_1 e_1$, it follows using Cramer's rule that

(9.26a) $\quad \eta_{k,i} = \eta_{k-1,i}, \quad i = 1, 2, \ldots, k-2,$

(9.26b) $\quad \eta_{k,k-1} = -\alpha\, s/\det,$

and

(9.26c) $\quad \eta_{k,k} = \beta\, s/\det$

where $\det \equiv \begin{vmatrix} \alpha & \beta \\ \beta & \tilde{\alpha} \end{vmatrix} = \alpha\tilde{\alpha} - \beta^2$

and $s \equiv (m_1 d_1 + m_2 d_2)\, \eta_{k,k-3} + (m_1 d_3 + m_2 d_4)\, \eta_{k,k-2}$.

Likewise, (9.18) can be written as

(9.27)
$$\left[\begin{array}{c|c} M_{k-2} & 0 \\ \hline 0 \text{—} 0\ m_1\ m_2 & 1\ 0 \\ 0 \text{———} 0 & 0\ 1 \end{array}\right] \left[\begin{array}{c} c_1^{(k)T} \\ \vdots \\ c_{k-2}^{(k)T} \\ \hline c_{k-1}^{(k)T} \\ c_k^{(k)T} \end{array}\right] = \left[\begin{array}{c} v_1^T \\ \vdots \\ v_{k-2}^T \\ \hline v_{k-1}^T \\ v_k^T \end{array}\right]$$

Again, since $M_{k-2}\, C_{k-2}^T = V_{k-2}^T$, from (9.27) we get

(9.28a) $$c_i^{(k)} = c_i^{(k-2)}, \quad i = 1, 2, \ldots, k-2,$$

(9.28b) $$c_{k-1}^{(k)} = v_{k-1} - m_1 c_{k-3}^{(k)} - m_2 c_{k-2}^{(k)},$$

and

(9.28c) $$c_k^{(k)} = v_k.$$

Thus x_k is given by

$$x_k = x_0 + \sum_{i=1}^{k} \eta_{k,i}\, c_i^{(k)}$$

(9.29) $$= x_{k-2} + \eta_{k,k-1}\, c_{k-1}^{(k)} + \eta_{k,k}\, c_k^{(k)}$$

So, for a 2 x 2 pivot, (9.26) - (9.29) may be used to get x_k from x_{k-2}. Since the $c_m^{(i)}$ are only defined for $i \geq m$ and $c_m^{(i)} = c_m^{(j)}$ $\forall$ $i,j \geq m$, from now on we suppress the superscripts on the c's.

The SYMMBK algorithm may be summarized as follows. To minimize element growth in the SYMMBK algorithm γ is chosen as in Theorem 9.2.

Algorithm 9.2: The SYMMBK Method

Step 1: Choose x_0 and γ.

Compute

$r_0 = f - Ax_0$,

$\beta_1 = \| r_0 \|_2$,

$v_1 = r_0/\beta_1$,

$\alpha_1 = (Av_1, v_1)$,

$v_2' = Av_1 - \alpha_1 v_1$,

$\beta_2 = \| v_2' \|_2$,

and $v_2 = v_2'/\beta_2$.

Set $c_1 = c_2 = \underline{0}$,

$m_1 = m_2 = 0$,

$s = -\beta_1$,

and $k = 0$.

Go to Step 3.

Step 2: Compute

$\alpha_{k+1} = (Av_{k+1}, v_{k+1})$,

$v_{k+2}' = Av_{k+1} - \alpha_{k+1} v_{k+1} - \beta_{k+1} v_k$,

$\beta_{k+2} = \| v_{k+2}' \|_2$,

and $v_{k+2} = v_{k+2}'/\beta_{k+2}$.

Set $\alpha_{k+1} = \alpha_{k+1} - m_2 \beta_{k+1}$.

Step 3: If $\alpha_{k+1} > \gamma\beta_{k+2}^2$ go to Step 4a;
else go to Step 4b.

Step 4a: (1 x 1 pivot)

Set $k = k + 1$.

Compute

$\eta_{k,k} = -s/\alpha_k$,

$c_k = v_k - m_1 c_{k-2} - m_2 c_{k-1}$,

$x_k = x_{k-1} + \eta_{k,k} c_k$,

and $m_2 = \beta_{k+1}/\alpha_k$.

Set $m_1 = d_2 = d_3 = 0$

and $d_4 = \alpha_k$.

Compute $s = m_2 d_4 \eta_{k,k}$.

Go to Step 5.

Step 4b: (2 x 2 pivot)

Set $k = k + 2$.

Compute

$\alpha_k = (Av_k, v_k)$,

$\det = \alpha_{k-1}\alpha_k - \beta_k^2$,

$\eta_{k,k-1} = -\alpha_k\,(s/\det)$,

$\eta_{k,k} = \beta_k(s/\det)$,

$c_{k-1} = v_{k-1} - m_1 c_{k-3} - m_2 c_{k-2}$,

$c_k = v_k$,

$x_k = x_{k-2} + \eta_{k,k-1} c_{k-1} + \eta_{k,k} c_k$,

$v'_{k+1} = Av_k - \alpha_k v_k - \beta_k v_{k-1}$,

$\beta_{k+1} = \| v'_{k+1} \|_2$,

$v_{k+1} = v'_{k+1}/\beta_{k+1}$,

$m_1 = -\beta_k\,(\beta_{k+1}/\det)$,

and $m_2 = \alpha_{k-1}(\beta_{k+1}/\det)$.

Set $d_1 = \alpha_{k-1}$,

$d_2 = d_3 = \beta_k$,

and $d_4 = \alpha_k$.

Compute $s = (m_1 d_1 + m_2 d_2)\,\eta_{k,k-1} + (m_1 d_3 + m_2 d_4)\eta_{k,k}$.

Step 5: If x_{i+1} is sufficiently close to x, terminate the iteration process;
else go to Step 2.

In any application of the SYMMBK method we would like to monitor the size of the error in the approximants x_k's. This information may be necessary to decide when to stop the iteration procedure. Even though the residual is not explicitly computed in the SYMMBK algorithm, $\| r_k \|_2$ is easily obtained. In fact

$$
\begin{aligned}
r_k &= f - Ax_k \\
&= f - A(x_0 + V_k y_k) \quad \text{by} \quad (9.13b) \\
&= \beta_1 v_1 - AV_k y_k \quad \text{by (9.9)} \\
&= \beta_1 v_1 - V_k T_k y_k - \beta_{k+1} v_{k+1} e_k^T y_k \quad \text{by (9.10)} \\
&= -\beta_{k+1}\,\xi_{k,k}\,v_{k+1} \quad \text{by (9.13a)},
\end{aligned}
\tag{9.30}
$$

where $\xi_{k,k}$ is the k^{th} element of the vector y_k.
Thus $\| r_k \|_2 = |\beta_{k+1}\xi_{k,k}|$ since $(v_{k+1}, v_{k+1}) = 1$.

Since $p_k = M_k^T y_k$ and M_k^T is unit upper triangular,

$$\eta_{k,k} = \xi_{k,k},$$

where $\eta_{k,k}$ is the k^{th} (and last) element of the vector p_k. Hence

$$\| r_k \|_2 = |\beta_{k+1} \eta_{k,k}|.$$

Thus $\| r_k \|_2$ is directly available and can be used to decide when to terminate the iteration process.

Algorithm 9.2 requires storage for the six vectors x, Av, the two most recent v's and the two most recent c's, a few scalars, and the matrix A. The symmetric matrix A does not get modified during the iteration process and is only used to compute a matrix-vector product.

The work estimates for SYMMBK for the model and generalized model problems are given in Table 9.1. We observe that given β_k and v_k, computing β_{k+1} and v_{k+1} requires one matrix-vector product and 5N multiplications, which is the cost associated with Step 2 of the above algorithm. Step 4a requires essentially 2N multiplications if the preceeding pivot was 1 x 1 (since $m_1 = 0$), and 3N multiplications if the preceding pivot was 2 x 2. Step 4b requires 8N multiplications if the last pivot was 1 x 1 and 9N multiplications if the last pivot was 2 x 2. Hence the cost associated with an iteration requiring a 2 x 2 pivot is 14N, twice that for one requiring a 1 x 1 pivot. Since a 2 x 2 pivot gets x_{k+2} directly from x_k it really constitutes a double iteration and we can thus say that each iteration of SYMMBK requires essentially a matrix-vector product plus 7N multiplications.

We'd like to mention here that if A is positive definite and not too ill-conditioned, we can always choose single element pivots in the SYMMBK method and have a numerically stable procedure. The resulting decomposition of T_k is simply the Cholesky decomposition where M_k is unit lower bidiagonal and D_k is a diagonal matrix. We also know (Theorem 9.1) that the iterates obtained by the SYMMBK method are the same as those for the CG method. Since the CG method requires less work per iteration, the CG method rather than the SYMMBK method should be used for problems that are positive definite and reasonably well-conditioned.

We now prove a few results that will be used in Section 6 to show the relation between SYMMBK and Luenberger's method.

For every k, let us define, the subspaces

$$\bar{V}_k = \{v_1, v_2, \ldots, v_k\}, \tag{9.31a}$$

$$\bar{C}_k = \{c_1, c_2, \ldots, c_k\}, \tag{9.31b}$$

and

$$\bar{S}_k = \{v_1, Av_1, A^2v_1, \ldots, A^{k-1}v_1\}. \tag{9.31c}$$

We have the following preliminary results.

Lemma 9.2: For every k, $\bar{S}_k = \bar{V}_k = \bar{C}_k$.

Proof: By the definition of the v_i's we have $\bar{V}_k \subseteq \bar{S}_k$, for every k. Moreover, $\bar{V}_k = \bar{S}_k$ since the v_i's are linearly independent.

For the second equality, we note that from (9.24) and (9.28) we can write for each k

$$v_k \in \{c_k, c_{k-1}, c_{k-2}\}$$

so that $\bar{V}_k \subseteq \bar{C}_k$.

Again, $\bar{V}_k = \bar{C}_k$ since the v_i's are linearly independent.

Q.E.D.

Lemma 9.3: For every k, the following identities hold:

(a) $(c_{k+1}, Au) = 0 \quad \forall u \in \bar{C}_{k-1}$;

(b) if x_k exists, then $(c_{k+1}, Au) = 0 \quad \forall u \in \bar{C}_k$;

(c) if $\gamma = 0$ and x_k exists, then x_{k+1} exists iff $(c_{k+1}, Ac_{k+1}) \neq 0$.

Proof: Let ℓ be an integer such that $\ell > k$ and x_ℓ exists. In particular ℓ can be chosen to be k+1 if a 1 x 1 pivot is used and k+2 if a 2 x 2 pivot is used. Using (9.18), we have

$$\begin{aligned} C_\ell^T A C_\ell &= (M_\ell^{-1} V_\ell^T) A (V_\ell M_\ell^{-T}) \\ &= M_\ell^{-1} T_\ell M_\ell^{-T} \\ &= D_\ell. \end{aligned}$$

Thus if $d_{i,j}$ is the $(i,j)^{th}$ element of D_ℓ,

$$d_{i,j} = (c_i, Ac_j), \quad i,j = 1,2,\ldots\ldots \ell. \tag{9.32}$$

Since D_ℓ has diagonal blocks of order 1 or 2 only,

$$d_{i,j} = 0, \quad |i-j| > 1,$$

which implies that

$$(c_i, Ac_j) = 0, \quad j < i-1.$$

For i = k+1 this gives

$$(c_{k+1}, Ac_j) = 0, \quad j < k,$$

which proves (a).

Now if possible let $d_{k+1,k} \neq 0$. Then, $\begin{bmatrix} d_{k,k} & d_{k,k+1} \\ d_{k+1,k} & d_{k+1,k+1} \end{bmatrix}$ is pivot so that x_k does not exist. Hence, if x_k exists, $d_{k+1,k} = 0$. Then from (9.32) we have (c_{k+1}, Ac_k) which combined with (a) yields (b).

To prove (c), let $\gamma = 0$ and assume x_k exists. Then x_{k+1} exists iff a 1 x 1 pivot is chosen and such a pivot exists iff $d_{k+1,k+1} \neq 0$.

But $d_{k+1,k+1} = (c_{k+1}, Ac_{k+1})$.

Hence, x_{k+1} exists iff $(c_{k+1}, Ac_{k+1}) \neq 0$, which proves (c).

Q.E.D.

9.5: SYMMLQ

We now present the SYMMLQ method that uses an alternative decomposition to solve the equations (9.13). SYMMLQ was proposed by Paige and Saunders [52]. In this section, we briefly outline the method

and show that it is less efficient than SYMMBK.

Following the development of Section 2, equations (9.13) are solved by using the orthogonal factorization

$$T_k = \bar{L}_k Q_k, \tag{9.33}$$

where $\bar{L}_k$, Q_k are k x k matrices, $Q_k^T Q_k = I$ and $\bar{L}_k$ is lower triangular with all the non-zeros restricted to the diagonal and the two adjacent sub-diagonals. The bar on the L_k denotes that $\bar{L}_k$ differs from the leading part of $\bar{L}_{k+1}$ in its (k,k) element only.

If we define $\bar{z}_k \equiv Q_k y_k$ and $\bar{W}_k \equiv V_k Q_k^T$, we find that equations (9.13) are equivalent to

$$\bar{L}_k \bar{z}_k = \beta_1 e_1 \tag{9.34a}$$

and

$$x_k = x_0 + \bar{W}_k \bar{z}_k \tag{9.34b}$$

where the η_i are scalars and the $\bar{z}_k \equiv (\eta_1, \eta_2, \ldots \eta_{k-1}, \bar{\eta}_k)^T$, and w_i are N-vectors, and $\bar{W}_k \equiv [w_1, w_2, \ldots w_{k-1}, \bar{w}_k]$.

The factorization is numerically stable even when T_k is indefinite and it can be shown that the v_i and w_i can be formed, used, and discarded one by one. However, when T_k is singular, then so is $\bar{L}_k$, so that $\bar{z}_k$ is undefined. The algorithm is therefore not implemented as shown in (9.34). Instead, we solve

$$L_k z_k = \beta_1 e_1 \tag{9.35a}$$

and

$$x_k^L = x_0 + W_k z_k, \tag{9.35b}$$

where $z_k \equiv (\eta_1, \eta_2, \ldots \eta_{k-1}, \eta_k)^T$, $W_k \equiv [w_1, w_2, \ldots w_{k-1}, w_k]$ and L_k denotes the k x k principal submatrix of $\bar{L}_{k+1}$.

It can be shown, cf.,[52], that L_k is never singular so that z_k is always well-defined, and that every iteration using (9.35) requires essentially N fewer multiplications than (9.34). However, the approximation x_k^L is different from x_k and is usually not as accurate an approximation to x. Even though x_k is not obtained at each iteration, r_k is available, on the basis of which a decision can be made of when to terminate the iteration procedure [52]. If at the last iteration $\|r_k\|_2 < \|r_k^L\|_2$ where $r_k^L = f - Ax_k^L$, x_k can be computed from x_k^L in N multiplications (see [52]).

This method requires storage for A and the five vectors x, w, Av, and the two most recent v's. Solving (9.35) requires approximately 4N multiplications. The v's are generated by Algorithm 9.1 so that the total cost per iteration is one matrix-vector product plus 9N multiplications. An extra N multiplications are required at the last step if x_k is obtained from x_k^L.

Now SYMMBK and SYMMLQ both solve equations (9.13). In the absence of roundoff error, when x_k is obtained by SYMMBK it can be obtained from the x_k^L of SYMMLQ, and the two methods require the same number of iterations. SYMMBK requires one extra vector of storage but it requires

2N fewer multiplications per iteration as compared to SYMMLQ, so that in terms of total work required, SYMMBK is more efficient than SYMMLQ. In Section 7, we give numerical results comparing the two algorithms on indefinite model problems. Fletcher [30] has proposed an algorithm called the orthogonal direction algorithm and has shown that assuming exact arithmetic, it gives the same sequence of iterates as SYMMLQ. In addition to a matrix-vector product, it requires 7N + 3 multiplications per iteration, and only five vectors of storage. Hence, it has the favorable operation count of the SYMMBK method and the favorable storage requirements of the SYMMLQ method.

9.6: Luenberger's Method of Hyperbolic Pairs

In this section we present the method of hyperbolic pairs proposed by Luenberger [46]. In finite precision arithmetic, the method is unstable. Assuming exact arithmetic, we prove that the iterates obtained are exactly the same as those for SYMMBK (with $\gamma = 0$), and hence also the same as the SYMMLQ iterates.

We will refer to a nonzero vector x as singular if $(x, Ax) = 0$. Also, two vectors x and y will be said to form a hyperbolic pair of x and y are both singular and $(x, Ay) \neq 0$.

The CG method applied to (9.1) breaks down if, at any stage, the direction vector p_i is singular. Luenberger's method forces a singular direction, if generated at all, to be the first vector of a hyperbolic

pair.

Algorithm 9.3: Luenberger's Method of Hyperbolic Pairs

Step 1: Choose x_0.

Compute $r_0 = f - Ax_0$.

Set $p_0 = r_0$

and $i = 0$.

Step 2: If $(p_i, Ap_i) \neq 0$ go to Step 3a;

else go to Step 3b.

Step 3a: (p_i nonsingular) Compute

$$a_i = (r_i, r_i)/(Ap_i, p_i),$$

$$x_{i+1} = x_i + a_i p_i,$$

$$r_{i+1} = r_i - a_i Ap_i,$$

$$b_i = (-r_{i+1}, Ap_i)/(Ap_i, p_i),$$

and $$p_{i+1} = r_{i+1} + b_i p_i.$$

If x_{i+1} is sufficiently close to x, terminate the iteration process;

else set $i = i+1$ and go to Step 2.

Step 3b: (p_i singular) Compute

$$p_{i+1} = Ap_i - \frac{(Ap_i, A^2 p_i)}{2(Ap_i, Ap_i)} p_i,$$

$$a_i = (r_i, p_{i+1})/(Ap_i, p_i),$$

$$a_{i+1} = (r_i, p_i)/(Ap_i, p_{i+1}),$$

$$x_{i+2} = x_i + a_i p_i + a_{i+1} p_{i+1},$$

$$r_{i+2} = r_i - a_i Ap_i - a_{i+1} Ap_{i+1},$$
$$\text{and } p_{i+2} = r_{i+2} - \frac{(r_{i+2}, Ap_{i+1})}{(Ap_i, p_{i+1})} p_i.$$

If x_{i+1} is sufficiently close to x, terminate the iteration process;

else set i = i+2 and go to Step 2.

Note that if p_i is singular, p_i and p_{i+1} form a hyperbolic pair, and that x_{i+1} and r_{i+1} do not exist. (For singular p_i, Luenberger defines $x_{i+1} = x_i + a_i p_i$ where a_i is as given under Step 3b, but we prefer to say that x_{i+1} does not exist as this simplifies the subsequent terminology.)

Theorem 9.3: ([46]) The following relations hold.

(a) $(p_i, Ap_j) = 0$, $i \neq j$ unless p_i, p_j is a hyperbolic pair.

(b) $(p_i, Ap_{i+1}) = (Ap_i, Ap_i) \neq 0$ if p_i, p_{i+1} is a hyperbolic pair.

(c) $(r_i, p_j) = 0 \qquad i > j$

(d) $(r_i, p_j) = (r_0, p_j) \qquad i \leq j$

(e) For every i, $\{p_0, p_1, \ldots, p_i\} = \{r_0, Ar_0, \ldots, A^i r_0\}$.

(f) If singular vectors appear in the sequence $p_0, p_1, \ldots$, they appear as hyperbolic pairs.

(g) Algorithm 9.2 converges to the unique solution of (9.1) in at most N iterations.

(h) If x_i exists, then x_{i+1} exists iff p_i is nonsingular. If x_{i+1} does not exist, then x_{i+2} does.

When p_i is nonsingular, an iteration of Algorithm 9.3 requires 6N + 2 multiplications plus the matrix vector product Ap_i. When p_i is singular, Step 3b computes x_{i+2} from x_i so that we have a double iteration. This double iteration requires 12N + 5 multiplications and two matrix-vector products, Ap_i and $A(Ap_i)$, essentially twice the work of a single iteration. By Theorem 9.3(b), we have that in a double iteration, only one of (Ap_i, Ap_i) and (Ap_i, p_{i+1}) has to be computed. In addition to the symmetric matrix A, storage is required for six vectors - viz., x_i, r_i, p_i, Ap_i, p_{i+1} and Ap_{i+1}. The vector $A^2 p_i$ can share storage with Ap_{i+1}.

In any computation with this method, we have to decide how small (p_i, Ap_i) must be in order to decide that p_i is singular. For small problems (ten variables or less) in single precision, Luenberger treated p_i as singular if

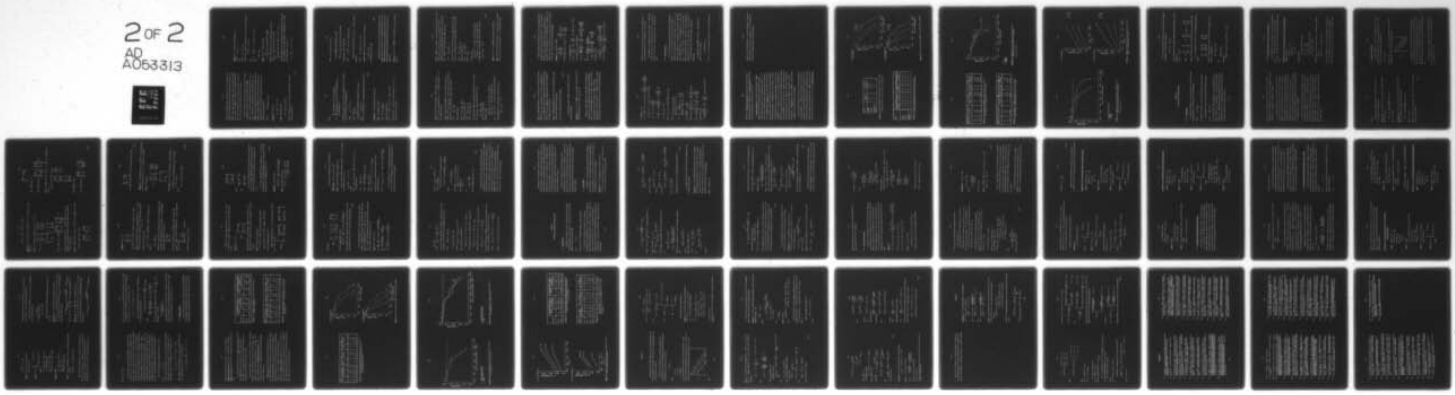
$$\frac{(p_i, Ap_i)}{(p_i, p_i)} < 10^{-4}$$

and found that the procedure worked. Larger problems required double precision.

However, there seems to be no theoretical justification for why the threshold of 10^{-4} (or any other for that matter) should work for a general problem. The reason is that Step 3b relies heavily on the fact that $(p_i, Ap_i) = 0$. In general if $(p_i, Ap_i) \neq 0$ and Step 3b is chosen, then p_{i+1} is not singular and not orthogonal to $\{Ap_j\}_{j=1}^{i-1}$. The properties in Theorem 9.3 will not hold for subsequent iterations and it

AD-A053 313 YALE UNIV NEW HAVEN CONN DEPT OF COMPUTER SCIENCE F/G 12/1
CONJUGATE GRADIENT METHODS FOR PARTIAL DIFFERENTIAL EQUATIONS.(U)
JAN 78 R CHANDRA N00014-76-C-0277
UNCLASSIFIED RR-129 NL

2 OF 2
AD
A053313



END
DATE
FILMED
6-78
DDC

seems entirely possible that the iterates will not converge to the true solution. The algorithm may even have serious stability problems; for example, (p_{i+1}, Ap_i) is no longer equal to (Ap_i, Ap_i) and may indeed be zero. Fletcher [30] suggested a modification to Luenberger's method to deal with the case when (p_i, Ap_i) is small but not zero. However, he did not follow it up due to potential stability problems.

We will now show that with exact arithmetic the iterates obtained by Luenberger's method are exactly those obtained by SYMMBK (with $\gamma = 0$). We prove some preliminary results which lead up to the result in Theorem 9.5. Superscripts L (for Luenberger) and B (for SYMMBK) are used to differentiate between vectors occurring in the two methods. We assume throughout that we start the algorithms with the same initial guess $x_0 = x_0^L = x_0^B$. We also define, for every k, the subspace $\bar{P}_k$ of dimension k by

$$\bar{P}_k \equiv \{p_0, p_1, \ldots, p_{k-1}\}. \tag{9.36}$$

Lemma 9.4: For every k,

$$\bar{P}_k = \bar{S}_k = \bar{V}_k = \bar{C}_k.$$

Proof: By Theorem 9.3(e)

$$\bar{P}_k = \{r_0, Ar_0, \ldots, A^{k-1}r_0\}.$$

Since $v_1 = r_0/\beta_1$ where β_1 is a scalar, we have $\bar{P}_k = \bar{S}_k$. The result follows from Lemma 9.2.

Q.E.D.

Lemma 9.5: Let k be such that x_k^L exists. Let

$$w_1 = \delta_1 z - u_1 \tag{9.37}$$

and

$$w_2 = \delta_2 z - u_2, \tag{9.38}$$

where the u_i, w_i and z are N-vectors, and $\delta_i \neq 0$ are scalars such that

$$u_i \in \bar{P}_k, \quad i = 1 \text{ and } 2,$$

and

$$(Aw_i, s) = 0, \quad \forall\, s \in \bar{P}_k, \; i = 1 \text{ and } 2. \tag{9.39}$$

Then

$$\delta_2 w_1 = \delta_1 w_2.$$

Proof: Since $u_1 \in \bar{P}_k$, we can write

$$u_1 = \sum_{i=0}^{k-1} \alpha_{1i} p_i \text{ for some scalars } \{\alpha_{1i}\}_{i=0}^{k-1}.$$

Taking the inner-product of (9.37) with each of the vectors $Ap_0, Ap_1, \ldots, Ap_{k-1}$ and using (9.39), we get

$$\sum_{i=0}^{k-1} \alpha_{1i}(p_i, Ap_j) = \delta_1(z, Ap_j), \quad j = 0, 1, \ldots, k-1.$$

Since x_k^L exists, we have by the definition of Algorithm 9.3 that p_{k-1} is either nonsingular or the second member of a hyperbolic pair. Hence, from Theorem 9.3(a), we get that

$$\alpha_{1i} = \delta_1 \beta_i, \quad i = 0, 1, \ldots, k-1,$$

where

$$\beta_i \equiv \begin{cases} (z, Ap_i)/(Ap_i, p_i) \text{ if } p_i \text{ is nonsingular,} \\ (z, Ap_i)/(Ap_i, p_{i-1}) \text{ if } p_i \text{ is the second member of a} \\ \text{hyperbolic pair.} \end{cases}$$

Since $(Ap_j, p_{j+1}) \neq 0$ if p_j, p_{j+1} form a hyperbolic pair (see Theorem 9.3(b)) the scalars β_i are well defined and

(9.40) $\quad w_1 = \delta_1 (z - u)$

where $u = \sum_{i=0}^{k-1} \beta_i p_i$.

Proceeding in the same way for equation (9.38), we get

(9.41) $\quad w_2 = \delta_2 (z - u)$

where u is as defined above. The Lemma follows from (9.40) and (9.41).

Q.E.D.

Theorem 9.4: If x_i^L and x_i^B exist, then $x_i^L = x_i^B$.

Proof: If x_i^L exists, we have

(9.42) $\quad x_i^L = x_0 + \sum_{j=0}^{i-1} a_j p_j$

where a_j are scalars, i.e.,

(9.43) $\quad (x - x_i^L) = (x - x_0) - u_1 \quad$ where $u_1 \in \bar{P}_i$.

Also, since $r_i^L = f - Ax_i^L = A(x - x_i^L)$ and $(r_i^L, p_j) = 0$, $j < i$, by Theorem 9.3(c), we have that

(9.44) $\quad ((x - x_i^L), As) = 0 \ \forall s \in \bar{P}_i$.

If the SYMMBK iterate x_i^B exists, we have

$x_i^B = x_0 + \sum_{j=1}^{i} \eta_{i,j} c_j$ where the $\eta_{i,j}$ are scalars.

Using Lemma 9.4, we can again write

(9.45) $\quad (x - x_i^B) = (x - x_0) - u_2$ where $u_2 \in \bar{P}_i$

From (9.8), we have that $(r_i^B, v_j) = 0$, $j \leq i$. Since $\bar{V}_i = \bar{P}_i$, we have that

(9.46) $\quad ((x - x_i^B), As) = 0, \quad \forall s \in \bar{P}_i$.

Applying Lemma 9.5 to equations (9.43) - (9.46) we get

$$x - x_i^L = x - x_i^B,$$

which implies that $x_i^L = x_i^B$.

Q.E.D.

Theorem 9.4 says that if the i^{th} iterates are generated in both methods they will be the same. The next Theorem shows that if $\gamma = 0$ in SYMMBK then x_i^B is generated iff x_i^L is generated. But first we need a lemma showing the relation between the direction vectors in the two algorithms.

Lemma 9.6: Let $\gamma = 0$ in SYMMBK. If $x_i^L = x_i^B$ then $p_i = \delta_i c_{i+1}$ for some $\delta_i \neq 0$.

Proof: We prove this by induction. Since $x_0^L = x_0^B$, $p_0 = r_0$, and $c_1 = v_1 = r_0/\beta_1$, the Lemma holds for i = 0. Suppose it holds for i = k. We consider two cases.

Case 1: Suppose p_k is nonsingular. Then by Theorem 9.3(h), x_{k+1}^L exists. Since $p_k = \delta_k c_{k+1}$, c_{k+1} is also nonsingular and x_{k+1}^B exists by Lemma 9.3(c). Then by Theorem 9.4, $x_{k+1}^L = x_{k+1}^B$, and hence

(9.47) $\quad r_{k+1}^L = r_{k+1}^B.$

Now, by definition,

(9.48) $\quad p_{k+1} = r_{k+1}^L - b_k p_k$

where b_k is a scalar, and

(9.49) $\quad c_{k+2} = v_{k+2} - u$

where $u \in \bar{C}_{k+1}$ by (9.24).

We intend to use Lemma 9.5 on equations (9.48) and (9.49) to obtain $p_{k+1} = \delta_{k+1} c_{k+2}$. Using (9.30) and (9.47) we have

$$v_{k+2} = \alpha r_{k+1}^B = \alpha r_{k+1}^L$$

where $\alpha \neq 0$ is some scalar.

Moreover, we have

$b_k p_k \in \bar{P}_{k+1}$, $u \in \bar{C}_{k+1} = \bar{P}_{k+1}$ by Lemma 9.4,

$(p_{k+1}, As) = 0$, $\forall s \in \bar{P}_{k+1}$ by Theorem 9.3(a),

and $(c_{k+2}, As) = 0$, $\forall s \in \bar{P}_{k+1}$ by Lemma 9.3(b) and Lemma 9.4.

Applying Lemma 9.5 to equations (9.48) and (9.49) we get

$$p_{k+1} = \alpha\, c_{k+2}, \quad \alpha \neq 0.$$

Case 2: Suppose p_k is singular. By Theorem 9.3(h), x_{k+1}^L does not exist but x_{k+2}^L does. Since $p_k = \delta_k c_{k+1}$, c_{k+1} is singular. Thus by Lemma 9.3(c) and the SYMMBK algorithm, x_{k+1}^B does not exist but x_{k+2}^B does. Theorem 9.4 implies that $x_{k+2}^L = x_{k+2}^B$, and hence

(9.50) $\quad r_{k+2}^L = r_{k+2}^B.$

Again, as in Case 1, we have that $v_{k+3} = \alpha r_{k+2}^B$ where $\alpha \neq 0$ is some scalar. By the definitions of the Algorithms 9.2 and 9.3, we have

(9.51) $\quad p_{k+2} = r_{k+2}^L - \beta p_k$

where β is a scalar, and

(9.52) $\quad c_{k+3} = \alpha r_{k+2}^B - u$

where $u \in \bar{C}_{k+2}$.

Since $p_k \in \bar{P}_{k+2}$,

$(p_{k+2}, As) = 0$, $\forall s \in \bar{P}_{k+2}$, by Theorem 9.3(a), and

$(c_{k+3}, As) = 0$, $\forall s \in \bar{C}_{k+2} = \bar{P}_{k+2}$ by Lemma 9.3(b).

We apply Lemma 9.5 to equations (9.51) and (9.52) to obtain

$$p_{k+2} = \alpha\, c_{k+2}, \quad \alpha \neq 0.$$

Q.E.D.

Lemma 9.7: If $\gamma = 0$ in SYMMBK, then x_i^L exists iff x_i^B does.

Proof: We prove the result by induction. Since $x_0^L = x_0^B$, it is clearly true for i = 0. Let it be true for i = k. We consider two cases.

Case 1: Suppose x_k^L and x_k^B exist. By Theorem 9.4, they are equal and by Lemma 9.6, $p_k = \delta_k c_{k+1}$ for some $\delta_k \neq 0$. Thus p_k is nonsingular iff c_{k+1} is nonsingular. Moreover, by Lemma 9.3(c) and Theorem 9.3(h), x_{k+1}^B exists iff c_{k+1} is nonsingular, and x_{k+1}^L exists iff p_k is nonsingular. Hence x_{k+1}^B exists iff x_{k+1}^L does.

Case 2: Suppose x_k^L and x_k^B do not exist. Then, by Theorem 9.3(h), x_{k+1}^L exists and by Algorithm 9.2, x_{k+1}^B exists.

Cases 1 and 2 complete the proof of the Lemma.

Q.E.D.

Combining the results of Lemma 9.7 and Theorem 9.4, we have proved the following.

Theorem 9.5: Let $\gamma = 0$ in SYMMBK. In the absence of roundoff error, x_i^L exists iff x_i^B exists and if they exist, $x_i^L = x_i^B$.

Theorem 9.5 clearly shows that, in the absence of roundoff error, the sequence of iterates generated by Luenberger's method and SYMMBK (with $\gamma = 0$) and hence SYMMLQ are all the same. However, with floating point arithmetic we can get stability in SYMMBK by choosing γ appropriately, as given in Theorem 9.2. Luenberger's method is unstable in the presence of roundoff and no similar "fix" has been demonstrated.

Lemma 9.6 implies that whenever $x_i^L = x_i^B$, the two methods generate the same new direction (up to a scalar factor). Since at least every second x_i exists, at least half the vectors in the sequence of p's generated are scalar multiples of the corresponding c's. However, if x_i^L and x_i^B do not exist, the vectors p_i and c_{i+1} may not be scalar multiples of each other. This is shown by the following example.

Example 9.1: Solve $Ax = f$ where

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & -3 \end{bmatrix}, \qquad f = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}, \text{ and the}$$

initial guess $x_0 = \underline{0}$. Then $r_0 = f - Ax_0 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}$.

I. Luenberger's method:

$p_0 = r_0 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}$, $Ap_0 = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$, and $(Ap_0, p_0) = 0$. Thus p_0 is singular and x_1 does not exist.

$$p_1 = Ap_0 - \frac{(Ap_0, A^2p_0)}{2(Ap_0, Ap_0)} p_0 = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + \frac{18}{2 \times 14} \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = \frac{1}{14} \begin{bmatrix} 23 \\ 37 \\ 33 \end{bmatrix}$$

and

$$x_2^L = x_0^L + \frac{(r_0, p_1)}{(p_0, Ap_1)} p_0 + \frac{(r_0, p_0)}{(p_0, Ap_1)} p_1$$

$$= \underline{0} + \frac{1}{14} \quad \frac{27}{14} \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} + \frac{3}{14} \begin{bmatrix} 23 \\ 37 \\ 33 \end{bmatrix} = \frac{3}{98} \begin{bmatrix} 16 \\ 23 \\ 12 \end{bmatrix}.$$

Continuing, we get

$$r_2^L = \frac{5}{49} \begin{bmatrix} 5 \\ -4 \\ 1 \end{bmatrix}, \quad p_2 = r_2^L - \frac{(r_2^L, Ap_1)}{(p_0, Ap_1)} p_0 = \frac{15}{343} \begin{bmatrix} 15 \\ -6 \\ -1 \end{bmatrix}$$

and $x_3^L = x_2^L + \frac{(r_2^L, p_2)}{(p_2, Ap_2)} p_2 = \begin{bmatrix} 1 \\ 1/2 \\ 1/3 \end{bmatrix}$, which is the solution.

II. SYMMBK ($\gamma = 0$):

$\beta_1 = \|r_0\|_2 = \sqrt{3}$, $v_1 = r_0/\|r_0\|_2 = \frac{1}{\sqrt{3}}\begin{bmatrix}1\\1\\-1\end{bmatrix}$, $Av_1 = \frac{1}{\sqrt{3}}\begin{bmatrix}1\\2\\3\end{bmatrix}$, and

$\alpha_1 = (Av_1, v_1) = 0$. So a 2 x 2 pivot is chosen.

$Av_1 - \alpha_1 v_1 = Av_1 = \frac{1}{\sqrt{3}}\begin{bmatrix}1\\2\\3\end{bmatrix}$,

$\beta_2 = \frac{14}{\sqrt{3}}$, $v_2 = \frac{1}{\sqrt{14}}\begin{bmatrix}1\\2\\3\end{bmatrix}$, $Av_2 = \frac{1}{\sqrt{14}}\begin{bmatrix}1\\4\\-9\end{bmatrix}$, and

$\alpha_2 = (Av_2, v_2) = -9/7$.

Now, $c_1 = v_1$, $c_2 = v_2$ and $x_2^B = x_0^B + \eta_1 c_1 + \eta_2 c_2$,

where

$$\begin{bmatrix}\alpha_1 & \beta_2\\ \beta_2 & \alpha_2\end{bmatrix}\begin{bmatrix}\eta_1\\ \eta_2\end{bmatrix} = \begin{bmatrix}\|r_0\|_2\\ 0\end{bmatrix}$$

i.e.,

$$\begin{bmatrix}0 & \sqrt{(14/3)}\\ \sqrt{(14/3)} & -9/7\end{bmatrix}\begin{bmatrix}\eta_1\\ \eta_2\end{bmatrix} = \begin{bmatrix}\sqrt{3}\\ 0\end{bmatrix}$$

or $\eta_1 = \frac{27\sqrt{3}}{98}$, $\eta_2 = \frac{3}{\sqrt{14}}$.

Thus with $c_1 = \frac{1}{\sqrt{3}}\begin{bmatrix}1\\1\\-1\end{bmatrix}$ and $c_2 = \frac{1}{\sqrt{14}}\begin{bmatrix}1\\2\\3\end{bmatrix}$, we have

$$x_2^B = \underline{0} + \frac{27}{98}\begin{bmatrix}1\\1\\-1\end{bmatrix} + \frac{3}{14}\begin{bmatrix}1\\2\\3\end{bmatrix} = \frac{3}{98}\begin{bmatrix}16\\23\\12\end{bmatrix}.$$

Continuing, we get

$c_3 = \frac{3}{7\sqrt{42}}\begin{bmatrix}-15\\6\\1\end{bmatrix}$ and $x_3^B = \begin{bmatrix}1\\1/2\\1/3\end{bmatrix}$, which is the solution.

Clearly p_1 is not a scalar multiple of c_2 so that the sequence of directions generated by the two algorithms is not the same.

The example also illustrates our previous result. It shows that for $x_0^L = x_0^B$, we get $x_2^L = x_2^B$ and $x_3^L = x_3^B$. Moreover, p_0 is a scalar multiple of c_1 and p_2 is a scalar multiple of c_3. x_1^L and x_1^B do not exist.

9.7: Numerical Results

The two- and three-dimensional model problems were introduced in Chapter 2. In this section we present the results of numerical experiments on these problems for various values of σ and h. If $\sigma = 0$, then all the eigenvalues of A are positive. The first few eigenvalues of the differential operator $-\Delta$ in two dimensions are 19.7, 49.4, 79.0, 98.7,..., (see [34]), so that for $\sigma = 30$ and 90 A has one and three negative eigenvalues respectively. Similarly, in three dimensions, the eigenvalues of $-\Delta$ are 29.6, 59.2, 88.8, 108.6,..., [34] so that for $\sigma = 50$ and 100 A again has one and three negative eigenvalues respectively.

In experiments with SYMMBK, we computed γ (as indicated in Theorem 9.2) by the formula

$$\gamma = 4d \sin^2(\pi n/(n+1)),$$

where d is 2 for two-dimensional problems and 3 for three-dimensional

problems. In Figures 9.1a and 9.1b we plot the error versus number of iterations for the model problems and in Tables 9.3 and 9.4 we give the total work requirements to reduce the error to 10^{-6}.

In Figures 9.2 and 9.3, we compare the performance of SYMMBK, SYMMLQ, and CG for the two-dimensional indefinite model problem. Similar results were obtained for three-dimensional problems. It is clear that the CG method can become unstable and may not converge at all. The SYMMBK curve in Figure 9.2 has a few peaks, but the process remains stable as predicted by the stability of the Bunch and Kaufman factorization procedure. The SYMMLQ curve is much smoother. Since the iterates x_k^L (and not x_k) are computed during SYMMLQ, the plotted curve corresponds to $\|x - x_k^L\|_2$. That x_k^L is not as good an approximation in general as x_k is evident from the fact that the SYMMLQ curve lags behind the one for SYMMBK. At the last step, x_k is computed from x_k^L, hence the nearly vertical drop at the last iteration. Since the two methods are equivalent, they require the same number of iterations for the same final accuracy. In Figure 9.3, we show that SYMMBK is clearly superior to SYMMLQ in terms of the total work requirements.

We do not have any theoretical results on the rate of convergence of the above methods. In order to give an indication of their rate of convergence for the model problems, we defined the solution as in (2.7) and (since the difference approximations are only second-order accurate) computed the number of iterations required by SYMMBK to reduce the error by a factor of $1/n^2$ for various σ and h. We see from Figures 9.4a and

9.4b that for a fixed problem (i.e. fixed σ) the graph of the number of iterations against n log n is a straight line. This illustrates the O(n log n) convergence result for the model problem.

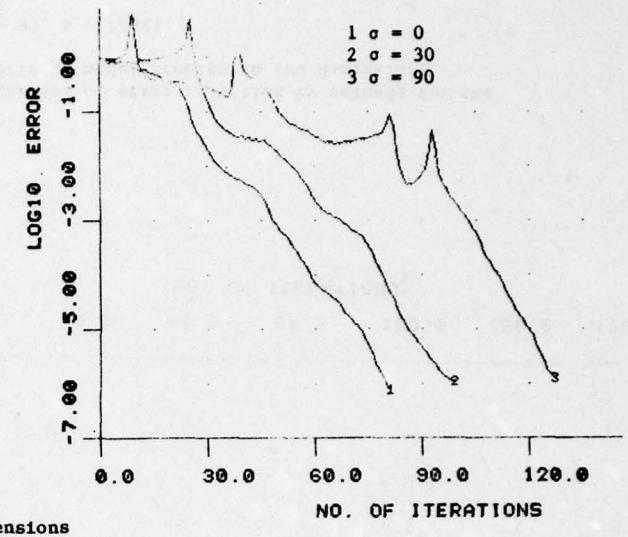
Method	Storage required	No. of mults./ iteration
SYMMBK	6N	8N + 8
SYMMLQ	5N	10N + 16
Luenberger's Method	6N	7N + 2

Table 9.1: Storage requirements and multiplication counts for model problem in two ($N = n^2$) and three ($N = n^3$) dimensions

Method	Two dimensions ($N = n^2$)		Three dimensions ($N = n^3$)	
	Storage required	No. of mults./ iteration	Storage required	No. of mults./ iteration
SYMMBK	9N - 2n	12N - 4n + 8	$10N - 3n^2$	$14N - 6n^2 + 8$
SYMMLQ	8N - 2n	14N - 4n + 16	$9N - 3n^2$	$16N - 6n^2 + 16$
Luenberger's Method	9N - 2n	11N - 4n + 2	$10N - 3n^2$	$13N - 6n^2 + 2$

Table 9.2: Storage requirements and multiplication counts for generalized model problem in two and three dimensions

(a) Two dimensions
(h = 1/32)



(b) Three dimensions
(h = 1/16)

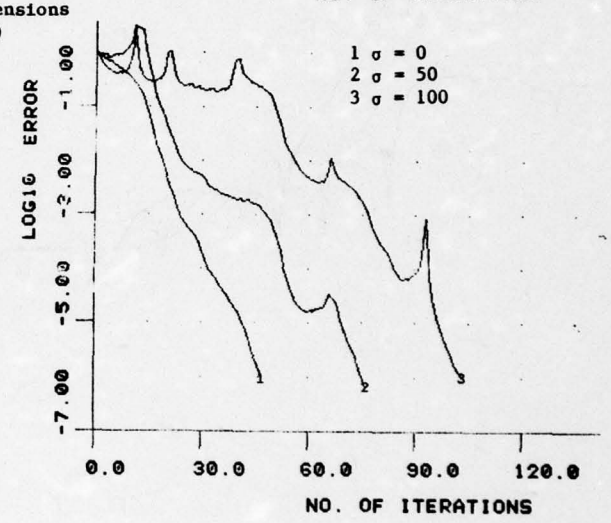


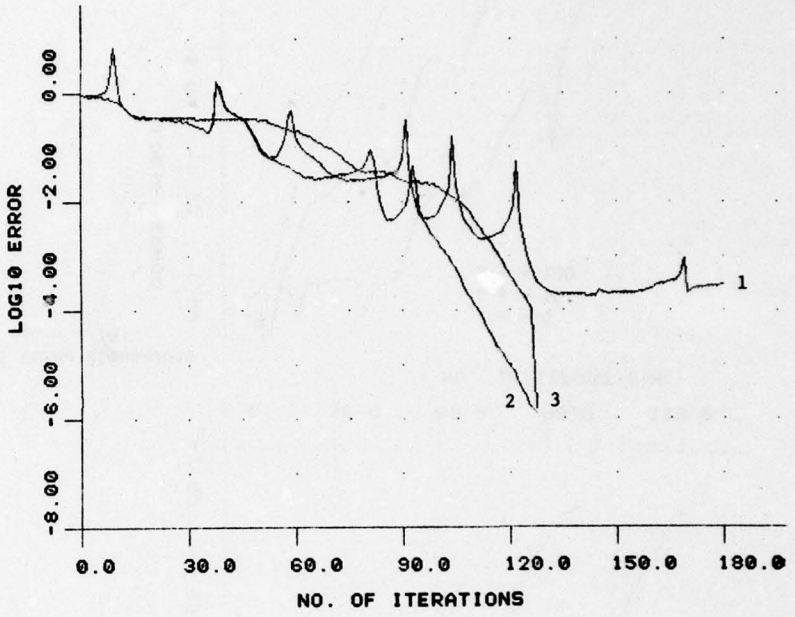
Figure 9.1: Error reduction by SYMMBK for indefinite model problem

Mesh-width h=1/(n+1)	No. of mults./ iteration	σ = 30		σ = 90	
		No. of iters.	Total no. of mults.	No. of iters.	Total no. of mults.
1/8	400	20	8,000	20	10,800
1/16	1,808	41	74,128	60	108,480
1/32	7,696	99	761,904	129	992,784

Table 9.3: Work required by SYMMBK to reduce the error to 10^{-6} for model problem in two dimensions on an n x n mesh

Mesh-width h=1/(n+1)	No. of mults./ iteration	σ = 50		σ = 100	
		No. of iters.	Total no. of mults.	No. of iters.	Total no. of mults.
1/4	224	9	2,016	9	2,016
1/8	2,752	37	101,824	66	181,632
1/16	27,008	76	2,052,608	109	2,943,872

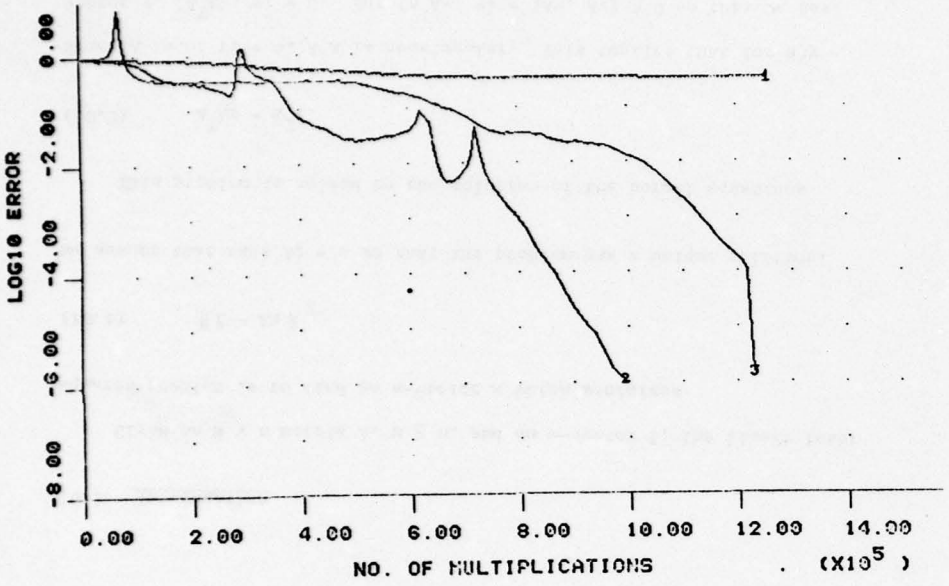
Table 9.4: Work required by SYMMBK to reduce the error to 10^{-6} for model problem in three dimensions on an n x n x n mesh.



1 CG
2 SYMMBK
3 SYMMLQ

Figure 9.2: Comparison of error reduction by methods for the indefinite model problem in two dimensions

(σ = 90, h = 1/32)

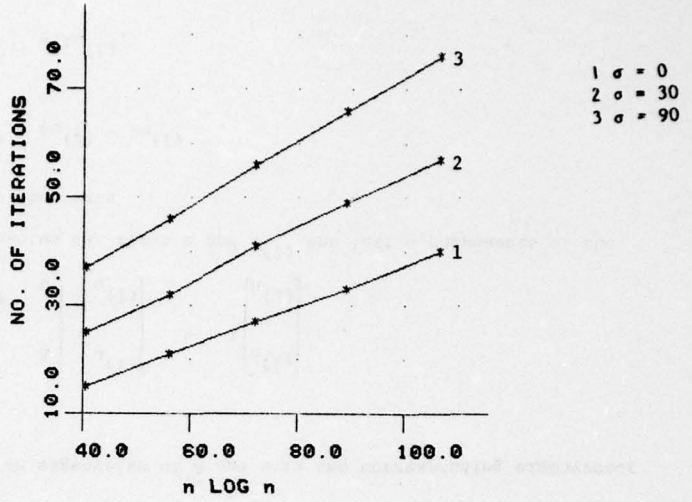


1 CG for Normal Equations
2 SYMMBK
3 SYMMLQ

Figure 9.3: Comparison of work requirements by methods for the indefinite model problem in three dimensions

(σ = 100, h = 1/16)

(a) Two dimensions



(b) Three dimensions

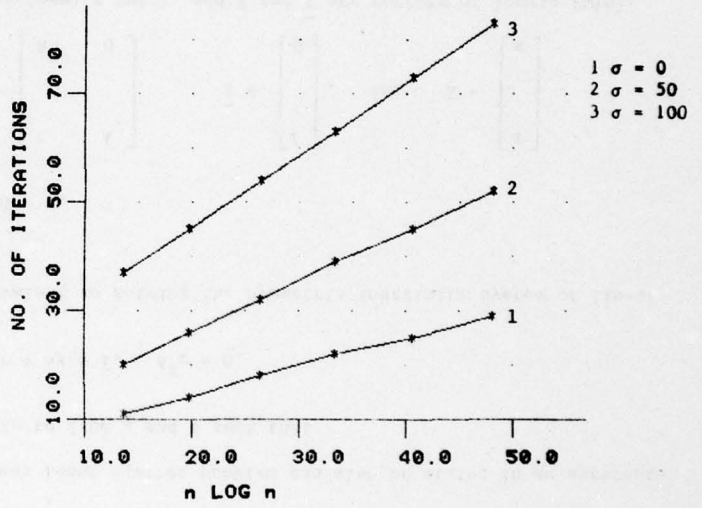


Figure 9.4: Graphs illustrating O(n log n) convergence result for SYMMBK on indefinite model problem

CHAPTER 10

THE LINEAR LEAST SQUARES PROBLEM

10.1: Introduction

Given an m x n matrix A, $m \geq n$, and an m-vector f, the linear least squares problem is to find an n-vector x which minimizes

(10.1) $\|f - Ax\|_2$.

We assume that rank(A) = n so that the problem has a unique solution.

This problem is solved by the solution of the normal equations

(10.2) $A^TAx = A^Tf$.

Since A is of rank n, A^TA is nonsingular. This implies that for any vector v, $(A^TAv, v) \neq 0$. But $(A^TAv, v) = (Av, Av) \geq 0$ so that we have $(A^TAv, v) > 0$. Therefore A^TA is positive definite, and we can use the CG method to solve (10.2). A computational version of the CG method on the normal equations [39] is given in Section 2.

The linear least squares problem can also be stated in an alternate form, which is to find x and r such that

$$r + Ax = f, \quad A^Tr = \underline{0}.$$

This is equivalent to solving the symmetric indefinite system of linear equations

(10.3) $\tilde{A}\tilde{x} = \tilde{f}$,

where $\tilde{A} = \begin{bmatrix} I & A \\ A^T & 0 \end{bmatrix}$, $\tilde{f} = \begin{bmatrix} f \\ 0 \end{bmatrix}$, and $\tilde{x} = \begin{bmatrix} r \\ x \end{bmatrix}$.

Clearly, $\tilde{A}$ is (m+n) x (m+n), and $\tilde{x}$ and $\tilde{f}$ are vectors of length (m+n).

If λ is an eigenvalue of $\tilde{A}$ and u is the corresponding eigenvector we have

$$\begin{bmatrix} I & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} u^{(1)} \\ u^{(2)} \end{bmatrix} = \lambda \begin{bmatrix} u^{(1)} \\ u^{(2)} \end{bmatrix},$$

where $u^{(1)}$ denotes the first m and $u^{(2)}$ the last n components of the vector u. We then have

$$u^{(1)} + Au^{(2)} = \lambda u^{(1)}$$

and

$$A^Tu^{(1)} = \lambda u^{(2)},$$

which give

$$A^TAu^{(2)} = \lambda(\lambda-1)u^{(2)} \quad \text{and} \quad AA^Tu^{(1)} = \lambda(\lambda-1)u^{(1)}.$$

Thus $\lambda(\lambda-1)$ is an eigenvalue of A^TA and also of AA^T. Since A^TA is nonsingular this implies that $\tilde{A}$ does not have a zero eigenvalue. Hence system (10.3) is nonsingular and symmetric indefinite.

It has been suggested, eg., [46] and [51], that the methods of Chapter 9 can be used to efficiently solve the linear least squares problem. In [51] Paige and Saunders point out that the tridiagonal matrix $\tilde{T}_k = V_k^T \tilde{A} V_k$ of (9.12), that would be obtained for this problem, has alternating 0's and 1's on the diagonal and that the matrix V_k has columns which alternatively are zero in the first m and last n components. Due to this special structure, the work per iteration and storage requirements for SYMMLQ are only about half of those be required for a general (m+n) x (m+n) problem.

We show in Section 3, that assuming exact arithmetic the sequence of approximations obtained by CG on the normal equations are exactly those obtained by SYMMBK (for any $\gamma > 0$). Moreover, the work and storage required to yield the same approximation is less for CG on the normal equations. Earlier, we proved the mathematical equivalence of SYMMBK and SYMMLQ. Therefore, in terms of efficiency, the methods of Chapter 9 seem not to offer any advantage over the CG method applied to the normal equations.

10.2: The Conjugate Gradient Method for Normal Equations

We use Algorithm 4.2, the CG method, to solve the normal equations (10.2). It turns out that A^TA and A^Tf do not have to be formed explicitly. This algorithm was first presented by Hestenes and Stiefel [39].

Algorithm 10.1: The Conjugate Gradient Method for Normal Equations

Step 1: Choose x_0.

Compute $r_0 = f - Ax_0$.

Set $p_0 = A^Tr_0$

and $i = 0$.

Step 2: Compute

$$a_i = (A^Tr_i, A^Tr_i)/(Ap_i, Ap_i),$$

$$x_{i+1} = x_i + a_ip_i,$$

$$r_{i+1} = r_i - a_iAp_i,$$

$$b_i = (A^Tr_{i+1}, A^Tr_{i+1})/(A^Tr_i, A^Tr_i),$$

and $$p_{i+1} = A^Tr_{i+1} + b_ip_i.$$

Step 3: If x_{i+1} is sufficiently close to x, terminate the iteration process;

else set $i = i + 1$ and go to Step 2.

The cost per iteration is two matrix-vector products, A^Tr and Ap, plus $3m + 2n + 2$ multiplications. Apart from the matrix A, storage is required for the four vectors x, r, p and Ap . A^Tr can share storage

with Ap.

Applying the convergence result of Theorem 4.1 to (10.2) with $\mu = 1$, we get the following bound.

Theorem 10.1: The iterates for the CG method for solving the normal equations satisfy

$$\| x_i - x \|_2 \leq \frac{2}{\sqrt{\alpha}} \left(\frac{1 - \sqrt{\alpha}}{1 + \sqrt{\alpha}}\right)^i \| x_0 - x \|_2 ,$$

where $\alpha \equiv 1/\kappa(A^T A)$.

Corollary 10.1: If the matrix A is square, the iterates for the CG method for solving the normal equations, satisfy

$$\| x_i - x \|_2 \leq \frac{2}{\alpha} \left(\frac{1 - \alpha}{1 + \alpha}\right)^i \| x_0 - x \|_2 ,$$

where $\alpha = 1/\kappa(A)$.

Proof: Noting that if A is square, $\kappa(A^T A) = (\kappa(A))^2$, the result follows directly from Theorem 10.1.

Q.E.D.

10.3: Comparison of Methods

We now apply SYMMBK ($\gamma > 0$) to the linear system (10.3). If x_0 is an initial approximation to x, then $\tilde{x}_0 = [r_0, x_0]^T$ where $r_0 = f - Ax_0$. We follow the notation of Chapter 9 and define

$$\tilde{T}_k \equiv V_k^T \tilde{A} V_k$$

where $V_k = [v_1, v_2, \ldots v_k]$, the v_i's are (m+n)-vectors that form an orthonormal set, and $\tilde{T}_k$ is the tridiagonal matrix

$$\tilde{T}_k = \begin{bmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \ddots & & \\ & \ddots & \ddots & \beta_k & \\ & & \beta_k & \alpha_k & \end{bmatrix} .$$

In the sequence of lemmas that follow we prove that the special structure of $\tilde{T}_k$ allows us to show that only 2 x 2 pivots are chosen and that the sequence of approximations obtained is $x_0, x_2, x_4, \ldots$; i.e., x_i exists iff i is even. In Theorem 10.2, we prove that $x_{2i} = x_i^{CG}$, where x_i^{CG} is the i^{th} iterate obtained in solving the normal equations by CG.

For any (m+n)-vector z, we denote by $z^{(1)}$ the vector consisting of the first m components of z, and by $z^{(2)}$ the vector consisting of the last n components of z. Then

$z \equiv [z^{(1)}, z^{(2)}]^T$. Also $\tilde{r}_i \equiv \tilde{f} - \tilde{A}\tilde{x}_i$.

Lemma 10.1: If i is even, $\alpha_i = 1$ and $v_i^{(2)} = \underline{0}$. If i is odd, $\alpha_i = 0$ and $v_i^{(1)} = \underline{0}$.

Proof: The proof is by induction. For i = 1 we have

$$\beta_1 v_1 = \tilde{r}_0 = \begin{bmatrix} f \\ 0 \end{bmatrix} - \begin{bmatrix} I & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} r_0 \\ x_0 \end{bmatrix},$$

$$= \begin{bmatrix} f - r_0 - Ax_0 \\ -A^T r_0 \end{bmatrix} = \begin{bmatrix} 0 \\ -A^T r_0 \end{bmatrix},$$

and $\alpha_1 = (\tilde{A}v_1, v_1) = (1/\beta_1^2)\left(\begin{bmatrix} -AA^T r_0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ -A^T r_0 \end{bmatrix} \right) = 0.$

so that the result holds for i = 1. Assume it holds for $i \leq k$. We consider the two cases, k odd and k even, and show that it holds for $i = k+1$ in both cases.

Case 1: Suppose k is odd. Then $\alpha_k = 0$, $v_k^{(1)} = \underline{0}$, and $v_{k-1}^{(2)} = \underline{0}$, so that

$$\beta_{k+1} v_{k+1} = \tilde{A}v_k - \alpha_k v_k - \beta_k v_{k-1}$$

$$= \begin{bmatrix} Av_k^{(2)} \\ 0 \end{bmatrix} - \beta_k \begin{bmatrix} v_{k-1}^{(1)} \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} Av_k^{(2)} - \beta_k v_{k-1}^{(1)} \\ 0 \end{bmatrix}$$

and so $v_{k+1}^{(2)} = \underline{0}$. Also

$$\alpha_{k+1} = (\tilde{A}v_{k+1}, v_{k+1}) = \left(\begin{bmatrix} v_{k+1}^{(1)} \\ A^T v_{k+1}^{(1)} \end{bmatrix}, \begin{bmatrix} v_{k+1}^{(1)} \\ 0 \end{bmatrix} \right) = 1.$$

Case 2: Suppose k is even. Then $\alpha_k = 1$, $v_k^{(2)} = \underline{0}$, and $v_{k-1}^{(1)} = \underline{0}$, so that

$$\beta_{k+1} v_{k+1} = \tilde{A}v_k - \alpha_k v_k - \beta_k v_{k-1}$$

$$= \begin{bmatrix} v_k^{(1)} \\ A^T v_k^{(1)} \end{bmatrix} - \begin{bmatrix} v_k^{(1)} \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 \\ A^T v_k^{(1)} \end{bmatrix}$$

and so $v_{k+1}^{(1)} = \underline{0}$. Also

$$\alpha_{k+1} = (\tilde{A}v_{k+1}, v_{k+1}) = \left(\begin{bmatrix} Av_{k+1}^{(2)} \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ v_{k+1}^{(2)} \end{bmatrix} \right) = 0.$$

Q.E.D.

Lemma 10.2: If $\gamma > 0$,

$$\begin{bmatrix} 0 & \beta_{2i} \\ \beta_{2i} & 1 \end{bmatrix}$$

is chosen as the i^{th} pivot. Moreover, $\tilde{x}_i$ exists iff i is even.

Proof: The proof is by induction. Since by Lemma 10.1, $0 = \alpha_1 < \gamma\beta_2^2$, the first pivot chosen is clearly $\begin{bmatrix} 0 & \beta_2 \\ \beta_2 & 1 \end{bmatrix}$. Assume that the Lemma holds for the first k pivots.

For $k = 2i$, Step 2 of Algorithm 9.2 gives us that α_{2i+1} is set equal to

$$\alpha_{2i+1} - m_2\beta_{2i+1},$$

where $m_2 = P_{11}\beta_{2i+1}/|P|$, $|P|$ denotes the value of the determinant of the last pivot, and P_{11} denotes the (1,1) element of P. By the induction hypothesis, we have that P is given by

$$\begin{bmatrix} 0 & \beta_{2k} \\ \beta_{2k} & 1 \end{bmatrix}$$

so that $m_2 = 0$ and α_{2i+1} is unchanged in value. By Lemma 10.1, $\alpha_{2i+1} = 0$. Hence, in Step 3 of Algorithm 9.2, $0 = \alpha_{2i+1} < \gamma\beta_{2i+1}^2$ and the 2 x 2 pivot

$$\begin{bmatrix} 0 & \beta_{2i+2} \\ \beta_{2i+2} & \alpha_{2i+2} \end{bmatrix}$$ is chosen.

By Lemma 10.1, $\alpha_{2i+2} = 1$, so that

$$\begin{bmatrix} 0 & \beta_{2i+2} \\ \beta_{2i+2} & 1 \end{bmatrix}$$

is chosen as the $(k+1)^{st}$ 2 x 2 pivot, and the result follows.

Q.E.D.

Lemma 10.3: If $\tilde{x}_i$ exists and $\tilde{x}_i = [q_i, x_i]^T$ where q_i is an m-vector and x_i is an n-vector, then $q_i = r_i \equiv f - Ax_i$ and $\tilde{r}_i = [0, -A^T r_i]^T$.

Proof: By definition

$$\tilde{r}_i = \begin{bmatrix} f \\ 0 \end{bmatrix} - \begin{bmatrix} I & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} q_i \\ x_i \end{bmatrix}$$

$$= \begin{bmatrix} f - q_i - Ax_i \\ -A^T q_i \end{bmatrix}.$$

Using Lemmas 10.1 and 10.2, we know that i is even and $v_{i+1}^{(1)} = \underline{0}$.

By (9.30), $\tilde{r}_i$ is a scalar multiple of v_{i+1}. This gives

$$f - q_i - Ax_i = \underline{0}.$$

Thus,

$$q_i = r_i \quad \text{and} \quad \tilde{r}_i = [0, -A^T r_i]^T.$$

Q.E.D.

Clearly x_i is defined only when $\tilde{x}_i$ exists, i.e. when i is even. To show that x_i is equal to $x^{CG}_{i/2}$ for even i, we need the following result.

Lemma 10.4: For $i \geq 2$,

$$\tilde{A}^i \tilde{r}_0 = \begin{bmatrix} A\, P_i^{(1)}\, A^T r_0 \\ \\ P_i^{(2)}\, A^T r_0 \end{bmatrix},$$

where $P_i^{(1)}$ and $P_i^{(2)}$ are polynomials of degree $(i-1)/2$ in A^TA if i is odd, and $P_i^{(1)}$ is of degree $(i/2)-1$ and $P_i^{(2)}$ is of degree $(i/2)$ in A^TA if i is even.

Proof: The proof is by induction.

Since $\tilde{r}_0 = [0, -A^T r_0]^T$, by direct multiplication we have

$$\tilde{A}\tilde{r}_0 = [-AA^T r_0, 0]^T \quad \text{and} \quad \tilde{A}^2\tilde{r}_0 = [-AA^T r_0, (-A^TA)\, A^T r_0]^T,$$

so that the Lemma holds for i = 2.

Suppose the result holds for $i \leq k$. We will prove that it holds for i = k+1. We have

$$\tilde{A}^{k+1}\tilde{r}_0 = \begin{bmatrix} I & A \\ \\ A^T & 0 \end{bmatrix} \begin{bmatrix} A\, P_k^{(1)} A^T r_0 \\ \\ P_k^{(2)} A^T r_0 \end{bmatrix} = \begin{bmatrix} A(P_k^{(1)} + P_k^{(2)}) A^T r_0 \\ \\ A^TA\, P_k^{(1)}\, A^T r_0 \end{bmatrix}$$

$$= \begin{bmatrix} A\, P_{k+1}^{(1)}\, A^T r_0 \\ \\ P_{k+1}^{(2)}\, A^T r_0 \end{bmatrix},$$

where

$$P_{k+1}^{(1)} = P_k^{(1)} + P_k^{(2)}$$

and

$$P_{k+1}^{(2)} = A^TA\, P_k^{(1)}.$$

If k is odd, then $P_k^{(1)}$ and $P_k^{(2)}$ are polynomials of degree $(k-1)/2$ in A^TA so that $P_{k+1}^{(1)}$ and $P_{k+1}^{(2)}$ are polynomials of degree $(k-1)/2$ and $(k+1)/2$ in A^TA respectively, which shows that the Lemma holds for i = k+1.

If k is even, then $P_k^{(1)}$, $P_k^{(2)}$ are of degree $(k/2) - 1$ and $(k/2)$ in A^TA respectively. Hence $P_{k+1}^{(1)}$ and $P_{k+1}^{(2)}$ are both of degree $(k/2)$ in A^TA, which shows that the Lemma holds for i = k+1.

Q.E.D.

Lemma 10.5: For every $i \geq 0$,

$$\{v_1^{(2)}, v_2^{(2)}, \ldots, v_{2i}^{(2)}\} \subseteq \{(A^TA)^j A^T r_0\}_{j=0}^{i-1}.$$

Proof: By Lemma 9.2 and (9.30),

$$\{v_1, v_2, \ldots, v_{2i}\} = \{v_1, \tilde{A}v_1, \ldots, \tilde{A}^{2i-1}v_1\}$$
$$= \{\tilde{r}_0, \tilde{A}\tilde{r}_0, \ldots, \tilde{A}^{2i-1}\tilde{r}_0\}.$$

Then since

$$\tilde{A}\tilde{r}_0 = \begin{bmatrix} I & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} 0 \\ -A^T r_0 \end{bmatrix} = \begin{bmatrix} -AA^T r_0 \\ 0 \end{bmatrix},$$

we get by using Lemma 10.4 that

$$\{v_1^{(2)}, v_2^{(2)}, \ldots, v_{2i}^{(2)}\} = \{A^T r_0, \ \{P_j^{(2)} A^T r_0\}_{j=2}^{2i-1}\},$$

where the $P_j^{(2)}$'s are polynomials of degree $(j-1)/2$ in A^TA if j is odd, and of degree $(j/2)$ in A^TA if j is even, and the Lemma follows.

Q.E.D.

We now apply Algorithm 10.1 to solve the linear least squares problem. We assume that the algorithm is started with the same initial guess as the SYMMBK algorithm, i.e., $x_0^{CG} = x_0$. We define $r_i^{CG} \equiv f - Ax_i^{CG}$. The following result comes directly from the properties of the CG method (see Sections 4.4 and 3.2).

Lemma 10.6: For all $i \geq 0$,

(a) $x_{i+1}^{CG} = x_0^{CG} + u$ where $u \in \{(A^TA)^j A^T r_0^{CG}\}_{j=0}^{i}$,

(b) $\{(A^TA)^j A^T r_0^{CG}\}_{j=0}^{i} = \{A^T r_j^{CG}\}_{j=0}^{i}$,

and

(c) $(A^T r_{i+1}^{CG}, w) = 0, \quad \forall w \in \{A^T r_j^{CG}\}_{j=0}^{i}$.

We now prove the equivalence of the two methods.

Theorem 10.2: For all $i \geq 0$, $x_{2i} = x_i^{CG}$.

Proof: We prove the Theorem by induction on i.

Since $x_0 = x_0^{CG}$ the Theorem clearly holds for $i = 0$. Let us assume that it holds for $i \leq k$, i.e.,

$$x_{2i} = x_i^{CG} \quad i = 0, 1, \ldots, k,$$

so that

$$r_{2i} = f - Ax_{2i} = f - Ax_i^{CG} = r_i^{CG}, \quad i = 0, 1, \ldots, k. \tag{10.4}$$

By Lemma 10.6(a),

$$x_{k+1}^{CG} = x_0^{CG} + u \tag{10.5}$$

where $u \in \{(A^TA)^j A^T r_0^{CG}\}_{j=0}^{k}$. By (9.13b),

$$\tilde{x}_{2(k+1)} = \tilde{x}_0 + z, \tag{10.6}$$

where $z \equiv [z^{(1)}, z^{(2)}]^T \in \{v_1, v_2, \ldots v_{2(k+1)}\}$.

Equating the last n components of both sides of (10.6) and using Lemma 10.5,

$$x_{2(k+1)} = x_0 + z^{(2)}, \quad \text{where } z^{(2)} \in \{(A^TA)^j A^T r_0\}_{j=0}^{k}. \tag{10.7}$$

Now if we show that $z^{(2)} = u$, we would have proved the result for k+1. We do this by considering the orthogonality properties of the residual vectors in the two algorithms.

From (10.5)

$$r_{k+1}^{CG} = f - Ax_{k+1}^{CG} = r_0^{CG} - Au. \tag{10.8}$$

Let w be any vector belonging to $\{A^T r_j^{CG}\}_{j=0}^{k}$. Taking the inner product of equation (10.8) with Aw and using Lemma 10.6(c), we get

$$(r_0^{CG}, Aw) = (Au, Aw). \tag{10.9}$$

We have from (10.7) that

$$r_{2(k+1)} = f - Ax_{2(k+1)} = r_0 - Az^{(2)}. \tag{10.10}$$

Also, by (9.8), we certainly have

$$(\tilde{r}_{2(k+1)}, q) = 0, \quad \forall q \in \{v_1, v_3, v_5, \ldots, v_{2(k+1)-1}\}.$$

Since $\tilde{r}_{2j}$ exists, and is a scalar multiple of v_{2j+1} we have

$$(\tilde{r}_{2(k+1)}, q) = 0, \quad \forall q \in \{\tilde{r}_0, \tilde{r}_2, \ldots, \tilde{r}_{2k}\}.$$

Now by Lemma 10.3, for all j, $\tilde{r}_{2j} = [0, -A^T r_{2j}]$, so that we have

$$(A^T r_{2(k+1)}, w') = 0, \quad \forall w' \in \{A^T r_{2j}\}_{j=0}^{k}.$$

Now, since $\{A^T r_{2j}\}_{j=0}^{k} = \{A^T r_j^{CG}\}_{j=0}^{k}$ by the induction hypothesis (10.4), taking the inner product of (10.10) with Aw', we have

$$(r_0, Aw') = (Az^{(2)}, Aw'), \quad \forall w' \in \{A^T r_j^{CG}\}_{j=0}^{k}. \tag{10.11}$$

Taking w = w', we get from (10.9) and (10.11) that

$$(Au, Aw) = (Az^{(2)}, Aw) \tag{10.12}$$

where $w \in \{A^T r_j^{CG}\}_{j=0}^{k}$.

Since $r_0^{CG} = r_0$, we have from Lemma 10.6(b) that

$$u \text{ and } z^{(2)} \in \{(A^TA)^j A^T r_0^{CG}\}_{j=0}^{k} = \{A^T r_j^{CG}\}_{j=0}^{k}.$$

Hence,

$$u - z^{(2)} \in \{A^T r_j^{CG}\}_{j=0}^{k},$$

and taking $w = u - z^{(2)}$, (10.12) gives

$$(A^TA(u - z^{(2)}), (u - z^{(2)})) = 0$$

which implies that

$$z^{(2)} = u$$

or

$$x_{2(k+1)} = x_{k+1}^{CG}.$$

Thus the Theorem holds for i = k+1.

Q.E.D.

From Section 10.2 we have that computing x_i^{CG} would require approximately (3n +2m + 2q)i multiplications, where q denotes the number of multiplications required to compute matrix vector products of the form Ay or A^Ty. Taking into account the fact that the v_i's and the c_i's are alternatively zero in the first m and last n components of the SYMMBK procedure, the cost of computing x_{2i} is approximately (7(m+n)/2 + 2q)i. Thus the CG method on the normal equations is more efficient for the linear least squares problem.

CHAPTER 11

THE MODIFIED CONJUGATE RESIDUAL METHOD

11.1: Introduction

In Chapter 3, we defined a class of variational iterative methods for solving symmetric linear systems. In this chapter, we apply these methods to solve the linear system

(11.1) $$Ax = f$$

where A is an N x N nonsingular symmetric indefinite matrix.

In Section 2, we derive error bounds for two particular distributions of the eigenvalues of A, and prove that for the indefinite model problem, these methods require $O(n \log n)$ iterations to reduce the error by a factor of n^{-p}, for $p > 0$. Thus, these methods have an advantage over the methods of the preceding chapter, for which no error bounds are known. In Section 3, we present a computational version of these methods.

Luenberger [45] has proposed an extension of the method of conjugate residuals to symmetric indefinite problems. However, even in routine application, the method suffers from computational difficulties which may cause the process to break down [45]. The MCR method [9], which arises from setting $\mu = 2$, can be viewed as a stabilized version of Luenberger's method and provides an efficient technique for solving large sparse symmetric indefinite systems of linear equations. We present this algorithm in Section 4.

In Section 5, we briefly outline some ideas for using preconditioning to accelerate the rate of convergence. In Section 6, we present the results of numerical experiments which illustrate some of the results of the preceding sections and compare the performance of MCR and preconditioned MCR with other methods for solving the symmetric indefinite problem.

11.2: Error Bounds

In Theorem 3.5, we obtained general error bounds for variational iterative methods. We now consider two approaches for getting bounds on the Q_i as defined in (3.9). First, we treat the case in which at most a few eigenvalues lie on one side of the origin.

Theorem 11.1: If the eigenvalues of the matrix A lie in $\Omega \equiv \{\lambda_1, \lambda_2, \ldots, \lambda_m\} \cup [a,b]$, where $1 \leq m < N$, $a, b > 0$, and $\lambda_i < 0$ for $1 \leq i \leq m$, then for $k \geq 0$ the iterates x_{k+m} satisfy

$$\| x - x_{k+m} \|_{A^\mu} \leq 2C \left(\frac{1 - \sqrt{\alpha}}{1 + \sqrt{\alpha}}\right)^k \| x - x_0 \|_{A^\mu}$$

where $\alpha \equiv a/b$ and $C \equiv \prod_{j=1}^{m} \left(\frac{\lambda_j - b}{\lambda_j}\right)$ are constants independent of k.

Proof: From (3.9), for $k \geq 0$,

$$(11.2) \qquad Q_{k+m} \leq \min_{R_{k+m} \in \bar{R}_{k+m}} \max_{z \in \Omega} |R_{k+m}(z)|$$

Now, let $R^*_{k+m}(z) \equiv P_m(z)\tilde{P}_k(z)$, where $P_m(z)$ and $\tilde{P}_k(z)$ are given by

$$P_m(z) \equiv \prod_{i=1}^{m} ((\lambda_i - z)/\lambda_i),$$

$$\tilde{P}_k(z) \equiv \frac{T_k((-2z + b + a)/(b - a))}{T_k((b + a)/(b - a))},$$

where $T_k(z) \equiv \cos(k \arccos z)$ is the k^{th} Chebyshev polynomial. Then

$$R^*_{k+m}(0) = P_m(0)\,\tilde{P}_k(0) = 1,$$

so that $R^*_{k+m} \in \bar{R}_{k+m}$, and from (11.2),

$$(11.3) \qquad Q_{k+m} \leq \max_{z \in \Omega} |R^*_{k+m}(z)|$$

But

$$\max_{z \in [a,b]} |P_m(z)| = \prod_{j=1}^{m} \left(\frac{\lambda_j - b}{\lambda_j}\right),$$

and [27]

$$\max_{z \in [a,b]} |\tilde{P}_m(z)| = \left(T_k\left(\frac{b - a}{b + a} \right) \right)^{-1} = 2\left(\frac{1 - \sqrt{\alpha}}{1 + \sqrt{\alpha}}\right)^k,$$

where $\alpha \equiv a/b$. Moreover

$$P_m(\lambda_i) = 0 \text{ for } 1 \leq i \leq m,$$

so that

$$R^*_{k+m}(\lambda_i) = 0, \quad 1 \leq i \leq m,$$

and hence

$$Q_{k+m} \leq \max_{z \in \Omega} |R^*_{k+m}(z)| = \max_{z \in [a,b]} |R^*_{k+m}(z)| \leq 2C \left(\frac{1 - \sqrt{\alpha}}{1 + \sqrt{\alpha}}\right)^k,$$

where $C \equiv \prod_{j=1}^{m} \left(\frac{\lambda_j - b}{\lambda_j}\right)$.

Substituting in (11.3), the Theorem follows from Theorem 3.5.

Q.E.D.

We now apply this result to obtain the rate of convergence for the indefinite model problem.

Theorem 11.2: For the indefinite model problem, O(n log n) iterations of the variational method suffice to reduce the 2-norm of the initial error by a factor of n^{-p}, for $p > 0$. The number of multiplications sufficient to reduce the error by the same factor are $O(n^3 \log n)$ in two dimensions and $O(n^4 \log n)$ in three dimensions.

Proof: By Theorem 11.1,

$$\| x - x_{k+m} \|_2 \leq 2C \sqrt{\kappa(A^\mu)} \left(\frac{1 - \sqrt{\alpha}}{1 + \sqrt{\alpha}}\right)^k \| x - x_0 \|_2$$

where we have from Section 2.2 that

$$\alpha = c_1^2/n^2, \text{ and } C\sqrt{\kappa(A^\mu)} = c_2 n^q$$

and $c_1, c_2, q > 0$ are constants independent of n.

By an argument similar to that in the Proof of Theorem 4.1, we get that $O(n \log n)$ iterations suffice to reduce the initial error by n^{-p}. Since each iteration requires $O(n^2)$ multiplications in two dimensions and $O(n^3)$ multiplications in three dimensions, the Theorem follows.

Q.E.D.

In the second approach we assume that the eigenvalues of A are contained in two intervals, one on either side of the origin.

Theorem 11.3: If the eigenvalues of the matrix A lie in $\Omega \equiv [-a,-b] \cup [c,d]$ where $a,b,c,d > 0$ and $a - b = d - c > 0$, then for $i \geq 0$, the iterates x_{i+1} satisfy

$$\| x - x_i \|_{A^\mu} \leq 2 \left(\frac{1 - \sqrt{\beta}}{1 + \sqrt{\beta}}\right)^{\lfloor i/2 \rfloor} \| x - x_0 \|_{A^\mu}$$

where $\beta \equiv (bc)/(ad)$.

Proof: Since the eigenvalues of A lie in Ω, we have from (3.9) that

$$(11.4) \qquad Q_{i+1} \leq \min_{R_i \in \mathbb{R}_i} \max_{z \in \Omega} |R_{i+1}(z)| .$$

Lebedev [44] has proved that when i is even, the polynomial $R_i^* \in \mathbb{R}_i$ such that $\max_{z \in \Omega} |R_i^*(z)|$ is minimum over all polynomials in $\mathbb{R}_i$, is unique and is given by

$$P_i(z) = \frac{T_j((-2y + M + m)/(M - m))}{T_j((M + m)/(M - m))} ,$$

where $j = i/2$, $y = z(z - (c - b))$, $m = bc$, $M = ad$, and $T_k(t) = \cos(k \arccos t)$ is the k^{th} Chebyshev polynomial in t. Using properties of Chebyshev polynomials (see [27]),

$$\max_{z \in \Omega} |P_i(z)| = \left(T_j\left(\frac{M - m}{M + m}\right) \right)^{-1} .$$

Since

$$\max_{z \in \Omega} |P_i(z)| \leq \max_{z \in \Omega} |P_{i-1}(z)| ,$$

we can write ($\forall$ i, odd or even)

$$\max_{z \in \Omega} |P_i(z)| \leq \left(T_{\lfloor i/2 \rfloor}\left(\frac{M - m}{M + m}\right) \right)^{-1}$$

$$\leq 2\left(\frac{1 - \sqrt{\beta}}{1 + \sqrt{\beta}}\right)^{\lfloor i/2 \rfloor}, \text{ where } \beta \equiv m/M,$$

again by the properties of Chebyshev polynomials. The Theorem follows from (11.4) and Theorem 3.5.

Q.E.D.

Note that this result requires that the two intervals known to contain the eigenvalues of A be of equal length. If $a - b \neq d - c$, then we can take the smaller of the two intervals and extend it away from the

origin to get intervals of equal length [44] and apply Theorem 11.3 to obtain bounds on the rate of convergence.

11.3: Computational Aspects

In Section 3.4, we showed that if $a_i \neq 0$, the direction vector p_{i+1} can be generated by (3.10) instead of (3.3) and some work can be saved. However, in practice, due to roundoff error, it may not always be possible to decide whether a_i is zero or not. We now show that if $a_i = 0$, but we decide that it is not, and use the short form (3.10) for generating p_{i+1}, then the computational procedure breaks down in some sense.

Lemma 11.1:

(a) If $a_{i-1} \neq 0$, then $a_i = \dfrac{(r_i, A^{\mu-1}r_i)}{(p_i, A^{\mu}p_i)}$ and $b_{i-1} = \dfrac{(r_i, A^{\mu-1}r_i)}{(r_{i-1}, A^{\mu-1}r_{i-1})}$.

(b) If $a_{i-1} = 0$, then $r_i = r_{i-1} = p_{i-1}$.

(c) If $a_{i-1} = 0$ and $r_i \neq \underline{0}$, then $a_i \neq 0$.

Proof: If $a_{i-1} \neq 0$, then

$$(r_i, A^{\mu-1}p_i) = (r_i, A^{\mu-1}r_i) + b_{i-1}(r_i, A^{\mu-1}p_{i-1}),$$

and since the second term on the right-hand side vanishes

by (3.6d), $a_i = \dfrac{(r_i, A^{\mu-1}r_i)}{(p_i, A^{\mu}p_i)}$.

Also,

$$b_{i-1} = \frac{(-r_i, A^{\mu}p_{i-1})}{(p_{i-1}, A^{\mu}p_{i-1})}$$

$$= \frac{(A^{\mu-1}r_i, r_i - r_{i-1})}{a_{i-1}(p_{i-1}, A^{\mu}p_{i-1})}$$

$$= \frac{(A^{\mu-1}r_i, r_i)}{(r_{i-1}, A^{\mu-1}r_{i-1})} \quad \text{using (3.6e)},$$

and (a) is proved.

We prove (b) and (c) by induction. Assume $a_0 = 0$. Then $r_1 = r_0 - a_0 A p_0 = r_0 = p_0$, and if $r_0 \neq 0$,

$$a_1 = \frac{(r_1, A^{\mu-1}(Ap_0 - \gamma_0 p_0))}{(p_1, A^{\mu}p_1)}$$

$$= \frac{(r_1, A^{\mu}p_0)}{(p_1, A^{\mu}p_1)} \quad \text{using (3.6d)}$$

$$= \frac{(r_0, A^{\mu}r_0)}{(p_1, A^{\mu}p_1)} \neq 0.$$

Assume (b) and (c) hold for $i \leq k$ and $a_k = 0$.

First consider the case $r_k = 0$. Then

(i) if $a_{k-1} = 0$, by the induction hypothesis we have that

$$r_{k-1} = r_k = p_{k-1} = 0.$$

By the definition of the p's this gives $p_k = 0$, and by the definition of the r's, we get $r_{k+1} = 0$. Hence $r_{k+1} = r_k = p_k$.

(ii) if $a_{k-1} \neq 0$, (a) gives $b_{k-1} = 0$. Thus by the definition of the p's, we get $p_k = r_k = 0$, and by the definition of the r's we get $r_{k+1} = r_k$. Hence $r_{k+1} = r_k = p_k$.

Next, assume that $r_k \neq \underline{0}$. By the induction hypothesis, $a_{k-1} \neq 0$ (for if $a_{k-1} = 0$, then $r_k \neq \underline{0}$ implies $a_{k-1} \neq 0$). Since $a_k = 0$, from Part (a) it follows that $(r_k, A^{\mu-1} r_k) = 0$, so that $b_{k-1} = 0$. Thus

$$p_k = r_k + b_{k-1} p_{k-1} = r_k = r_{k+1},$$

since $r_{k+1} = r_k - a_k A p_k$ and $a_k = 0$.
So (b) holds for $i = k+1$. Now,

$$\begin{aligned}(A^{\mu-1} r_{k+1}, p_{k+1}) &= (A^{\mu-1} r_{k+1}, Ap_k - \gamma_k p_k - \delta_k p_{k-1}) \\ &= (A^{\mu-1} r_{k+1}, Ap_k) \quad \text{using (3.6d)} \\ &= (A^{\mu-1} r_k, Ap_k) \quad \text{since } r_{k+1} = r_k = p_k \\ &\neq 0, \quad \text{since } p_k = r_k \neq 0.\end{aligned}$$

Thus

$$a_{k+1} = \frac{(r_{k+1}, A^{\mu-1} p_{k+1})}{(p_{k+1}, A^{\mu} p_{k+1})} \neq 0,$$

and (c) holds for $i = k+1$.

Q.E.D.

Theorem 11.4: If $a_i = 0$ and

$$p_{i+1} = r_{i+1} + b_i p_i$$

where

$$b_i = \frac{(-r_{i+1}, A^{\mu} p_i)}{(p_i, A^{\mu} p_i)},$$

then $p_{i+1} = \underline{0}$.

Proof: By Lemma 11.1(b), $r_{i+1} = p_i$ and so $b_i = -1$. Thus

$$p_{i+1} = r_{i+1} - b_i p_i = \underline{0}.$$

Q.E.D.

Thus if, at some iteration i+1, the direction vector is generated by the shorter process even though $a_i = 0$, then the direction vector p_{i+1} is the zero vector. If this error is not detected, the iteration process may become highly unstable, since, in the next iteration, p_{i+1} occurs in the denominator of the expressions for the scalars a_{i+1}, b_{i+1}, γ_{i+1}, and δ_{i+1}.

We overcome this difficulty as follows. We choose, a priori, a threshold ϵ, and use the shorter iteration only when $|a_i| > \epsilon$. The threshold $\epsilon > 0$ is chosen large enough that roundoff will not allow the

new direction vector to be generated by (3.10) if $a_i = 0$. Note, however, that if $0 < |a_i| < \epsilon$, then we are using the longer iteration at step i, even though the shorter one is sufficient.

We now give a simpler formula for obtaining δ_i.

Theorem 11.5:

$$\delta_i = c_i(p_i, A^\mu p_i)/(p_{i-1}, A^\mu p_{i-1}),$$

where $$c_i = \begin{cases} 1 & \text{if } |a_{i-1}| \le \epsilon \\ -1/a_{i-1} & \text{if } |a_{i-1}| > \epsilon. \end{cases}$$

Proof: If $|a_{i-1}| \le \epsilon$, then

$$(A^\mu p_i, Ap_{i-1}) = (A^\mu p_i,\ p_i + \gamma_{i-1}p_{i-1} + \delta_{i-1}p_{i-2})$$

$$= (A^\mu p_i, p_i)$$

since the p's are mutually A^μ-orthogonal.

Therefore $c_i = 1$.

If $|a_{i-1}| > \epsilon$, then by (3.10),

$$p_i = -a_{i-1}(Ap_{i-1} - \gamma_{i-1}p_{i-1} - \delta_{i-1}p_{i-2})$$

and so

$$(A^\mu p_i, Ap_{i-1}) = (A^\mu p_i, -\frac{p_i}{a_{i-1}} + \gamma_{i-1}p_{i-1} + \delta_{i-1}p_{i-2})$$

$$= -\frac{1}{a_{i-1}}(A^\mu p_i, p_i).$$

Therefore $c_i = -\frac{1}{a_{i-1}}$.

Q.E.D.

Making the above modifications in Algorithm 3.2 we obtain the following version of the variational method for any symmetric indefinite A.

Algorithm 11.1: The Variational Method for Indefinite Systems

Step 1: Choose x_0 and $\epsilon > 0$.

Compute $r_0 = f - Ax_0$.

Set $p_0 = r_0$,

and $i = 0$.

Step 2: Compute

$$a_i = (r_i, A^{\mu-1}p_i)/(p_i, A^\mu p_i),$$

$$x_{i+1} = x_i + a_i p_i,$$

and $$r_{i+1} = r_i - a_i Ap_i.$$

Step 3: If $|a_i| \le \epsilon$, go to Step 4a;

else go to Step 4b.

Step 4a: Set $$c_i = \begin{cases} 1 & \text{if } |a_{i-1}| \le \epsilon, \\ -1/a_{i-1} & \text{if } |a_{i-1}| > \epsilon. \end{cases}$$

Compute

$$\delta_i = c_i(p_i, A^\mu p_i)/(p_{i-1}, A^\mu p_{i-1}),$$

$\gamma_i = (Ap_i, A^{\mu}p_i)/(p_i, A^{\mu}p_i)$,

and $p_{i+1} = Ap_i - \gamma_i p_i - \delta_i p_{i-1}$.

Go to Step 5.

Step 4b: Compute

$b_i = (-r_{i+1}, A^{\mu}p_i)/(p_i, A^{\mu}p_i)$,

and $p_{i+1} = r_{i+1} + b_i p_i$.

Step 5: If x_{i+1} is sufficiently close to x, terminate the iteration process;

else set $i = i + 1$ and go to Step 2.

11.4: The Modified Conjugate Residual (MCR) Method

The iteration of Algorithm 11.1 takes on its simplest form for $\mu = 2$. The iterates of the resulting method, the modified conjugate residual (MCR) method, then satisfy the error bounds derived in Section 11.2 and the properties of Section 3.2. In particular, the 2-norm of the residual is minimized over subspaces of increasing dimension, and, therefore, decreases monotonically during the iteration process.

Algorithm 11.2: The Modified Conjugate Gradient (MCR) Method

Step 1: Choose x_0 and $\epsilon > 0$.

Compute $r_0 = f - Ax_0$.

Set $p_0 = r_0$,

and $i = 0$.

Step 2: Compute

$a_i = (r_i, Ap_i)/(Ap_i, Ap_i)$,

$x_{i+1} = x_i + a_i p_i$,

and $r_{i+1} = r_i - a_i Ap_i$.

Step 3: If $|a_i| \leq \epsilon$, go to Step 4a.

else go to Step 4b.

Step 4a: Set $c_i = \begin{cases} 1 & \text{if } |a_{i-1}| \leq \epsilon. \\ -1/a_{i-1} & \text{if } |a_{i-1}| > \epsilon. \end{cases}$

Compute

$\delta_i = c_i(Ap_i, Ap_i)/(p_{i-1}, Ap_{i-1})$,

$\gamma_i = (Ap_i, A^2 p_i)/(Ap_i, Ap_i)$,

$p_{i+1} = Ap_i - \gamma_i p_i - \delta_i p_{i-1}$,

and $Ap_{i+1} = A(Ap_i) - \gamma_i Ap_i - \delta_i Ap_{i-1}$.

Go to Step 5.

Step 4b: Compute

$b_i = (-Ar_{i+1}, Ap_i)/(Ap_i, Ap_i)$,

$p_{i+1} = r_{i+1} + b_i p_i$,

and $Ap_{i+1} = Ar_{i+1} + b_i Ap_i$.

Step 5: If x_{i+1} is sufficiently close to x, terminate the iteration process;
else set i = i + 1 and go to Step 2.

Each iteration of Algorithm 11.2 requires only one matrix-vector product, viz., $A(Ap_i)$ if $|a_i| \leq \epsilon$, and Ar_{i+1} if $|a_i| > \epsilon$. In addition, the iteration requires 9N + 4 multiplications if $|a_i| \leq \epsilon$ and 7N + 2 multiplications if $|a_i| > \epsilon$. Storage is required for the seven vectors x_i, r_i, p_i, Ap_i, p_{i-1}, Ap_{i-1}, and Ar_{i+1}. If $|a_i| \leq \epsilon$, then $A(Ap_i)$ can be stored in place of Ar_{i+1}. Storage and work requirements for the model and generalized model problems are given in Table 11.1.

Luenberger's extension [45] of the conjugate residual method to symmetric indefinite problems is the same as Algorithm 11.2 with $\epsilon = 0$, except that if $a_i = 0$, then Luenberger generates the new direction vector by

$$p_{i+1} = Ar_{i+1} - \gamma_i p_i - \delta_i p_{i-1}$$

where

$$\gamma_i = \frac{(Ar_{i+1}, A^2 p_i)}{(Ap_i, Ap_i)}, \quad \text{and} \quad \delta_i = \frac{(Ar_{i+1}, A^2 p_{i-1})}{(Ap_{i-1}, Ap_{i-1})}.$$

By Lemma 11.1(b), if $\epsilon = 0$ and $a_i = 0$, then $r_{i+1} = p_i$. Thus, Luenberger's method is mathematically equivalent to the MCR algorithm with $\epsilon = 0$, and therefore suffers from the computational instability

described in Section 3.4.

Fletcher [30] proposed another way to get around the difficulty with applying the CR method to indefinite systems. His algorithm corresponds to the MCR algorithm with $\epsilon = \infty$, so that it does not take advantage of the fact that there is a less expensive scheme for generating the direction vector p_{i+1} if a_i is not too small.

Paige and Saunders proposed MINRES [52], another method that attempts to to avoid the instability in the CR method. They use an approach similar to the one for SYMMLQ (see Section 9.4), in which they carry out the same stable factorization of a tridiagonal matrix.

11.5: The Preconditioned Modified Conjugate Residual Method

In this Section, we consider an approach similar to that adopted for positive definite problems in Chapter 4, and attempt to apply preconditioning in an effort to increase the rate of convergence of the variational method for symmetric indefinite systems. We choose a splitting

(11.5) $\qquad A = M - R$

where M is positive definite and apply the Algorithm 11.1 to solve the symmetric linear system

(11.6) $\qquad A'x' = f'$

where $A' = Q^{-1}AQ^{-T}$, $M = QQ^T$, $x' = Q^Tx$, and $f' = Q^{-1}f$. As in the case of positive definite A, it turns out that we do not have to actually compute A'. Simple algebraic manipulation gives us the following equivalent algorithm.

Algorithm 11.3: The Preconditioned Variational Method for Indefinite Systems

Step 1: Choose x_0 and $\epsilon > 0$.

Compute $r_0 = f - Ax_0$.

Solve $M\bar{r}_0 = r_0$.

Set $p_0 = \bar{r}_0$,

and $i = 0$.

Step 2: Compute

$$a_i = (\bar{r}_i, A(M^{-1}A)^{\mu-2}p_i)/(p_i, A(M^{-1}A)^{\mu-1}p_i),$$

$$x_{i+1} = x_i + a_ip_i,$$

$$r_{i+1} = r_i - a_iAp_i,$$

and $\bar{r}_{i+1} = \bar{r}_i - a_iM^{-1}Ap_i$.

Step 3: If $|a_i| \leq \epsilon$, go to Step 4a;

else go to Step 4b.

Step 4a: Set $c_i = \begin{cases} 1 & \text{if } |a_{i-1}| \leq \epsilon. \\ -1/a_{i-1} & \text{if } |a_{i-1}| > \epsilon. \end{cases}$

Compute

$$\delta_i = c_i(p_i, A(M^{-1}A)^{\mu-1}p_i)/(p_{i-1}, A(M^{-1}A)^{\mu-1}p_{i-1}),$$

$$\gamma_i = (Ap_i, (M^{-1}A)^{\mu}p_i)/(p_i, A(M^{-1}A)^{\mu-1}p_i),$$

and $p_{i+1} = M^{-1}Ap_i - \gamma_ip_i - \delta_ip_{i-1}$.

Go to Step 5.

Step 4b: Compute

$$b_i = (-A\bar{r}_{i+1}, (M^{-1}A)^{\mu-1}p_i)/(p_i, A(M^{-1}A)^{\mu-1}p_i),$$

and $p_{i+1} = \bar{r}_{i+1} + b_ip_i$.

Step 5: If x_{i+1} is sufficiently close to x, terminate the iteration process;

else set $i = i + 1$ and go to Step 2.

In particular, setting $\mu = 2$ gives the preconditioned MCR method for solving (11.1).

Algorithm 11.4: The Preconditioned Modified Conjugate Residual Method

Step 1: Choose x_0 and $\epsilon > 0$.

Compute $r_0 = f - Ax_0$.

Solve $M\bar{r}_0 = r_0$.

Set $p_0 = \bar{r}_0$,

and $i = 0$.

Step 2: Solve $q_i = M^{-1}(Ap_i)$.

Compute $a_i = (\bar{r}_i, Ap_i)/(Ap_i, q_i)$,

$x_{i+1} = x_i + a_ip_i$,

and $\tilde{r}_{i+1} = \tilde{r}_i - a_i q_i$.

Step 3: If $|a_i| \leq \epsilon$, go to Step 4a;
else go to Step 5.

Step 4a: Set $c_i = \begin{cases} 1 & \text{if } |a_{i-1}| \leq \epsilon. \\ -1/a_{i-1} & \text{if } |a_{i-1}| > \epsilon. \end{cases}$

Compute

$$\delta_i = c_i(Ap_i, q_i) \,/\, (Ap_{i-1}, q_{i-1}),$$

$$\gamma_i = (Aq_i, q_i)/(Ap_i, q_i),$$

$$p_{i+1} = q_i - \gamma_i p_i - \delta_i p_{i-1},$$

and $Ap_{i+1} = Aq_i - \gamma_i Ap_i - \delta_i Ap_{i-1}$.

Step 4b: Compute

$$b_i = (-A\tilde{r}_{i+1}, q_i) \,/\, (Ap_i, q_i),$$

$$p_{i+1} = \tilde{r}_{i+1} + b_i p_i,$$

and $Ap_{i+1} = A\tilde{r}_{i+1} + b_i Ap_i$.

Step 5: If x_{i+1} is sufficiently close to x, terminate the iteration process;
else set i = i + 1 and go to Step 2.

Each iteration of Algorithm 11.4 requires only one matrix-vector product, viz. Aq_i if $|a_i| \leq \epsilon$ and $A\tilde{r}_{i+1}$ if $|a_i| > \epsilon$, and one solve of the form Mu = v. In addition, each iteration requires 9N + 4 multiplications if $|a_i| \leq \epsilon$ and 7N + 2 multiplications if $|a_i| > \epsilon$.

Storage is required for the eight vectors x_i, $\tilde{r}_i$, p_i, Ap_i, q_i, Aq_i, p_{i-1}, and Ap_{i-1}. $A\tilde{r}_{i+1}$ can share storage with Aq_i.

Note that the true residual $r_{i+1} = f - Ax_{i+1}$ is not computed but may be obtained easily by

$$r_{i+1} = M\tilde{r}_{i+1}$$

or computed recursively as

$$r_{i+1} = r_i - a_i Ap_i.$$

Error bounds can be obtained by applying Theorems 11.1 and 11.3 to (11.6).

Theorem 11.6: If the eigenvalues of the matrix $M^{-1}A$ lie in $\Omega \equiv \{\lambda_1, \lambda_2, \ldots \lambda_m\} \cup [a,b]$ where $1 \leq m < N$, a, b, > 0, and $\lambda_i < 0$ for $1 \leq i \leq m$, then for $k \geq 0$ the iterates x_{k+m} given by the preconditioned variational method satisfy

$$\|x - x_{k+m}\|_{A(M^{-1}A)^{\mu-1}} \leq 2C \left(\frac{1 - \sqrt{\alpha}}{1 + \sqrt{\alpha}}\right)^k \|x - x_0\|_{A(M^{-1}A)^{\mu-1}}$$

where $\alpha \equiv a/b$ and $C \equiv \prod_{j=1}^{m} \left(\frac{\lambda_j - b}{\lambda_j}\right)$ is a constant independent of k.

Theorem 11.7: If the eigenvalues of the matrix $M^{-1}A$ lie in $\Omega \equiv [-a, -b] \cup [c, d]$ where a, b, c, d > 0 and a - b = d - c > 0, then for $i \geq 0$, the iterates x_i given by the preconditioned variational method satisfy

$$\|x - x_i\|_{A(M^{-1}A)^{\mu-1}} \leq 2 \left(\frac{1 - \sqrt{\beta}}{1 + \sqrt{\beta}}\right)^{\lfloor i/2 \rfloor} \|x - x_0\|_{A(M^{-1}A)^{\mu-1}}$$

where $\beta \equiv (bc)/(ad)$.

We would like to choose a preconditioning, or equivalently the splitting in (11.5), which increases the rate of convergence of these methods. Typically, M should be "close" to A in some sense, and systems of the form $Mu = v$ should be easy to solve. Since A is not positive definite, the preconditionings based on the approximate factorization of A discussed in Chapter 6 are, in general, not applicable. Instead consider taking some matrix $\tilde{A}$ which is positive definite, "close" to A, and choosing M based on the splitting of $\tilde{A}$ rather than that of A. The analysis of this approach seems difficult even for the simple case of the indefinite model problem. It may however give fast convergence in practice. We give some experimental results in the next section.

We now consider symmetric indefinite matrices of the form

$$A = M - sI \tag{11.7}$$

where M is positive definite, $s > 0$ is a scalar constant, and systems of the form $Mu = v$ can be solved easily. We further assume that a few fixed number (say m) of the eigenvalues of A are negative. Let

$$\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_m < 0 < \lambda_{m+1} \leq \cdots \leq \lambda_N,$$

where $\{\lambda_i\}_{i=1}^N$ are the eigenvalues of A. Then the eigenvalues of M are $\{\lambda_i + s\}_{i=1}^N$ and those of $M^{-1}A = I - sM^{-1}$ are given by

$$\gamma_i = 1 - \frac{s}{\lambda_i + s} = \frac{\lambda_i}{\lambda_i + s}, \quad i = 1, 2, \ldots N$$

Now $\lambda_i + s > 0$ for all i, since M is positive definite, and $\lambda_i \neq 0$ since A is nonsingular; this implies that γ_i is non-zero, has the same sign as λ_i, and $\gamma_i < \gamma_j$ iff $\lambda_i < \lambda_j$ for all i and j. Thus

$$\gamma_1 \leq \gamma_2 \cdots \leq \gamma_m < 0 < \gamma_{m+1} \leq \cdots \leq \gamma_N,$$

Then for the preconditioned MCR method, we have by Theorem 11.6, the following convergence result.

$$\| f - Ax_{k+m} \|_2 \leq 2C \sqrt{\kappa(M^{-1})} \left(\frac{1 - \sqrt{\alpha}}{1 + \sqrt{\alpha}}\right)^k \| f - Ax_0 \|_2$$

where

$$\alpha \equiv \frac{\gamma_{m+1}}{\gamma_N} = \frac{\lambda_{m+1}}{\lambda_N} \frac{\lambda_N + s}{\lambda_{m+1} + s} \quad \text{and} \quad C \equiv \prod_{j=1}^{m} \frac{(\gamma_j - \gamma_N)}{\gamma_j}.$$

For the indefinite model problem, $\kappa(M^{-1}) = O(n^2)$ and $\frac{\lambda_N}{\lambda_{m+1}} = O(n^2)$. Therefore, $\alpha = O(1)$ and

$$C = \prod_{j=1}^{m} \frac{s(\lambda_j - \lambda_N)}{\lambda_j(\lambda_N + s)} = O(1).$$

By an argument similar to that for the Proof of Theorem 4.2, we get that O(log n) iterations of the preconditioned MCR method suffice to reduce the 2-norm of the residual by a factor of n^{-p}, $p > 0$.

11.6: Numerical Results

In this Section, we present the results of numerical experiments which demonstrate the performance of the MCR method on the two- and three-dimensional indefinite model problems. Values of σ and h were

chosen as in Section 9.7, and we took $\epsilon = 10^{-4}$ as the threshold. Figures 11.1a and 11.1b show the error versus number of iterations. Tables 11.2 and 11.3 give the total work required to reduce the error to 10^{-6}.

The numbers in these tables and in Figures 11.2a and 11.2b indicate that SYMMBK and MCR are very similar in their storage and work requirements for the model problem. The MCR method however has the advantage that theoretical error bounds are known.

Choosing the solution as in (2.7), we plot the number of iterations required to reduce the error by a factor of $1/n^2$ for various σ and h against (n log n). For particular problems, i.e., for fixed σ, the plots are straight lines (see Figures 11.3a and 11.3b). This illustrates the O(n log n) convergence result for the model problems obtained in Theorem 11.2.

Table 11.4 gives the storage and work requirements for the preconditioned-MCR method for the indefinite model and general problems. The preconditioning matrix M is taken to be the Dupont, Kendall, and Rachford factorization of the positive definite matrix corresponding to the case $\sigma = 0$ (i.e., the positive definite model problem). A comparison of Tables 11.2 and 11.5 shows that if σ and h are small, then the preconditioning is effective in improving the rate of convergence of MCR. Similar results were obtained for the three-dimensional indefinite model problem and with other preconditionings.

Problem	Two dimensions ($N=n^2$)			Three dimensions ($N=n^3$)		
	Storage reqd.	No. of mults./ iteration		Storage reqd.	No. of mults./ iteration	
		$\lvert a_i\rvert>\epsilon$	$\lvert a_i\rvert\leq\epsilon$		$\lvert a_i\rvert>\epsilon$	$\lvert a_i\rvert\leq\epsilon$
Model	7N	8N+2	10N+4	7N	8N+2	10N+4
General	10N-2n	12N-4n+2	14N-4n+4	11N-3n^2	14N-6n^2+2	16N-6n^2+4

Table 11.1: Storage requirements and multiplication counts for MCR for N x N systems

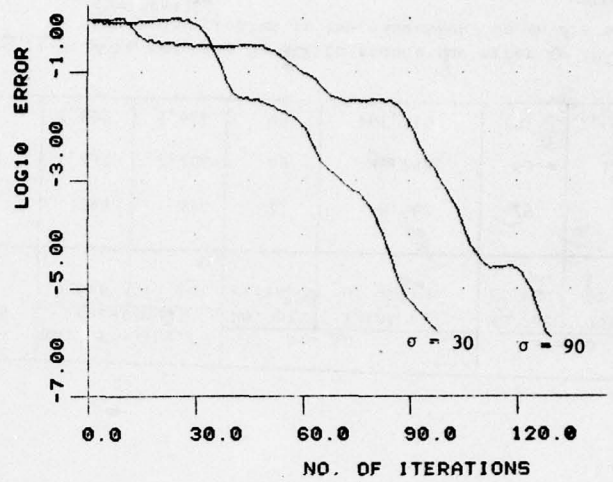
Mesh-width h	No. of mults./ iteration		$\sigma = 30$		$\sigma = 90$	
	$\lvert a_i\rvert>\epsilon$	$\lvert a_i\rvert\leq\epsilon$	No. of iters.	Total no. of mults.	No. of iters.	Total no. of mults.
1/8	394	494	22	8,668	29	11,426
1/16	1,802	2,254	49	88,298	63	113,526
1/32	7,690	9,614	99	761,310	131	1,007,390

Table 11.2: Work required by MCR to reduce the error to 10^{-6} for the model problem in two dimensions on an n x n mesh (h=1/(n+1))

Mesh-width h	No. of mults./iteration $\|a_i\|>\epsilon$	No. of mults./iteration $\|a_i\|\leq\epsilon$	σ = 50 No. of iters.	σ = 50 Total no. of mults.	σ = 100 No. of iters.	σ = 100 Total no. of mults.
1/4	218	274	9	1,962	8	1,744
1/8	2,746	3,434	32	87,872	52	142,792
1/16	27,002	33,754	71	1,917,142	97	2,619,194

Table 11.3: Work required by MCR to reduce the error to 10^{-6} for the model problem in three dimensions on an n x n x n mesh (h=1/(n+1))

(a) Two dimensions
(h = 1/32)



(b) Three dimensions
(h = 1/16)

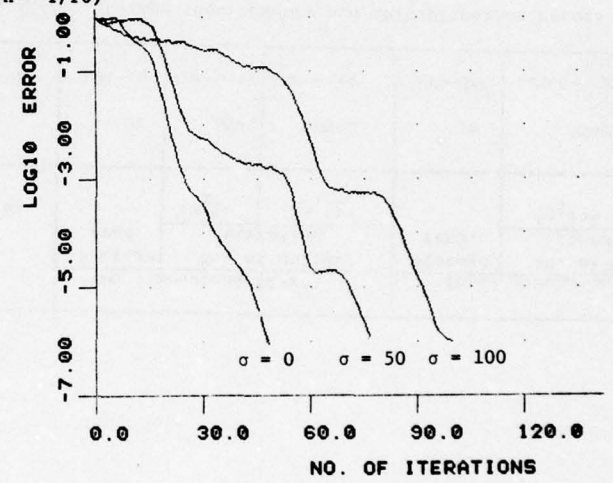
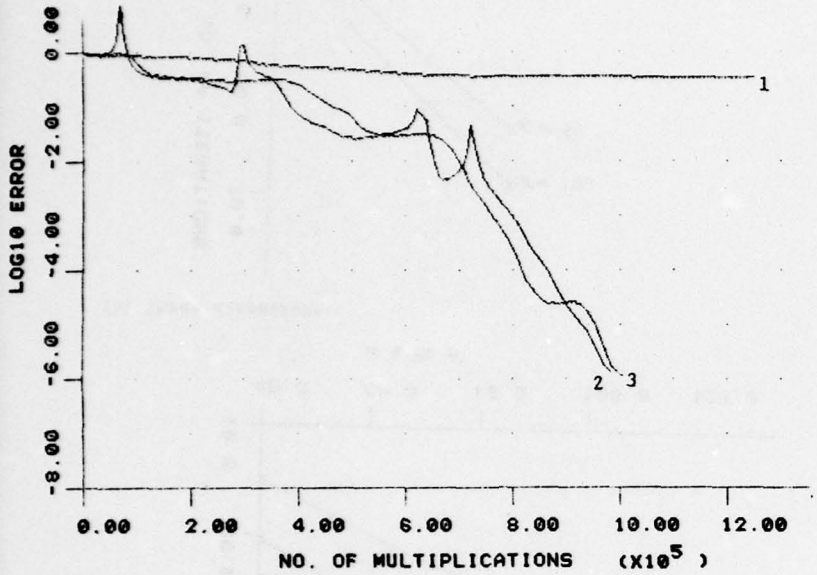
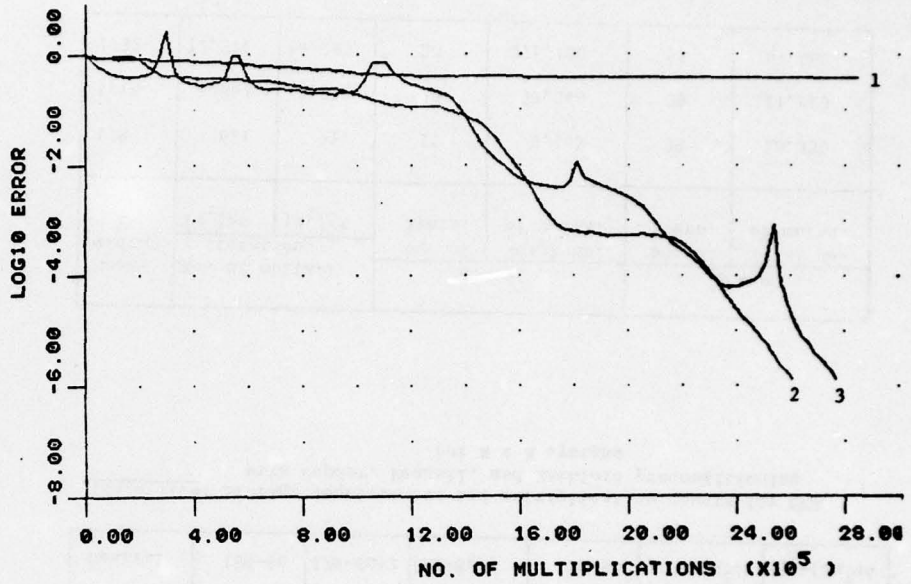


Figure 11.1: Error reduction by MCR for indefinite model problem



1 CG on Normal Equations
2 SYMMBK (Gamma = .078)
3 MCR

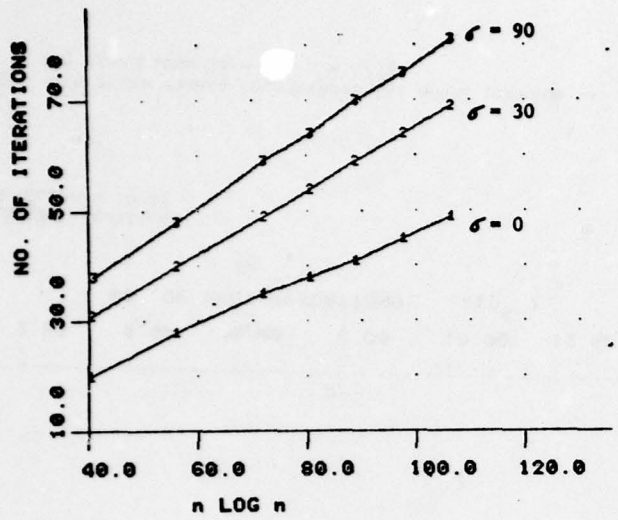
Figure 11.2a: Work requirements for methods for model problem in two dimensions (σ = 90, h = 1/32)



1 CG on Normal Equations
2 SYMMBK (Gamma = .054)
3 MCR

Figure 11.2b: Work requirements for methods for model problem in three dimensions (σ = 100, h = 1/16)

(a) Two dimensions



(b) Three dimensions

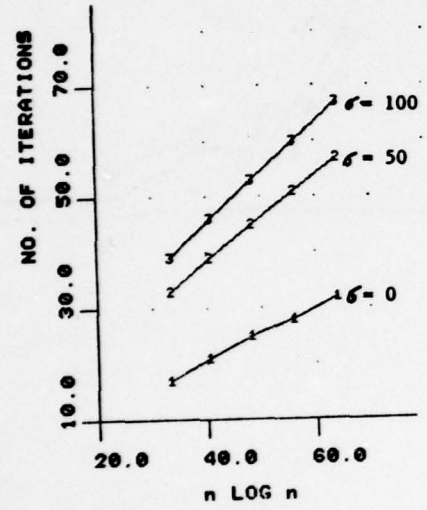


Figure 11.3: Graphs illustrating O(n log n) convergence result for MCR on indefinite model problem

Problem	Two dimensions ($N=n^2$)			Three dimensions ($N=n^3$)		
	Storage reqd.	No. of mults./ iteration		Storage reqd.	No. of mults./ iteration	
		$\|a_i\|>\epsilon$	$\|a_i\|\leq\epsilon$		$\|a_i\|>\epsilon$	$\|a_i\|\leq\epsilon$
Model	8N	13N-4n+2	15N-4n+4	8N	15N-6n^2+2	17N-6n^2+4
General	16N-6n	17N-8n+2	19N-8n+4	19N-9n^2	21N-12n^2+2	23N-12n^2+4

Table 11.4: Storage requirements and multiplication counts for MCR with Dupont, Kendall, and Rachford preconditioning for N x N systems

Mesh-width h	No. of mults./ iteration		σ = 30		σ = 90	
	$\|a_i\|>\epsilon$	$\|a_i\|\leq\epsilon$	No. of iters.	Total no. of mults.	No. of iters.	Total no. of mults.
1/8	611	711	15	9,165	30	18,330
1/16	2,867	3,319	20	57,340	39	111,813
1/32	12,371	14,295	30	371,130	62	767,002

Table 11.5: Work required by MCR with Dupont, Kendall, and Rachford preconditioning (σ = 0) to reduce the error to 10^{-6} for the model problem in two dimensions on an n x n mesh (h=1/(n+1))

APPENDIX A

We show that for the model problem in two and three dimensions, the condition number of the ICCG(0) iteration matrix is bounded below by cn^2 for some positive constant c.

For ICCG(0), the relations (6.13) define the approximate L'U' factorization of A for the two-dimensional generalized model problem. As mentioned in Section 6.4.2, we can write

$$L'U' = LL^T$$

where

$$L' = LD,$$

$$U' = D^{-1}L^T,$$

$$D = \text{diag}(L),$$

L has the same non-zero structure as the lower triangle of A, and consider instead the approximate LL^T decomposition of A. From (6.13) it follows that for the two-dimensional model problem, we can write

$$L = \begin{bmatrix} \sqrt{v_1} & & & & & \\ -1/\sqrt{v_1} & \sqrt{v_2} & & & 0 & \\ & \ddots & \ddots & & & \\ -1/\sqrt{v_1} & & & & & \\ & \ddots & & \ddots & & \\ & & \ddots & & \ddots & \\ 0 & & & -1/\sqrt{v_{N-n}} & -1/\sqrt{v_{N-1}} & \sqrt{v_N} \end{bmatrix}$$

where

$$\text{(A.1)} \quad \ell_{ij} = \begin{cases} 0, & \text{if } a_{ij} = 0, \\ -1/\sqrt{v_j}, & \text{if } a_{ij} \neq 0 \text{ and } i \neq j, \\ \sqrt{v_j}, & \text{if } a_{ij} \neq 0 \text{ and } i = j; \end{cases}$$

and

$$v_i = 4 - \frac{\delta_i}{v_{i-1}} - \frac{\gamma_i}{v_{i-n}}, \quad 1 \leq i \leq N,$$

$$\text{(A.2)} \quad \text{where } \delta_i = \begin{cases} 0, & i = mn + 1, \quad m = 0,1,\ldots(n-1), \\ 1, & \text{otherwise,} \end{cases}$$

$$\text{and } \gamma_i = \begin{cases} 0, & i \leq n, \\ 1, & \text{otherwise.} \end{cases}$$

We first prove the following result.

Lemma A.1: For the model problem in two dimensions†,

(a) $2 + \sqrt{2} < v_i \leq 4$, for $1 \leq i \leq N$,

(b) L is strictly diagonally dominant,

and

(c) if $D \equiv \text{diag}(L)$ and $\tilde{L} \equiv I - D^{-1}L$,

$$\|D^{-1}\|_\infty = (2 + \sqrt{2})^{-1/2},$$

$$\|L\|_\infty, \quad \|L^T\|_\infty \leq 2 + 2/(2 + \sqrt{2})^{1/2},$$

and $\|\tilde{L}\|_\infty, \quad \|\tilde{L}^T\|_\infty \leq 2/(2 + \sqrt{2})$.

---

† For the model problem in three dimensions, an analogous result is obtained.

Proof: We first prove (a) by induction on i. Since $v_1 = 4$, (a) holds for $i = 1$. Assume it holds for $i \leq k$. Then by (A.2) and the induction hypothesis,

$$v_{k+1} \geq 4 - 2 / (\min_{1 \leq j \leq k} v_j) > 4 - 2/(2 + \sqrt{2}) = 2 + \sqrt{2},$$

and

$$v_{k+1} \leq 4 - \frac{\delta_{k+1}}{\max_{1 \leq j \leq k} v_j} - \frac{\gamma_{k+1}}{\max_{1 \leq j \leq k} v_j} \leq 4,$$

so that (a) holds for $k+1$.

Moreover, for any i

$$\sum_{\substack{j=1 \\ i \neq j}}^{N} |\ell_{ij}| = 1/\sqrt{v_{i-1}} + 1/\sqrt{v_{i-n}} \leq 2/ \min_{1 \leq j \leq k} \sqrt{v_j} < 2/(2 + \sqrt{2})^{1/2}$$

and

$$\ell_{ii} = \sqrt{v_i} > (2 + \sqrt{2})^{1/2}.$$

Thus L is strictly diagonally dominant.

We note that

$$\|D^{-1}\|_\infty = \max_i 1/\sqrt{v_i}$$

$$\|L\|_\infty, \quad \|L^T\|_\infty \leq \max_i \sqrt{v_i} + 2/ \min_i \sqrt{v_i},$$

and

$$\|\tilde{L}\|_\infty, \quad \|\tilde{L}^T\|_\infty \leq \max_i 2/v_i.$$

Then, using (a), (c) follows.

Q.E.D.

We now prove the following general result for strictly diagonally dominant matrices with positive diagonal elements.

Lemma A.2: Let B be a strictly diagonally dominant matrix with positive diagonal elements. Then

$$\lambda(BB^T) \in [k_1, k_2]$$

where

$$k_1 = \frac{(1 - \|\tilde{B}\|_\infty)(1 - \|\tilde{B}^T\|_\infty)}{\|D^{-1}\|_\infty^2},$$

$$k_2 = \|B\|_\infty \|B^T\|_\infty,$$

$$\tilde{B} = I - D^{-1}B,$$

and

$$D = \text{diag}(B).$$

Proof: We have

$$\lambda_{\max}((BB^T)^{-1}) \leq \|(BB^T)^{-1}\|_\infty \leq \|B^{-1}\|_\infty \|B^{-T}\|_\infty$$

$$\leq \|D^{-1}\|_\infty^2 \|(I - \tilde{B})^{-1}\|_\infty \|(I - \tilde{B}^T)^{-1}\|_\infty \tag{A.3}$$

The matrix $D^{-1}B$ is strictly diagonally dominant and has all its diagonal terms equal to 1. Hence the sum of the elements in any row of $I - D^{-1}B$ is less than 1. It follows that $\|\tilde{B}\|_\infty < 1$, and hence $\rho(\tilde{B}) < 1$.

Then, (see Varga [60], Theorem 3.7)

$$(I - \tilde{B})^{-1} = I + \tilde{B} + \tilde{B}^2 + \ldots$$

and so

$$\| (I - \tilde{B})^{-1} \|_\infty \leq \sum_{k=0}^{\infty} \| \tilde{B}^k \|_\infty$$

$$\leq \sum_{k=0}^{\infty} \| \tilde{B} \|_\infty^k$$

$$= 1 / (1 - \| \tilde{B} \|_\infty)$$

since $\| \tilde{B} \|_\infty < 1$.

Proceeding as above,

$$\| (I - \tilde{B}^T)^{-1} \|_\infty \leq 1 / (1 - \| \tilde{B}^T \|_\infty).$$

Substituting in (A.3),

$$\lambda_{max}(BB^T)^{-1} \leq \frac{\| D^{-1} \|_\infty^2}{(1 - \| \tilde{B} \|_\infty) (1 - \| \tilde{B}^T \|_\infty)}$$

Now, since $\lambda_{min}(BB^T) = \dfrac{1}{\lambda_{max}((BB^T)^{-1})}$,

and

$$\lambda_{max}(BB^T) \leq \| BB^T \|_\infty \leq \| B \|_\infty \| B^T \|_\infty ,$$

the Lemma follows.

Q.E.D.

Theorem A.1: For the model problem in two dimensions†,

$$\frac{1}{17} \kappa(A) \leq \kappa(L^{-1}AL^{-T}) \leq 17 \kappa(A).$$

Proof: For $\forall\, x \neq 0$, we can write

$$(A.4) \qquad \frac{(x,L^{-1}AL^{-T}x)}{(x,x)} = \frac{(y,Ay)/(y,y)}{(y,LL^Ty)/(y,y)}$$

where $y = L^{-T}x$. It follows that

---

† For the model problem in three dimensions, the result holds with the constant 17 replaced by 49.

$$\lambda_{max}(L^{-1}AL^{-T}) \leq \frac{\lambda_{max}(A)}{\lambda_{min}(LL^T)}$$

and

$$\lambda_{min}(L^{-1}AL^{-T}) \geq \frac{\lambda_{min}(A)}{\lambda_{max}(LL^T)}$$

so that

$$(A.5) \qquad \kappa(L^{-1}AL^{-T}) \leq \kappa(LL^T)\, \kappa(A).$$

Rewriting (A.4) as

$$\frac{(y,Ay)}{(y,y)} = \frac{(x,L^{-1}AL^{-T}x)}{(x,x)} \; \frac{(y,LL^Ty)}{(y,y)}$$

we have that

$$\lambda_{max}(A) \leq \lambda_{max}(L^{-1}AL^{-T})\, \lambda_{max}(LL^T)$$

and

$$\lambda_{min}(A) \geq \lambda_{min}(L^{-1}AL^{-T})\, \lambda_{min}(LL^T)$$

so that

$$(A.6) \qquad \frac{\kappa(A)}{\kappa(LL^T)} \leq \kappa(L^{-1}AL^{-T}).$$

Combining (A.5) and (A.6), we have

$$(A.7) \qquad \frac{\kappa(A)}{\kappa(LL^T)} \leq \kappa(L^{-1}AL^{-T}) \leq \kappa(LL^T)\, \kappa(A).$$

Now since L is strictly diagonally dominant, we can use Lemma A.2 to get a bound on $\kappa(LL^T)$. Using Lemma A.1(c) we get

$$\kappa(LL^T) \leq 4(2 + \sqrt{2})^{1/2} + 2\sqrt{2} + 6 \leq 17 .$$

Substituting in (A.7), the result follows.

Q.E.D.

Since $\kappa(A) = cn^2$, where c is a positive constant, it follows that the condition number of the ICCG(0) iteration matrix is bounded above and below by quantities of the form cn^2, where c is a positive constant.

APPENDIX B

Theorem B.1: For the model problem in three dimensions,

(B.1) for $\omega_1 = 2/[1 + \sin(\pi h)/2]$

$$\rho(S^{(\pi,\bullet)}_{\omega_1}) \sim 1 - \pi h, \quad \text{as } h \to 0;$$

(B.2) for $\omega_1 = 2/[1 + \{6(\frac{1-\cos(\pi h)}{1+\cos(\pi h)})\}^{1/2}]$

$$\rho(S^{(\pi_L)}_{\omega_1}) \sim 1 - 2^{-1/2}\, 3^{-1/2}\, \pi h, \quad \text{as } h \to 0;$$

and

(B.3) for $\omega_1 = 2/[1 + \{6(\frac{1-\cos(\pi h)}{3-2\cos(\pi h)})\}^{1/2}]$

$$\rho(S^{(\pi_P)}_{\omega_1}) \sim 1 - 3^{1/4}\pi h, \quad \text{as } h \to 0.$$

Proof: For each partition π, we prove that $\rho(L^{(\pi)} U^{(\pi)}) \leq 1/4$ and obtain $\rho(B^{(\pi)})$. (B.1) - (B.3) then follow from Theorem 6.5.

Note that

$$\begin{aligned}\rho(L^{(\pi)} U^{(\pi)}) &\leq \| L^{(\pi)} U^{(\pi)} \|_\infty \\ &\leq \| (D^{(\pi)})^{-1} C_L^{(\pi)} (D^{(\pi)})^{-1} C_U^{(\pi)} \|_\infty \\ &\leq \| (D^{(\pi)})^{-1} \|_\infty^2 \| C_L^{(\pi)} \|_\infty \| C_U^{(\pi)} \|_\infty\end{aligned}$$

Since $D^{(\pi)}$ is diagonally dominant, we can use a result due to Varah [59] to get

$$\| (D^{(\pi)})^{-1} \|_\infty < 1/[\min_k(|d_{kk}| - \sum_{\substack{j=1 \\ j\neq k}}^{N} |d_{kj}|)]$$

where d_{kj} are elements of $D^{(\pi)}$. Thus, we have

$$\|(D^{(\pi)})^{-1}\|_\infty < \begin{cases} 1/6 & \text{for } \pi = \pi_\theta, \\ 1/4 & \text{for } \pi = \pi_L, \\ 1/2 & \text{for } \pi = \pi_P. \end{cases}$$

Also

$$\|C_L^{(\pi)}\|_\infty = \|C_U^{(\pi)}\|_\infty = \begin{cases} 3 & \text{for } \pi = \pi_\theta, \\ 2 & \text{for } \pi = \pi_L, \\ 1 & \text{for } \pi = \pi_P. \end{cases}$$

Thus

(B.4) $\rho(L^{(\pi)} U^{(\pi)}) \leq 1/4$

for $\pi = \pi_\theta$, π_L, and π_P.

In [34] it is proved that

(B.5) $\rho(B^{(\pi_\theta)}) = \cos \pi h$.

We now obtain $\rho(B^{(\pi)})$ for $\pi = \pi_L$ and π_P.

Let v be an eigenvector of $B^{(\pi)}$ with components $v_{i,j,k}$, and let λ be the corresponding eigenvalue. Then, since

$$B^{(\pi)} = (D^{(\pi)})^{-1} (C_L^{(\pi)} + C_U^{(\pi)}),$$

we have

(B.6) $(C_L^{(\pi)} + C_U^{(\pi)})v = \lambda D^{(\pi)} v.$

We consider the following set of vectors which span R^N:

$$v_{i,j,k}^{(p,q,r)} = \gamma_{p,q,r} \sin(p\pi ih) \sin(q\pi jh) \sin(r\pi kh),$$

$$1 \leq p,q,r \leq n, \quad 1 \leq i,j,k \leq n,$$

where $h = 1/(n+1)$.

Then we have for $1 \leq p,q,r \leq n$ and $1 \leq i,j,k \leq n$,

$$(C_L^{(\pi)} + C_U^{(\pi)}) v_{i,j,k}^{(p,q,r)} = \begin{cases} 2[\cos(p\pi h) + \cos(q\pi h)] v_{i,j,k}^{(p,q,r)} & \text{for } \pi = \pi_L, \\ 2\cos(p\pi h) v_{i,j,k}^{(p,q,r)} & \text{for } \pi = \pi_P, \end{cases}$$

and

$$D^{(\pi)} v_{i,j,k}^{(p,q,r)} = \begin{cases} [6 - 2\cos(r\pi h)] v_{i,j,k}^{(p,q,r)} & \text{for } \pi = \pi_L, \\ [6 - 2\cos(q\pi h) - 2\cos(r\pi h)] v_{i,j,k}^{(p,q,r)} & \text{for } \pi = \pi_P. \end{cases}$$

Substituting in (B.6), we get that for all $1 \leq p,q,r \leq n$, $v^{(p,q,r)}$ are eigenvectors of $B^{(\pi)}$ with corresponding eigenvalues

$$\lambda_{p,q,r} = \begin{cases} \dfrac{\cos(p\pi h) + \cos(q\pi h)}{3 - \cos(r\pi h)} & \text{for } \pi = \pi_L, \\ \dfrac{\cos(p\pi h)}{3 - \cos(q\pi h) - \cos(r\pi h)} & \text{for } \pi = \pi_P. \end{cases}$$

Therefore,

(B.7) $\rho(B^{(\pi)}) = \lambda_{1,1,1} = \begin{cases} \dfrac{2\cos(\pi h)}{3 - 2\cos(\pi h)} & \text{for } \pi = \pi_L, \\ \dfrac{\cos(\pi h)}{3 - 2\cos(\pi h)} & \text{for } \pi = \pi_P. \end{cases}$

Applying Theorem 6.5 to (B.4), (B.5), and (B.7), the result follows.

Q.E.D.

REFERENCES

[1] O. Axelsson. A class of iterative methods for finite element equations. Computer Methods in Applied Mechanics and Engineering 9:123-137, 1976.

[2] O. Axelsson. A generalized SSOR method. BIT 13:443-467, 1972.

[3] R. E. Bank and D. J. Rose. Marching algorithms for elliptic boundary value problems I: The constant coefficient case. SIAM Journal on Numerical Analysis 14:792-829, 1977.

[4] R. Beauwens and L. Quenon. Existence criteria for partial matrix factorizations in iterative methods. SIAM Journal on Numerical Analysis 13:615-643, 1976.

[5] G. Birkhoff, R. S. Varga, and D. M. Young. Alternating direction implicit methods. Advances in Computers (Eds. F. L. Alt and M. Rubinoff) 3:189-273, 1962.

[6] A. Bracha-Barak and P. E. Saylor. A symmetric factorization procedure for the solution of elliptic boundary value problems. SIAM Journal on Numerical Analysis 10:190-206, 1973.

[7] J. R. Bunch. Partial pivoting strategies for symmetric matrices. SIAM Journal on Numerical Analysis 11:521-528, 1974.

[8] J. R. Bunch and L. Kaufman. Some stable methods for calculating inertia and solving symmetric linear systems. Report CU-CS-063-75, Department of Mathematics, University of California at San Diego, 1975.

[9] R. Chandra, S. C. Eisenstat, and M. H. Schultz. The modified conjugate residual method for partial differential equations. Proceedings of the Second IMACS(AICA) International Symposium on Computer Methods for Partial Differential Equations (Ed. R. Vichnevetsky), Lehigh University, Bethlehem, Pennsylvania, 1977, pp. 13-19.

[10] R. Chandra, S. C. Eisenstat, and M. H. Schultz. Conjugate gradient methods for partial differential equations. Proceedings of the AICA International Symposium on Computer Methods for Partial Differential Equations (Ed. R. Vichnevetsky), Lehigh University, Bethlehem, Pennsylvania, 1975, pp. 60-64.

[11] A. K. Cline. Several observations on the use of conjugate gradient methods. Department of Computer Science, University of Texas at Austin.

[12] P. Concus, G. H. Golub, and D. P. O'Leary. A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations. Proceedings of the Symposium on Sparse Matrix Computations, Argonne National Laboratory. Academic Press, 1975, pp. 309-332.

[13] P. Concus and G. H. Golub. A generalized conjugate gradient method for nonsymmetric systems of linear equations. Report STAN-CS-76-535, Department of Computer Science, Stanford University, 1976.

[14] E. H. Cuthill and R. S. Varga. A method of normalized block iteration. Journal of the Association of Computing Machinery 6:236-244, 1959.

[15] J. W. Daniel. The conjugate gradient method for linear and non-linear operator equations. SIAM Journal on Numerical Analysis 4:10-26, 1967.

[16] J. W. Daniel. The Approximate Minimization of Functionals. Prentice-Hall, 1971.

[17] F. W. Dorr. The direct solution of the discrete Poisson equation on a rectangle. SIAM Review 12:248-263, 1970.

[18] P. F. Dubois, A. Greenbaum and G. H. Rodrigue. Approximating the inverse of a matrix for use in iterative algorithms on vector processors. Report UCRL-80244, Lawrence Livermore Laboratory, 1977.

[19] T. Dupont. A factorization procedure for the solution of elliptic difference equations. SIAM Journal on Numerical Analysis 6:753-782,1968.

[20] T. Dupont, R. P. Kendall, and H. H. Rachford. An approximate factorization procedure for solving self-adjoint elliptic difference equations. SIAM Journal on Numerical Analysis 5:559-573,1968.

[21] L. W. Ehrlich. The Block Symmetric Successive Overrelaxation Method. PhD dissertation, University of Texas at Austin, 1963.

[22] L. W. Ehrlich. The block symmetric successive overrelaxation method. Journal of the Society for Industrial and Applied Mathematics 12:807-826, 1964.

[23] S. C. Eisenstat. Private communication.

[24] S. C. Eisenstat. Complexity bounds for Gaussian elimination. To appear.

[25] S. C. Eisenstat, J. W. Lewis, and M. H. Schultz. To appear.

[26] S. C. Eisenstat, M. H. Schultz and A. H. Sherman. Application of sparse matrix methods to partial differential equations. Proceedings of the AICA International Symposium on Computer Methods for Partial Differential Equations, Bethlehem, Pennsylvania, 1975, pp. 60-64.

[27] H. Engeli, T. Ginsburg, H. Ruthishauser, and E. Stiefel. Refined iterative methods for computation of the solution and the eigenvalues of self-adjoint boundary value problems. Mitteilungen aus dem Institut fur Angewandte Mathematik :8, Birkhauser Verlag, Basel, Stuttgart, 1959.

[28] D. J. Evans. The analysis and application of sparse matrix algorithms in the finite element method. The Mathematics of Finite Elements and Applications. Proceedings of the Brunel University Conference (Ed. Whiteman), Academic Press, London, 1973.

[29] D. K. Faddeeva and V. N. Faddeeva. Computational Methods of Linear Algebra. Freeman and Company, London, 1963.

[30] R. Fletcher. Conjugate gradient methods for indefinite systems. Proceedings of the Dundee Biennial Conference on Numerical Analysis (Ed. G. A. Watson). Springer Verlag, 1975.

[31] I. Fried and J. A. Metzler. The conjugate gradient method with finite elements. Proceedings of the Second IMACS(AICA) International Symposium on Computer Methods for Partial Differential Equations (Ed. R. Vichnevetsky), Lehigh University, Bethlehem, Pennsylvania, 1977, pp. 165-169.

[32] G. E. Forsythe and C. B. Moler. Computer Solution of Linear Algebraic Systems. Prentice-Hall, 1967.

[33] G. E. Forsythe and E. G. Strauss. On best-conditioned matrices. Proceedings of the American Mathematical Society 6:340-345, 1955.

[34] G. E. Forsythe and W. R. Wasow. Finite Difference Methods for Partial Differential Equations. John Wiley and Sons, 1960.

[35] J. A. George. Nested dissection of a regular finite element mesh. SIAM Journal on Numerical Analysis 10:345-363, 1973.

[36] F. G. Gustavson. Basic techniques for solving sparse linear systems. Sparse Matrices and Their Applications (Eds. Rose and Willoughby). Plenum Press, New York, 1972.

[37] G. J. Habetler and E. L. Wachspress. Symmetric successive overrelaxation in solving diffusion difference equations. Mathematics of Computation 15:356-362, 1961.

[38] M. R. Hestenes. The conjugate gradient method for solving linear systems. Proceedings of the Symposia in Applied Mathematics 6:83-102, Numerical Analysis. McGraw-Hill, 1956.

[39] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. Journal of Research of the National Bureau of Standards 49:409-436, 1952.

[40] A. S. Householder. The Theory of Matrices in Numerical Analysis. Blaisdell Publishing Company, 1964.

[41] D. S. Kershaw. The incomplete Cholesky conjugate gradient method for the iterative solution of systems of linear equations. Report UCID-30083, Lawrence Livermore Laboratory, 1976.

[42] C. N. Kostem and E. G. Schultchen. Solution of linear equations by iterative methods in finite element analysis. Fritz Engineering Laboratory Report 400.10, Lehigh University, Bethlehem, Pennsylvania, 1973.

[43] C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. Journal of Research of the National Bureau of Standards 45:255-282, 1950.

[44] V. I. Lebedev. Iterative methods for the solution of operator equations with their spectrum lying on several intervals. Journal of Computational Mathematics and Mathematical Physics 9(6):1247-1252, 1969.

[45] D. G. Luenberger. The conjugate residual method for constrained minimization problems. SIAM Journal on Numerical Analysis 7:390-398, 1970.

[46] D. G. Luenberger. Hyperbolic pairs in the method of conjugate gradients. SIAM Journal on Applied Mathematics 17:1263-1267, 1969.

[47] T. A. Manteuffel. An Iterative Method for Solving Linear Systems with Dynamic Estimation of Parameters. PhD dissertation, University of Illinois at Urbana-Champaign, Urbana, Illinois, 1975.

[48] J. A. Meijerink and H. A. van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. Mathematics of Computation 31:148-162, 1977.

[49] D. P. O'Leary. Hybrid Conjugate Gradient Algorithms. PhD dissertation, Department of Computer Science, Stanford University, 1976.

[50] C. C. Paige. Computational variants of the Lanczos method for the eigenproblem. Journal of the Institute of Mathematics and its Applications 10:373-381, 1972.

[51] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of equations and least squares problems. Report CS-399, Department of Computer Science, Stanford University, 1973.

[52] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. SIAM Journal on Numerical Analysis 12:617-629, 1975.

[53] J. K. Reid. On the method of conjugate gradients for the solution of large sparse systems of linear equations. Proceedings of the Conference on Large Sparse Sets of Linear Equations(Ed. J. K. Reid). Academic Press, 1971, pp. 231-254.

[54] J. K. Reid. The use of conjugate gradients for systems of linear equations possessing "Property A". SIAM Journal on Numerical Analysis 9:325-332, 1972.

[55] G. W. Stewart. The convergence of the method of conjugate gradients at isolated extreme points of the the spectrum. Numerische Mathematik 24:85-93, 1975.

[56] E. Stiefel. Relaxationmethoden bester strategie zur losung linearer gleichungssystems. Comm. Math. Helv. 29:157-179, 1955.

[57] H. L. Stone. Iterative solution of implicit approximation of multidimensional partial differential equations. SIAM Journal on Numerical Analysis 5:530-558, 1968.

[58] A. van der Sluis. Condition numbers and equilibration of matrices. Numerische Mathematik 14:14-23, 1969.

[59] J. M. Varah. A lower bound for the smallest singular value of a matrix. Linear Algebra Appl. II:3-5, 1975.

[60] R. S. Varga. Matrix Iterative Analysis. Prentice-Hall, 1962.

[61] E. L. Wachspress. Extended application of alternating direction implicit iteration model problem theory. Journal of the Society for Industrial and Applied Mathematics 11:994-1016, 1963.

[62] D. M. Young. Iterative Solution of Large Linear Systems. Academic Press, 1971.