1982

Parallel algorithms for the iterative solution of large sparse linear systems

Krettmann, Jurgen

PARALLEL ALGORITHMS FOR THE ITERATIVE SOLUTION OF LARGE SPARSE LINEAR SYSTEMS

A thesis submitted to
Lakehead University
in partial fulfillment of the requirements
for the degree of
Master of Science

by

© Jürgen Krettmann

1982

ACKNOWLEDGMENTS

I wish to thank my supervisor, Professor M.W. Benson, for his advice and encouragement during the preparation of this thesis.

The use of approximate inverse for preconditioning the conjugate gradient method was suggested by Dr. P. O. Frederickson, Los Alamos National Laboratory, Los Alamos, New Mexico.

Finally, I would like to thank Ahmoi for her support during the preparation of this thesis.

Abstract

In this thesis we are concerned with iterative parallel algorithms for solving finite difference equations arising from boundary value problems. We propose various new methods based on approximate inverses. The Jacobi iterative method for the solution of linear equations is highly parallel. Its slow convergence rate, however, has initiated the study of a variety of acceleration techniques. We introduce the "Diagonal-Block" and the "Least Squares" techniques for calculating approximate inverses which lead naturally to generalized Jacobi methods that are well suited to parallel processing. In addition, the calculations required to establish the iterative process can be done in parallel. A convergence theorem for a one-dimensional model problem is given. Gauss-Seidel versions of these methods are also considered. These versions converge faster but often at the cost of decreased parallelism.

Preconditioning has been successfully used in conjunction with the conjugate gradient method. Many such techniques use an approximation M to the matrix A of the linear system under consideration and solve a system of the form $M z = d$ at each iteration. If, however, an approximate inverse M^{-1} is used, the solution $z = M^{-1} d$ only involves a matrix-vector multiplication which is well suited to parallel computation. We examine the

Diagonal-Block and the Least-Squares techniques for preconditioning the conjugate gradient method.

Numerical results are presented. The proposed Jacobi like and Gauss-Seidel like methods are compared with parallel Gauss-Seidel methods in a class of parallel algorithms proposed by Evans and Barlow (1982). The preconditioned conjugate gradient methods are compared with the plain conjugate gradient method. In all cases the speed of convergence increased substantially, but often at the expense of increased computational work. In some cases it was necessary to assume the number of processors to be on the order of N (the number of unknowns) in order to gain an advantage from parallel processing.

LIST OF ITERATIVE METHODS

Abbreviation	Explanation
PGS1	A Parallel Gauss-Seidel Version for the one dimensional model problem.
PGS2	A Parallel Gauss-Seidel Version for the two dimensional model problem.
JDB1(I)	Jacobi Diagonal-Block methods for the one dimensional model problem.
JDB2(I)	Jacobi Diagonal-Block methods for the two dimensional model problem.
GDB1(I)	Gauss-Seidel Diagonal-Block methods for the one dimensional model problem.
GDB2(I)	Gauss-Seidel Diagonal-Block methods for the two dimensional model problem.
JLSQ1(I)	Jacobi Least-Squares methods for the one dimensional model problem.
JLSQ2(I)	Jacobi Least-Squares methods for the two dimensional model problem.
GLSQ1(I)	Gauss-Seidel Least-Squares methods for the one dimensional model problem.
GLSQ2(I)	Gausss-Seidel Least-Squares methods
CG	Conjugate Gradient Method
DBCG1(I)	Diagonal-Block Conjugate Gradient methods for the one dimensional model problem.
DBCG2(I)	Diagonal-Block Conjugate Gradient methods for the two dimensional model problem.
LSQCG1(I)	Least-Squares Conjugate Gradient methods for the one dimensional model problem.
LSQCG2(I)	Least-Squares Conjugate Gradient methods for the two dimensional model problem.

CONTENTS

I. INTRODUCTION

In this thesis, we consider linear systems arising in connection with the numerical solution of boundary value problems. These linear systems are large and sparse. We propose several iterative algorithms that are based on approximating the inverse of the coefficient matrix of the linear system. Our main objective is to investigate the use of these approximate inverses for linear stationary methods (see chapter 3) and preconditioned conjugate gradient methods (see chapter 4) as a means to enhance the rate of convergence.

The iterative methods in this thesis, however, can be extremely costly in a computational sense (especially for large linear systems). This suggests the concurrent use of many processing elements in order to improve the speed of computation and to handle large volumes of data. Massively parallel computers, as assumed in this thesis, are not yet available, but the development of VLSI circuits has made them feasible (Siegel, 1982, Haynes, 1982). Moreover, it is an open problem how expensive the interconnection between processors will be and what limitations that will pose on the speed of computation. The interconnection requirements will not be considered in our model of computation. The methods under consideration, however, involve mainly matrix-vector products which can be computed very

efficiently on a linear-connected systolic array as considered by Kung and Leierson (see Haynes, et al.1982). No control of the data items is required once it is input into the array. Systolic arrays can be thought of as part of a larger computer system replacing certain software routines (Haynes, et al.1982).

In chapter 2 we discuss certain aspects of parallel computation and establish a general model of a parallel computer which sets the background for comparing our iterative processes. In section II.3 we develop two parallel versions of the Gauss-Seidel iterative methods, PGS1 and PGS2, that exploit the sparsity structure of the matrices under consideration. These methods serve as a basis of comparison for the methods using approximate inverses that are developed in the chapters that follow.

We will use two examples to illustrate our results (see appendix A). We consider Young's two dimensional model problem (Young,1971,pp. 2-4) and its one dimensional analog. The one dimensional model problem provides a relatively simple test problem for studying our proposed algorithms.

In chapter 3, we deal with linear stationary methods of the form $u_{k+1} = (I - BA) u_k + Bb$ to solve the linear system $Au = b$. In sections III.1 and III.2 we develop some standard convergence results. According to these results we attempt in section III.3 to create approximate inverses B such that the spectral radius of $I-BA$ is as

small as possible. The approximate inverse B is required to satisfy a certain sparsity structure and the non-zero entries are determined in a way that BA approximates the identity. Two different techniques are developed, the Diagonal-Block (DB) and the Least-Squares (LSQ) technique (Benson,Frederickson,1981,Benson,1973). The corresponding approximate inverses are used in Jacobi like and Gauss-Seidel like iterations. In case of the Diagonal-Block (DB) approximate inverses, we give a convergence result for the Jacobi like methods applied to the one-dimensional model problem. All experiments have been carried out on sparse matrices. We remark that these techniques for approximating the inverse are not necessarily restricted to such matrices. The Least-Squares technique, however, is highly expensive if the coefficient matrix is dense.

In chapter 4, we develop a generalized conjugate gradient method for solving linear systems and discuss its main properties. This method can be used in conjunction with an approximation M to the coefficient matrix A. In this case a linear system of the form $Mz = d$ must be solved at each iteration. Applying the DB or LSQ approximate inverses to this method, we can compute the vector z by matrix-vector multiplication.

In chapter 5, we present the results of our numerical experiments on the two model problems considered. Two sets of algorithms are compared with respect to two measures, the speed of convergence and the amount of computational work

per iteration. The latter varies with the number of independent processors that are assumed. The linear stationary methods of chapter 3 are compared with the Gauss-Seidel methods PGS1 and PGS2 of chapter 2 and the preconditioned conjugate gradient methods are compared with the plain conjugate gradient method. It was found that the number of iterations is reduced significantly when approximate inverses are used. Moreover, the "better" the inverse of the coefficient matrix is approximated, the fewer iterations are needed in most cases. If we include the second measure of comparison, this result cannot be obtained in all cases. We observe that in many cases the linear stationary methods using the proposed approximate inverses are shown to be advantageous over the parallel Gauss-Seidel methods of chapter 2, if an appropriate number of processors is assumed. The same result applies to the preconditioned conjugate gradient methods and the conjugate gradient method. Finally we comment that all parallel algorithms in this thesis are simulated on a uniprocessor.

II. CONCEPTS OF PARALLEL COMPUTATION

The advances in parallel computer systems during the last decade have brought a new aspect to the classification of numerical algorithms: sequential vs. parallel. The concept behind parallel computing is that programs are designed to have paths of independent calculations in order to make use of many processors at a time. For optimal efficiency, programs using P processors should run P times faster than otherwise identical programs using only one processor. However, "experience and theory show that the actual speed-up is often much less" (Heller,1978).

The first section of this chapter is devoted to a general discussion of parallel computers. In the second section, we develop our model of parallel computation and outline some of the problems occurring in this field. In the third section, we present several parallel algorithms for solving linear systems arising from differential equations.

II.1 CLASSIFICATION OF PARALLEL COMPUTERS

Thus far, no single parallel computer system can be considered as the dominant one. Our interest here is in the following two important classes of parallel computers, first defined by Flynn, 1966.

Instructions for each processor in a multi - processor system come either from a central control unit or from the individual processor. In the first case, the system has only one stream of instructions in execution at a given time, but each processor may affect many different data. This is the single - instruction stream multiple - data stream model (S I M D). SIMD machines are best suited for algorithms requiring the same operation on large arrays of independent data. Examples of this type are array processors, pipeline processors, associate processors and bit - slice processors (Stone,1973). The more general multiple - instruction stream multiple - data stream model (M I M D) is configured as multiple independent processing elements with no processor having overall control. MIMD machines are capable of executing different instructions simultaneously, and each instruction may operate on a different datum. For complex programs, the processors are initially allocated to independent (parallel) paths, which they execute until completion. Communication among the individual processors still takes place in order to share information and to otherwise cooperate in the solution of

the problem (synchronization may take place).

The following discussion will emphasize SIMD machines, since the iterative methods of the chapters that follow are ideally suited for vector processes. Moreover, most of the existing machines are of this type.

The importance of the above characterization of parallel computer systems lies in the fact that an algorithm designed for an SIMD machine can be used with essentially equal efficiency by all systems of that class. For example, both an ILLIAC IV and a pipeline computer such as CDC STAR 100 are heavily oriented to vector processes, so that a good algorithm for one of them will tend to be a good one for the other one as well. Nevertheless, to design an algorithm that runs at maximum efficiency on a particular computer, one may have to take the architecture of that computer into account, since SIMD computers differ in specific capabilities. However, it is worthwhile to investigate algorithms for SIMD computers because the principle design of an algorithm is usually not affected by the changes required for optimal efficiency. The same can be said about algorithms for MIMD computers.(Stone,1973)

II.2 MODEL OF PARALLEL COMPUTATION

There are various models of parallel computation. The theoretical model assumes unlimited parallelism in the sense that the number of required processors varies with the size of the problem under consideration (" sufficiently many processors ",Heller,1978). Our interest is in the iterative solution of a linear system of equations with a banded (or striped) and large matrix, say of order 10^3 to 10^6. Therefore we prefer the practical model that has a fixed number of processors, independent of the application. Nevertheless, we will also for simplicity refer to the theoretical model, which might be justified by the current technology that has created the " age of the microprocessor " thus making the future construction of computer systems with 2^{10} to 2^{14} processors feasible (Siegel,1982,see also Haynes,1982).

The precise details of the architectures are unimportant for our purposes. When we speak of a SIMD machine, we have in mind a computer system consisting of a control unit, P processing elements (PE's) and an interconnection network. Each PE consists of a processor with its own private memory for intermediate storage, and each PE also has access to a common memory. The PE's are connected by a network, and fed instructions by the control unit. To simplify discussions we make the strong assumption that any processor can obtain any data in one time step,

that is to say we will ignore the time required for data acquisition and concentrate on the arithmetic time. We remark that we can convert this computer model into an MIMD model by storing predetermined instructions in the private memory of each PE.

The above assumption that each processor can fetch any data item in one time step (cycle) is somewhat ideal. In reality, constraints on the computer architecture can create complex problems of data manipulation, which may result in a significant loss of processing power. The systolic architectural concept (see Kung,1982) optimizes the data-communication structures among the processors at the price of specialization. This concept results in extremely high speed and efficient special-purpose systems. The basic idea is an overlapping of I/O and computation. Each data item is operated on many times and no further control is required after the data item is input into the array of processors. A simple example is the linearly connected array considered by Kung and Leiserson (see Haynes,et al.,1982,p.10).This systolic array can compute a matrix-vector product of size N with a banded matrix of bandwidth W in 2N+W time units when W/2 processors are used. This is fairly close to the 2N time units that are required when the above idealized computer model is assumed. Since systolic arrays are very specialized systems of processors it is desirable that they be part of a larger computing system. At this point, however, the integration

of these special purpose devices into a larger computing system is an open problem.

In order to evaluate the algorithms and the computer systems under consideration, it is necessary to have some measure of efficiency. We mentioned earlier that for a parallel computer system that can support P simultaneous processes, the ideal speed-up is P . This leads to the following definitions (Schendel,1981).

Definition 2.1 :

For a given problem let T(1) be the running time of the best known (fastest) sequential algorithm, and let T(P) be the running time of a parallel algorithm using P processors and solving the same problem. Then the speed-up ratio is defined as

(2.1) $S(P) = T(1)/T(P)$.

The speed-up ratio measures the improvement in execution time using parallelism, but it does not take into account how well each processor is being used. We may have severe unemployment among the processors. Therefore we define efficiency as

(2.2) $E(P) = S(P) / P$.

To compare parallel algorithms solving the same problem, we define the effectiveness of a parallel algorithm as

(2.3) $F(P) = S(P) / (P * T(P))$

Note that $C(P) := P * T(P)$ measures the cost of the algorithm. Also $F(P) = S(P)/(P*T(P)) = E(P)/T(P) = E(P) * S(P)/T(1)$,i.e. $F(P) \leq 1$, can be seen as a measure for the speed-up and the efficiency. Therefore we tend to maximize the function $F(P)$ in order to achieve a good parallel algorithm. It is not difficult to show that $S(P) \leq P$ and $E(P) \leq 1$. Although one of our objectives is to obtain optimal efficiency, it is not practical to choose the number of processors in order to maximize the function $E(P)$ (note that $E(1) = 1$). (Heller,1978)

The ideal speed-up ratio is linear in P, since the processors are used efficiently in that case. Such speed-up ratios can be attained in problems that have iterative structure, such as systems of linear equations and other vector-matrix problems. A number of factors such as synchronization, overhead, or input/output (I/O) can cause a smaller speed-up ratio. Speed-up ratios of the form $k * P / \log(P)$ are still acceptable, but algorithms with a speed-up ratio of $k * \log(P)$ are not suited very well to parallel computation. (Stone,1973)

In the remainder of this section we describe briefly some of the problems that occur in parallel computation. Parallel computation cannot be considered as a trivial extension of serial computation, in the sense that efficient serial algorithms cannot necessarily be automatically transformed into efficient parallel algorithms. In fact,

inefficient serial algorithms may even lead to efficient parallel algorithms and vice versa. Batcher (1968), for example, has formulated a parallel sorting algorithm that stems from an inefficient serial sorting algorithm (see Stone,1973). Moreover, memory access and the interprocessor communication is crucial for successful exploitation of parallelism (Haynes,et al.,1982).

We mentioned earlier that the problem of communication among the processors is crucial to successful exploitation of parallelism. Therefore the data in parallel computers must be arranged in memory for efficient parallel computation. One example is the storage of a matrix in an unconventional way known as skewed storage (see Schendel,1981,pp.19-24), which allows for fetching rows and columns with equal ease. An extension of skewed memory storage is given by Budnik and Kuck,1971, who explore variations of the ILLIAC IV architecture which allow for fetching rows, columns, diagonals and certain submatrices with equal efficiency.

The architectural feature of the CDC-STAR, however, requires vector operants to be contiguous blocks of memory locations in order to simplify memory transfers (Heller,1978). Consequently, the above storage technique does not apply for the STAR. But the STAR has features that allow for transposing matrices very efficiently (Stone,1973).

Finally we observe that parallel programs may behave quite differently numerically than their serial counterpart. In the next section we describe an iterative algorithm where the parallel version converges more slowly than the serial version. However, the convergence rate of the parallel algorithm can be improved by simple modifications.

II.3 PARALLEL LINEAR SYSTEM SOLVERS

In this section we are concerned with a class of parallel methods proposed by Barlow and Evans, 1982 (see also Stone,1973). The methods are parallel forms of the Gauss-Seidel method.

For example, consider the two dimensional model problem of appendix A , that is the discretized version of Laplace's equation on the interior of a square with zero boundary conditions. First, we consider the so-called natural ordering of the mesh-points, that is the rows of the mesh are numbered from bottom to top and the columns are numbered from left to right. Denoting the solution at an arbitrary mesh-point in the interior of the square by MP and the solution at its four neighbours by N,E,W and S , the Jacobi iterative method is

$$(2.4) \qquad MP(i) = [\ N(i-1) + E(i-1) + W(i-1) + S(i-1)\ ] / 4,$$

where MP(i) denotes the value of MP at the i'th iteration. From formula (2.4) one sees that all new values

at the mesh-points are calculated from the old values at the neighbouring mesh-points. Thus, all mesh-points can be updated in parallel. The convergence rate of the Jacobi method can be improved by using new data as soon as it becomes available, but at a cost of reduced parallelism. Thus, we get the Gauss-Seidel method :

(2.5) $MP(i) = [\ N(i-1) + E(i-1) + W(i) + S(i)\ ] / 4$.

This method converges much faster than the Jacobi method, since it makes use of the newer data which are more accurate than the older data. In the case where the coefficient matrix A has property A [see Young,1971,pp. 41-42] the Gauss-Seidel method converges twice as fast as the Jacobi method.

One simple parallel implementation of the above method is to update the points on one entire row of the mesh at a time, starting at the bottom. In this case the iteration formula is given by

(2.6) $MP(i) = [\ N(i-1) + E(i-1) + W(i-1) + S(i)\ ] / 4.$

Since only the point S contributes new data in this formula, we can expect that this method requires roughly 50% more iterations than the Gauss-Seidel method.

Scanning the mesh-points by diagonals as shown in

Fig. (2.8), we obtain a parallel algorithm that has the Gauss - Seidel convergence rate (Stone,1973).

43	2	10	18	26	34	42
36	44	3	11	19	27	35
29	37	45	4	12	20	28
22	30	38	46	5	13	21
15	23	31	39	47	6	14
8	16	24	32	40	48	7
1	9	17	25	33	41	49

Fig. (2.8)

The iteration formula for this method is equation (2.5), so that the rate of convergence is the same as the serial Gauss-Seidel algorithm. In fact, by theorem 3.4 (Young,1971,p.147) any consistent reordering does not change the eigenvalues of the iteration matrix. In particular, the matrix $\tilde{A}$ of the linear system corresponding to a permutation σ of the mesh-points is a similarity transformation of the matrix A associated with the natural ordering :

$$\tilde{A} = P^{-1} A P ,$$

where the permutation matrix P consists of the columns of the identity matrix permuted according to σ . Thus we can see that

$$\text{diag } \tilde{A} = P^{-1} (\text{diag } A) P ,$$

where diag A denotes the diagonal matrix associated with A .

Now,

$$\begin{aligned}
\tilde{G} &= I - (\mathrm{diag}\ \tilde{A})^{-1}\ \tilde{A} \\
&= P^{-1}P - P^{-1}P\ (\mathrm{diag}\ \tilde{A})^{-1}\ P^{-1}\ A\ P \\
&= P^{-1}\ (\ I - P\ (\mathrm{diag}\ A)^{-1}\ P^{-1}\ A\ )\ \ P \\
&= P^{-1}\ (\ I - (\mathrm{diag}\ A)^{-1}\ A\ )\ \ P \\
&= P^{-1}\ G\ \ P\ ,
\end{aligned}$$

where G and $\tilde{G}$ are the iteration matrices of the Jacobi method for the systems $Au = b$ and $\tilde{A}\ u = b$ respectively. Hence G and $\tilde{G}$ have the same eigenvalues and therefore - by Young's result (see above) - the eigenvalues of the Gauss-Seidel iteration matrix are the same for the two systems.

Let T(1) = a * (N**2) be the number of time units required for the sequential Gauss-Seidel algorithm, where "a" is the number of required iterations. Then the time units used by the line-parallel Gauss-Seidel (2.6) and the diagonal-parallel Gauss-Seidel method (see (2.5) and fig.(2.8)) are T(N) = 3/2 a N and T(N) = a N respectively, so that we have a speed-up of 2/3 N for the line-parallel Gauss-Seidel and a speed-up of N for the diagonal-parallel Gauss-Seidel method assuming that N processors are available.

The above idea of constructing parallel Gauss - Seidel methods has been generalized by Barlow and Evans (1982). A matrix A having property A can be rearranged so that

$$A = \begin{bmatrix} D_1 & H \\ K & D_2 \end{bmatrix} ,$$

where D_i ,i = 1,2 are diagonal matrices. This means that the indices of the equations of the linear system can be divided into two disjoint subsets so that the update equation corresponding to one subset involves only the components corresponding to the other set. Thus all the updates within one subset can be done in parallel.

Applying this idea to a tridiagonal matrix we get the following method

$$u(j,i+1) = [\ a(j) * u(j-1,i) + c(j) * u(j+1,i)\ ] / b(j) ,$$

for j odd, and

(2.9)

$$u(j,i+1) = [\ a(j) * u(j-1,i+1) + c(j) * u(j+1,i+1)\ ] / b(j) ,$$

for j even ,

where i is the iteration index , j is the index for the j'th component of the vector u and the vectors a , b and c represent the subdiagonal, the diagonal and the superdiagonal of the tridiagonal matrix . The above updating scheme uses "old" data for updating the "even" components and "new" data for updating the "odd" components.

Thus, in the mean, over a sequence of iterations, this method makes use of one "new" datum for updating one component of u. For this method we get the same convergence rate as for the sequential Gauss-Seidel method, since this method corresponds to a consistent reordering of the matrix A of the linear system (the same argument as above for the method using a reordering as shown in Fig (2.8)). Two steps are needed in order to update all components of u at each iteration if $N/2$ processors are available ($\lceil x \rceil$ is the integer defined by $x \leq \lceil x \rceil < x+1$). Since the convergence rate is the same as for the Gauss-Seidel method we get a speed-up ratio of $N/2$ (see equ. (2.1)). In the chapters that follow we refer to this method as the PGS1 method. This method will be used in the numerical section for the purpose of comparison.

The same idea applies to the solution of the linear system arising from the discretization of Laplace's equation in the interior of a square (second model problem) . We scan the mesh of unknowns by diagonals. Now the updating scheme consists of updating every second diagonal starting in the lower left corner with "old" data and then updating the rest of the mesh-points with "new" data . Note that during the second half of an iteration all four neighbours have already been updated during the first half of the iteration. In the mean (over several iterations) each component of u is updated with two "new" (updated) neighbours and two "old" neighbours. Thus the convergence

rate is again the same as for the sequential Gauss-Seidel method, since this method corresponds to a consistent reordering of the matrix A of the linear system (the same argument as above). Therefore we can achieve a speed-up of (N**2)/2 which is an improvement over the diagonal parallel Gauss-Seidel method (the method based on the ordering of Fig.(2.8)). On the other hand, the number of required processors to achieve the above speed-up increases like O(N**2). We denote this method for future reference by PGS2 . The PGS2 method will serve as a basis of comparison in chapter 5.

Parallel versions of the standard successive overrelaxation (SOR) method can be constructed in a manner similar to the parallel Gauss-Seidel methods.

III. LINEAR STATIONARY METHODS

Our concern in this section is solely with linear stationary methods of first degree, which have the form (Young, 1971)

$$(3.1) \qquad u_{k+1} = G\, u_k + L \quad , \; k \in N \; ,$$

where G is an NxN matrix and L some N vector. The iterative method (3.1) is used to solve a linear system $Au = b$, where we assume throughout that A is a real nonsingular matrix. Thus the solution of the linear system exists and is unique, and the solution vector is given explicitly by

$$(3.2) \qquad u = A^{-1} b$$

In this chapter, we give some basic definitions and develop some standard results which set a background for the work to follow. We also propose a generalization of the well-known Jacobi method based on the idea of approximate inverses. These methods are ideally suited for parallel processing. The approximate inverses can also be used in a Gauss-Seidel-like implementation of (3.1), where each component is updated sequentially with the most recent data available. The parallelism is thereby limited to the number of non-zero entries per row of the iteration matrix G . In some cases, however, techniques of chapter two can be used to increase parallelism (see section III.4).

III.1 GENERAL CONVERGENCE THEOREM

In this section we determine under what conditions the sequence $\{ u_k \}_{k \in \mathbb{N}}$ defined by (3.1) converges for any starting vector to the unique solution of the system under consideration. The following results are developed in Young (1971) in a more general context than that required for our purposes. First we give the following

DEFINITION 3.1

The spectral radius of a matrix A is defined as

$$S(A) = \max \{ |\lambda| : \lambda \text{ is an eigenvalue of } A \}.$$

Before we can establish our main result of this section, we need

LEMMA 3.2 [Young, 1971, p.32]

For any matrix norm $\|\cdot\|$

we have

$$S(A) \leq \|A\|$$

Proof:

Defining the compatible vector norm $\|\cdot\|_*$ by

$$\|v\|_* = \|B\| ,$$

where B is the matrix whose first column is v and all other elements vanish, we have

$$\|A\, v\|_* \leq \|A\| \, \|v\|_* ,$$

for all A and v. For any eigenvalue λ of A and an associated eigenvector v we have $A\, v = \lambda\, v$, and hence $\|A\, v\|_* = |\lambda| \, \|v\|_* \leq \|A\| \, \|v\|_*$, i.e. $|\lambda| \leq \|A\|$. ∎

We also require

LEMMA 3.3 [Young, 1971, p. 36]

The following conditions are equivalent

i) $\lim_{n \to \infty} \|A^n v\| = 0$,

for all $v \in R^n$ and any vector norm.

ii) $S(A) < 1$

Proof:

(ii => i)

Suppose $S(A) < 1$. By Theorem 3.5 (Young, 1971, p.33) there exists for any given $\varepsilon > 0$ a nonsingular matrix M such that

(*) $\|A\|_M \quad S(A) + \varepsilon$

where $\|A\|_M = \| M^{-1} A M \|_2$ and $\|A\|_2 = S(A^T A)^{1/2}$. Hence, for ε sufficiently small, we have

$\|A\|_M < 1.$

Since $\|A^n\|_M \leq \|A\|_M^n$ it follows that $\|A^n\|_M \to 0$. Thus

(**) $\lim_{n \to \infty} \|A^n v\|_x \leq \lim_{n \to \infty} \|A\|_M^n \|v\|_x = 0,$

where $\|\cdot\|_x$ is the vector norm defined as in the proof of Lemma 3.2 for the case of the matrix norm $\|\cdot\|_M = \|\cdot\|$. But (**) implies i) , which completes the first half of the proof.

(i => ii)

Suppose $\|A^n v\| \to 0$ and $\|A^n\|$ does not converge to zero for any vector norm $\|\cdot\|$. Now it is impossible that $S(A) < 1$, since by (*) we could find a matrix norm such that $\|A\| < 1$. Thus we can find an eigenvalue λ of A

with $|\lambda| \geq 1$ and a vector $v \neq 0$ such that $Av = \lambda v$ holds. Hence $\|A^n v\| = |\lambda|^n \|v\|$ which does not converge to zero. Therefore condition i) implies that $\|A^n\|$ converges to zero for any matrix norm.

Now suppose $S(A) \geq 1$. Then, by Lemma 2.2 $1 \leq S(A) \leq \|A\|$ for any matrix norm. Thus we have $1 \leq S(A^n) \leq \|A^n\|$ for all n. Hence ii) is a necessary condition for i) which completes the proof. ∎

Now we can state our main result of this section.

GENERAL CONVERGENCE THEOREM 3.4 [Young,1971,p.77]
The iterative method

$$u_{k+1} = Gu_k + L$$

is convergent independent of the starting vector u_0 if and only if

$$S(G) < 1 .$$

Moveover, it converges to the unique solution of the linear system $Au = b$ if and only if

$$L = (I-G)\, A^{-1} b$$

and

$$S(G) < 1$$

holds.

Proof

If $\lim_{k\to\infty} u_k = \hat{u}$ exists independent of the starting vector u then

$$\hat{u} = \lim_{k\to\infty} u_{k+1} = \lim_{k\to\infty} (G u_k + L)$$

$$= G (\lim_{k\to\infty} u_k) + L$$

$$= G \hat{u} + L ,$$

i.e. (*) $(I-G)\hat{u} = L$.

Defining $\varepsilon_k = u_k - \hat{u}$ we get from (*), $u_{k+1} = G u_k + (I-G)\hat{u}$. Hence $\varepsilon_{k+1} = G\varepsilon_k$ and

(**) $\varepsilon_k = G^k \varepsilon_0$.

Now, by Lemma 3.3, $\lim_{k\to\infty} \|\varepsilon_k\| = 0$ for all ε_0 if and only if $S(G) < 1$ holds.

If $S(G) < 1$, then $\det(I-G) = \prod_{j=1}^{N} (1-\lambda_j) \neq 0$ (where the λ_j are the eigenvalues of G) and the equation $(I-G)u=L$ has a unique solution, say $\hat{u}$. By Lemma 3.3 we have $\|\varepsilon_k\| \to 0$, as $k \to \infty$, which completes the first part of the proof.

Since $I-G$ is nonsingular when $S(A) < 1$, we get from (*) $\hat{u} = (I-G)^{-1} L = A^{-1} b$. Conversely, if the sequence $\{u_k\}_{k\in\mathbb{N}}$ converges to $z = A^{-1}b$ then $L = (I-G) A^{-1} b$, which completes the proof. ∎

Finally, we remark that from Lemma 3.2 and Theorem 3.4 we can see that $\|G\| < 1$ for some matrix norm is sufficient for convergence of the iterative method (3.1).

III.2 RATES OF CONVERGENCE

It is important to understand something about the rate of convergence of different algorithms, since to a certain extent the rate of convergence of a method is as important as the fact that it converges; if it converges slowly we may never be able to see it converge. Therefore, in this section, we shall outline certain results which give insight into rates of convergence.

A reasonable algorithm should at least be linearly convergent in the sense that if the sequence $\{u_k\}$ is generated by the algorithm and converges to u^*, then for some norm $\|\cdot\|$ there is a $c \in (0,1)$ and $k_0 \geq 0$ such that

$$(3.3) \qquad \|e_{k+1}\| \leq c \cdot \|e_k\| \ , \ k \geq k_0 ,$$

where $e_k = u_k - u^*$. This guarantees that eventually the error will be decreased by the factor $c < 1$ on each iteration.

We observe that the method (3.1) is linearly convergent if $\|G\| < 1$ for some norm ($c = \|G\|$, see proof of Theorem 3.4). However, in many cases it appears to be difficult to show that $\|G\| < 1$ for some matrix norm. This is, for

example, the case with our two model problems. We follow Varga (1962) and define the asymptotic rate of convergence in terms of the spectral radius. Our interest is in the behaviour of the term

$$\delta_k = (\|e_k\| \ / \ \|e_0\|)^{1/k}$$

as $k \to \infty$. We have seen before that $e_{k+1} = G\, e_k$ ((**) in the proof of Theorem 3.4). Thus we can estimate δ_k by $\|G^k\|^{1/k}$.

Using the fact

$$(3.4) \qquad S(A) = \lim_{k \to \infty} (\|G^k\|)^{1/k},$$

for any complex matrix G and arbitrary norm (see Varga 1962, p.67, Young, 1971, p.87),we have

$$(3.5) \qquad \lim_{k \to \infty} (- \ln \|G^k\|^{1/k}) = - \ln S(G) = R_\infty(G).$$

The quantity $R_\infty(G)$ is called the asymptotic rate of convergence. We remark that $R_\infty(G)$ is an asymptotic value and does not necessarily reflect the initial behaviour of a particular iterative process. However, in order to obtain a high convergence rate of the iteratve process (3.1) , one of our objectives should be the creation of a matrix G such that S(G) is as small as possible. At the same time one must decide if the increased amount of work involved in reducing S(G) is justified.

III.3 GENERALIZED JACOBI METHODS

For our purpose we need the iteration (3.1) in a slightly modified form which is more appropriate for the context of this section:

$$u_{K+1} = (I - BA)\, u_K + Bb \,, \tag{3.6}$$

where B is an NxN matrix. In fact, Theorem 2.6 in Young (1971, p.68) shows that iteration (3.1) and iteration (3.6) are equivalent (if B is nonsingular). The matrix B is serving as an approximate inverse to A . The concept of an approximate inverse is fundamental to all the iterative procedures considered in this thesis. The "better" the approximate inverse B , the faster we expect the method (3.6) to converge to the unique solution $u^* = A^{-1}b$. In fact, if $B = A^{-1}$, equation (3.6) shows that $u_{K+1} = u^*$.

III.3.1 JACOBI METHOD

The well-known Jacobi Method uses a diagonal approximate inverse B , which satisfies the following conditions

$$(BA)_{ii} = 1 \quad , \quad i = 1,..,N \quad , \tag{3.7}$$

where B is a diagonal matrix. The Jacobi Method is highly parallel but it shows a slow convergence rate. In the remainder of this section we define better sparse approximate inverses and give a convergence proof for the one dimensional model problem.

III.3.2 JACOBI DIAGONAL-BLOCK METHODS

In case of a one dimensional problem these methods use a p-diagonal ($p \in N$, odd , $p \leqslant 2N-1$) approximate inverse B that satisfies the following condition (Benson,Frederickson,1981)

$$(3.8) \qquad (BA)_{ij} = \delta_{ij} \qquad |i-j| \leqslant (p-1)/2 ,$$

where we assumed that B is a banded matrix with bandwidth $(p-1)/2$.

The non-zero elements of the i'th row of the approximate inverse B are - according to (3.8) - defined as the solution of the following linear system:

$$(3.9) \qquad \begin{bmatrix} a_{i-k_1,i-k_1} & \cdots & & \cdots & a_{i+k_2,i-k_1} \\ \vdots & & \vdots & & \vdots \\ a_{i-k_1,i} & \cdots & a_{ii} & \cdots & a_{i+k_2,i} \\ \vdots & & \vdots & & \vdots \\ a_{i-k_1,i+k_2} & \cdots & & \cdots & a_{i+k_2,i+k_2} \end{bmatrix} x = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} ,$$

where x is some vector and

$$k_1 = \begin{cases} (p-1)/2 & \text{if } i > (p-1)/2 \\ i-1 & \text{if } i \leqslant (p-1)/2 \end{cases}$$

and

$$k_2 = \begin{cases} (p-2)/2 & \text{if } i < N-(p-1)/2 \\ N-i & \text{if } i \geqslant N-(p-1)/2 \end{cases}$$

and the "1" on the right-hand side of (3.9) is in position k_1+1 .

A method of the form (3.6) using an approximate inverse B defined by (3.8) is referred to as Jacobi Diagonal-Block Method (JDB1(p)) in the chapters that follow.

Now we show that the method (3.6) with an approximate inverse defined by (3.8) converges if applied to our one dimensional model problem.

Consider a linear system with the tridiagonal matrix

$$(3.10)\quad A = \begin{bmatrix} -2 & 1 & & & & & \\ 1 & -2 & 1 & & & & \\ & 1 & -2 & 1 & & & \\ & & \cdot & \cdot & \cdot & & \\ & & & \cdot & \cdot & \cdot & \\ & & & & 1 & -2 & 1 \\ & & & & & 1 & -2 \end{bmatrix}.$$

The systems of the form (3.9) now become

$$(3.11)\quad \begin{bmatrix} -2 & 1 & & & & & \\ 1 & -2 & 1 & & & & \\ & \cdot & \cdot & \cdot & & & \\ & & \cdot & \cdot & \cdot & & \\ & & & \cdot & \cdot & \cdot & \\ & & & & 1 & -2 & 1 \\ & & & & & 1 & -2 \end{bmatrix} x = \begin{bmatrix} 0 \\ \cdot \\ 0 \\ 1 \\ 0 \\ \cdot \\ 0 \end{bmatrix},$$

with appropriate modifications near the start and the end of the band of A. Here x is a vector of unknowns and the position of the "1" on the right-hand side is dependent on the row of the approximate inverse B to be determined (see formula (3.9)).

Defining the sequence $\{ r \}$ by

$$r_1 = 1/2$$

(3.12)

$$r_m = (2 - r_{m-1})^{-1} = m/(m+1)$$

the first and the last component of the solution vector f of (3.11) are given by

$$f_1 = \begin{cases} \left(\prod_{m=1}^{k_1} r_m\right) (r_{k_1} - 2 + r_{k_2})^{-1} & , \text{ if } k_1 > 0 \\ (- 2 + r_{k_2})^{-1} & , \text{ if } k_1 = 0 \end{cases}$$

$$f_l = \begin{cases} \left(\prod_{m=1}^{k_2} r_m\right) (r_{k_1} - 2 + r_{k_2})^{-1} & , \text{ if } k_2 > 0 \\ (r_{k_1} - 2)^{-1} & , \text{ if } k_2 = 0 \end{cases}$$

where k_1 and k_2 are the same as in (3.9) and $l = k_1 + k_2 + 1$. This can be simplified to

$$f_1 = - (k_2 + 1) / (k_1 + k_2 + 2)$$

(3.13)

$$f_l = - (k_1 + 1) / (k_1 + k_2 + 2).$$

Thus we have

$$
(3.14)\quad BA = \begin{bmatrix}
1 & 0 & \cdot & \cdot & 0 & c(1) & & & & & & \\
0 & 1 & & & & 0 & & & & & & \\
\cdot & & \cdot & & & & \cdot & & & & & \\
\cdot & & & \cdot & & & & \cdot & & & 0 & \\
\cdot & & & & \cdot & & & & \cdot & & & \\
\cdot & & & & & \cdot & & & & \cdot & & \\
0 & & & & & & & & & c(s+1) & & \\
-.5 & & & & & 1 & & & \cdot & & -.5 & \\
 & \cdot & & & & & \cdot & & & & & \cdot \\
 & & \cdot & & & & & \cdot & & & & \\
 & & & -.5 & & & & 1 & & & & -.5 \\
 & & & & d(N-s) & & & & 1 & & & 0 \\
 & & & & & \cdot & & & & \cdot & & \cdot \\
 & & & & & & \cdot & & & & \cdot & \cdot \\
 & & 0 & & & & & & \cdot & & \cdot & \cdot \\
 & & & & & & & & & \cdot & & \cdot \; 0 \\
 & & & & & & & & & d(N) & 0 \; \cdot \; \cdot \; 0 & 1
\end{bmatrix},
$$

where $s = (p-1)/2$, $c(j) = -j/(s+j+1)$, $j = 1,..,s+1$, and $d(i) = [\, i - (N+1)\,] \,/\, [\, s - (\, i - (N+2)\,)\,]$, $i = N-s,..,N$. The following theorem is an extension of theorem 2.1 of Young (1971,p. 107) for the special case of our one dimensional model problem.

CONVERGENCE THEOREM 3.5

Consider a linear system with the matrix A given in (3.10) and the iterative method (3.6). Let the approximate inverse B be defined by condition (3.8).

Then the Jacobi Diagonal-Block Method converges.

Proof

If $S(I-BA) \geq 1$, there exists an eigenvalue μ of $I-BA$ such that $|\mu| \geq 1$. Now $\det[(I-BA) - \mu I] = 0$. Denoting the associated eigenvector of μ by v we have $[(I-BA) - \mu I]\, v = 0$. Thus $\{I - \mu^{-1}(I-BA)\}\, v = 0$ and we have a nontrivial solution $v \neq 0$ of a homogeneous linear system. Therefore $\det[I - \mu^{-1}(I-BA)] = 0$. For our one dimensional model problem we see from (3.14) that the nonzero elements of the matrix $\mu^{-1}(I-BA)$ are less than one in absolute value (since $|\mu| \geq 1$). Hence the $M = I - \mu^{-1}(I-BA)$ matrix is weakly diagonal dominant (see Young, 1971, p. 107).

Reordering the rows and columns of M, we can get a matrix $\tilde{M}$ of the form

$$\tilde{M} = \begin{bmatrix} \tilde{M}_1 & & & \\ & \tilde{M}_2 & & \\ & & & \\ & & & \tilde{M}_{\frac{p-1}{2}+1} \end{bmatrix}$$

where each block $\tilde{M}_i$ is irreducible and weakly diagonally dominant. Hence by Theorem 2-5.3 (Young, 1971, p 40) $\det \tilde{M} = \det M \neq 0$ and we have a contradiction. ∎

The Diagonal-Block Methods for two dimensional boundary value problems use a striped approximate inverse that has p ($p = 5 + pp*6$, $pp = 0,1,..$) stripes of non-zero entries. For an $(N^2 x N^2)$ matrix A and assuming that $pp < (N-1)/2$, the approximate inverse B with p stripes of non-zero entries is defined by the following conditions

$$
\begin{aligned}
&(B * A)_{ii} = 1 \\
&(B * A)_{i,i+k} = 0 \qquad i < N^2 - k + 1 , \; k=1,2,..,pp+1 \\
&(B * A)_{i,i-k} = 0 \qquad i > k , \; k=1,2,..,pp+1 \\
(3.15) \quad &(B * A)_{i,i+N+k} = 0 \qquad i < N^2 - (N+k) + 1 , \; k=0,1,..,pp \\
&(B * A)_{i,i+N-k} = 0 \qquad i < N^2 - (N-k) + 1 , \; k=0,1,..,pp \\
&(B * A)_{i,i-N+k} = 0 \qquad i > N - k , \; k=0,1,..,pp \\
&(B * A)_{i,i-N-k} = 0 \qquad i > N + k , \; k=0,1,..,pp ,
\end{aligned}
$$

where the indices of the non-zero values of B are the ones used for the conditions (3.15) .

As in the one dimensional case, the non-zero elements of each row of the approximate inverses B are - according to (3.15) - defined as the solution of a small linear system. For the two dimensional model problem with p = 5 (pp = Ø), for instance, a typical system is :

$$(3.16) \qquad \begin{bmatrix} -4 & 0 & 1 & 0 & 0 \\ 0 & -4 & 1 & 0 & 0 \\ 1 & 1 & -4 & 1 & 1 \\ 0 & 0 & 1 & -4 & 0 \\ 0 & 0 & 1 & 0 & -4 \end{bmatrix} x = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} ,$$

In the chapters that follow we refer to methods of the form (3.6) that use an approximate inverse B defined by (3.15) with p stripes of non-zero values as the Jacobi Diagonal-Block (JDB2(p)) Methods.

III.3.3 JACOBI LEAST-SQUARES METHODS

The approximate inverses B for the Jacobi Least-Squares (JLSQ(p)) Method that are defined in this section have the same sparsity pattern as the approximate inverses for the corresponding Jacobi Diagonal-Block Methods. The non-zero entries of B , however, are determined in a different way. Each row of B is defined as the solution of an overdetermined system of linear equations. Suppose that the non-zero entries of the i'th row of B are in the columns $i_1, i_2, \ldots, i_s$ and the indices of the columns of A (the matrix of the linear system to be solved) that have

at least one non-zero element in either one of the rows $i_1, i_2, \ldots, i_s$ are $j_1, j_2, \ldots, j_\ell$, then the overdetermined system for the i'th row of B is given by

$$\begin{bmatrix} a_{i_1 j_1} & \cdots & a_{i_s j_1} \\ \vdots & & \vdots \\ a_{i_1 j_\ell} & \cdots & a_{i_s j_\ell} \end{bmatrix} x = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

where x is a vector of unknowns and the "1" on the right-hand side is in the position of the row that contains the diagonal element a_{ii} of A .

Solving the above systems for each i in the least-squares sense and applying the resulting approximate inverse to the update formula (3.6) we get the JLSQ1(p) and JLSQ2(p) method (p is the number of stripes with non-zero enties) for the one dimensional and two dimensional model problem respectively. We remark that these methods minimize the Frobenius norm of the iteration matrix $\|I - BA\|_F$ over the set of matrices B with a given sparsity pattern . The calculations for each row of B are uncoupled and hence can be performed in parallel.

III.4 GENERALIZED GAUSS-SEIDEL METHODS

As mentioned earlier, each approximate inverse defined above can also be used in a Gauss-Seidel like implementation of (3.1) or (3.6) . In this case each component is updated sequentially so that the most recent data is always used. This increases the speed of convergence, but at a cost of decreased parallelism. However, if the number of processors is not greater than the maximum number of non-zeros of a row of the iteration matrix I-BA , then a Gauss-Seidel like method involves the same amount of work per iteration as the corresponding Jacobi like method. We use this type of parallel Gauss-Seidel method for solving the two dimensional model problem and denote these methods by GDB2(p) and GLSQ2(p) for the diagonal-block and the least-squares versions respectively. In the one dimensional case we use this type of method only in conjunction with the least-squares approximate inverses resulting in the GLSQ1(p) methods, but it can also be defined for diagonal-block approximate inverses.

In the tridiagonal case we can increase the number of operations that may be executed simultaneously, when diagonal-block approximate inverses are used. From (3.14) we conclude that it is possible to divide the set of row indices of the iteration matrix I-BA into two disjoint subsets, such that the update formula for the components

corresponding to the one subset only involves components corresponding to the other subset and vice versa. Thus we can apply the technique of chapter two to obtain a parallel Gauss-Seidel version that has more independent calculations than the above version. In particular, if the approximate inverse B has p stripes of non-zero entries, the subsets of indices are the following :

$$L = \{ 1,..,z,2z+1,..,3z,4z+1,... \} \cap \{ 1,..,N \}$$

$$H = \{ z+1,..,2z,3z+1,..,4z,5z+1,... \} \cap \{ 1,..,N \} ,$$

where $z = (p-1)/2+1$ and N is the size of the linear system under consideration. These methods are referred to as the Gauss-Seidel Diagonal-Block (GDB1(p)) Methods. They are as parallel as the Gauss-Seidel method of chapter 2, that is, they can benefit from as many parallel processors as the number of unknowns. We remark that this is a very special case. The extension of these ideas to more general cases could be a topic for future investigation.

IV. GENERALIZED CONJUGATE GRADIENT METHODS

In this chapter we are concerned with generalized conjugate gradient methods that minimize the error functional over subspaces of increasing dimension (Chandra,1978). Solving the linear system $Ax = b$ with an $N \times N$ symmetric and positive definite matrix is equivalent to minimizing the functional

$$(4.1) \qquad E(x) = (A (x^* - x) , x^* - x) ,$$

where x^* is the solution of the linear system under consideration and $(\cdot , \cdot)$ denotes the usual vector inner-product (this notation will be used throughout this chapter).These methods generate for each iterate x_k a direction p_k of local descent in the sense that there is an α_k^* such that $E(x_k + \alpha p_k) < E(x_k)$ for $\alpha \in (0, \alpha_k^*)$. The next iterate is of the form $x_{k+1} = x_k + \alpha_k p_k$,where the parameter α_k is chosen as the minimum of the error functional E along the direction p_k . The directions and the parameter are chosen in such a way that the sequence of gradients $\{ \mathrm{grad}(E(x_k)) \}_{k \in \mathbb{N}}$ converges to zero. The idea behind this approach is that if $\| \mathrm{grad}(E(x_k)) \|$ is small then usually x_k is near a zero of grad E while the fact that $\{ E(x_k) \}$ is decreasing indicates that this zero of grad E is probably a minimizer of E . In fact, the directions are pairwise M-orthogonal for some symmetric positive definite matrix M and the x_{k+1}

obtained actually minimizes the error functional on the affine subspace $x_0 + \text{span}(p_0, p_1, \ldots, p_k)$,where x_0 is the starting vector (see Chandra,1978).Thus we get the important property that, theoretically, the solution of a linear system with an NxN positive definite matrix is obtained in at most N steps.

The conjugate gradient method (proposed by Hesteness and Stiefel in 1952 as a direct method for solving linear systems) did not, in practice, perform as well as it was expected to from its theoretical properties. The solution procedure was often seriously perturbed by accumulating round-off errors (even for small systems). However, extensive numerical tests with the CG-method used as an iterative method has shown it to be an efficient method for solving large sparse systems (Chandra,1978,Concus,et al.,1976).

Before we discuss the generalized conjugate gradient methods in the form mentioned above, we proceed as follows. The CG algorithm can be generalized using a matrix splitting, $A = M - N$. At each iteration, then, a system with coefficient matrix M must be solved. For rapid convergence, the matrix M should, in some sense, approximate A, that is, the matrix B A , where $B = M^{-1}$, should approximate the identity. Now, solving the linear

system $Ax = b$ is equivalent to solving

(4.2) $$M x = N x + b$$

where $N = M - A$.

Using this approach, the CG algorithm can be stated as a higher order method of the form

(4.3) $$x_{k+1} = x_{k-1} + \omega_{k+1}(\alpha_k z_k + x_k - x_{k-1})$$

where ω_k and α_k are real parameters and the vector z_k is defined by

(4.4) $$M z_k = b - A x_k ,$$

where $M^{-1} = B$. The question of choosing an appropriate matrix splitting is equivalent to the question of choosing an appropriate matrix for scaling (preconditioning). Thus, the matrix B serves for preconditioning the linear system to be solved :

(4.5) $$B A x = B b .$$

Many iterative methods can be described by (4.3) . The Richardson second order method and the Chebyshev semi-iterative method are examples of such a method (see Young,1971,pp. 361-367). Our interest here is in the following generalized conjugate gradient algorithm using the above preconditioning technique. This algorithm is also of the form (4.3) .

Algorithm 1 [Concus, et al.,1976]

Let x_0 be a given vector. Let M be an arbitrary nxn positive definite and symmetric matrix.

1.) Solve $M z_0 = b - A x_0$

and set $\omega_1 = 1$

2.) Compute $\alpha_0 = (z_0 , M z_0)/(z_0 , A z_0)$,

$x_1 = \alpha_0 x_0$,

For k = 1,2,...

3.) Compute

$$\alpha_k = (z_k , M z_k)/(z_k , A z_k) ,$$

$$\omega_{k+1} = \left(1 - \frac{\alpha_k}{\alpha_{k-1}} \frac{(z_k , M z_k)}{(z_{k-1}, M z_{k-1})} \frac{1}{\omega_k}\right)^{-1}, \quad (k \geqslant 1)$$

4.) Compute

$$x_{k+1} = x_{k-1} + \omega_{k+1}(\alpha_k z_k + x_k - x_{k-1}) ,$$

It can be shown (see Concus,et al.,1976), that the calculated vectors $\{ z_k \}_{k=0}^{n}$ are M-orthogonal if we assume that M is positive definite and M and N are symmetric.

The CG algorithm (M = I in the above algorithm) does not require an estimation of the extreme eigenvalues while the Richardson and Chebyshev methods, for example, do. Thus, in cases where nothing is known about the eigenvalues of the coefficient matrix, the CG method should be used. Also, the CG method behaves optimally for a wider class of matrices , than competing methods such as SOR. (see Concus,et al.,1976 and Chandra,1978)

To be more efficient in terms of storage, the following equivalent form of algorithm 1 can be used (see Chandra, 1978).

Algorithm 2 [Chandra, 1978]

Let x_0 be an initial approximation to the solution of (4.5) or (4.2). Let M be a given nxn positive definite and symmetric matrix. Compute the residual

$$r_0 = b - A x_0 ,$$

then solve

$$M z_0 = r_0$$

for z_0 and set $p_0 = z_0$. For $k = 0,1,2,\ldots$ iteratively compute steps (1) through (6)

(1) $a_k = (z_k , r_k)/(p_k , A p_k)$

(2) $x_{k+1} = x_k + a_k p_k$

(3) $r_{k+1} = r_k - a_k A p_k$

(4) Solve $M z_{k+1} = r_{k+1}$ for z_{k+1}.

(5) $b_k = (r_{k+1}, z_{k+1})/(r_k , z_k)$

(6) $p_{k+1} = z_{k+1} + b_k p_k$

In the numerical chapter we use algorithm 2 in a slightly different form in conjunction with the approximate inverses defined in chapter 3 . Instead of solving a linear system at each iteration we compute the vectors d_k by multiplying the residuals with the approximate inverse B . When DB approximate inverses are used, these methods are referred to as DBCG1(I) and DBCG2(I) for the one and two dimensional model problems respectively (I is the number of non-zero stripes in the approximate inverse). In the case of LSQ approximate inverses, these methods are referred to as LSQCG1(I) and LSQCG2(I) . We remark that B^{-1} need not be positive definite and symmetric in the actual implementation. This assumption was made to ensure the finite termination property.

In order to see that algorithm 1 and algorithm 2 are equivalent in the sense that for the same starting vector they create in the absence of round-off error the same sequence $\{ x_k \}$, we rewrite equation (2) of the second algorithm :

$$x_{k+1} = x_{k-1} + (x_k - x_{k-1}) + a_k \{ z_k + b_{k-1} / a_{k-1} (x_k - x_{k-1}) \}$$

and hence

$$(4.6) \quad x_{k+1} = x_{k-1} + a_k z_k + (1 + (a_k b_{k-1})/a_{k-1}) (x_k - x_{k-1})$$

Comparing (4.6) with step (4) in the first algorithm ,

we get the following condition for the parameters

$$a_k = \omega_{k+1}\,\alpha_k$$

(4.7)

$$1 + (a_k\, b_{k-1})/\, a_{k-1} = \omega_{k+1} \;.$$

But, condition (4.7) holds for the parameters a_k, b_k and ω_k, α_k as defined in the above algorithms. This can be seen as follows :

For $k = \emptyset$ both equations in (4.7) hold, since $z_0 = p_0$, $b_0 = \emptyset$ and $\omega_1 = 1$.
Now,

$$\frac{1}{a_k} = \frac{(p_k\,,\,(M-N)\,p_k)}{(z_k\,,\,M\,z_k)}$$

$$= \frac{(z_k + b_{k-1}p_{k-1}, (M-N)(z_k + b_{k-1}p_{k-1})}{(z_k\,,M\,z_k)}$$

$$= \frac{(z_k\,,(M-N)\,z_k) + b_{k-1}^2(p_{k-1}\,,(M-N)\,p_{k-1}) - 2b_{k-1}(z_k\,,N\,z_k)}{(z_k\,,M\,z_k)}\,,$$

because the vectors $\{z_k\}$ are M-orthogonal and each vector p_k is a linear combination of $\{z_i\}_{i=1}^{k}$.
Thus

$$\frac{1}{a_k} = \frac{1}{\alpha_k} + b_{k-1}\frac{1}{a_{k-1}} - 2\,b_{k-1}\frac{1}{\omega_k\,\alpha_{k-1}}\,,$$

where we used (see Concus, et al., 1976)

$$(z_k\,,N\,z_{k-1}) = (z_k\,,M\,z_k)/\,\omega_k\,\alpha_{k-1}$$

and

$$(z_k\,,N\,z_j) = \emptyset \quad \text{for} \quad j < k-1 \;.$$

Supposing

$$\omega_k \alpha_{k-1} = a_{k-1}$$

we get

$$\frac{\alpha_k}{a_k} = \frac{a_{k-1} - b_{k-1}\alpha_k}{a_{k-1}}$$

$$= \left(1 - \frac{\alpha_k}{\alpha_{k-1}} b_{k-1} \frac{1}{\omega_k}\right) = \omega_{k+1}^{-1} .$$

Thus we have proved the first condition of (4.7) by induction. From this result we get

$$1 + \frac{a_k}{a_{k-1}} b_{k-1} = 1 + \frac{\omega_{k+1}\alpha_k}{\omega_k \alpha_{k-1}} b_k$$

$$= 1 + \left(1 - \frac{\alpha_k}{\alpha_{k-1}} b_{k-1} \frac{1}{\omega_k}\right)^{-1} \frac{\alpha_k b_{k-1}}{\alpha_{k-1}\omega_k}$$

$$= 1 + \frac{\alpha_k b_{k-1}}{\alpha_{k-1}\omega_k - \alpha_k b_{k-1}}$$

$$= \{ 1 - (\alpha_k b_{k-1})/(\alpha_{k-1}\omega_k) \}^{-1} = \omega_{k+1} ,$$

which proves that the second condition in (4.7) also holds for the parameters defined in algorithms 1 and 2 .

The speed with which the generalized conjugate gradient algorithm converges depends strongly on the choice of the preconditioning matrix M (or B). For rapid convergence one seeks a splitting so that $M^{-1}N = I-BA$ has small or nearly equal eigenvalues or that it has small rank (see Chandra,1978,th.4.1 and th.3.5). Also we require that M retain any desirable features of A such as sparsity and that the system of equations with coefficient matrix M be

easily solvable. However, when approximating the inverse of A it is desirable that the approximate inverse B be almost as sparse as A.

Several suggestions in the past few years have been made concerning the choice of M for matrices A which arise from discrete approximation to boundary value problems. Stone (1968), for example, determined the matrix M by altering the coefficient matrix A in such a way that the LU-decomposition of the perturbed matrix is sparse in nature. An incomplete LU-decomposition of A is used for the family of Incomplete Cholesky Conjugate Gradient (ICCG) methods (Meijerink, van der Vorst, 1977). During the calculation certain elements are neglected in the L and U matrices in order to keep the preconditioning matrix M sparse. These methods are not ideally suited for vector computers because of the forward - backward substitutions involved.

Dubois, Greenbaum and Rodrigue (1977) replaced the incomplete factorization with an incomplete inverse using a truncated Neumann series. Thus they avoid solving a linear system of the form $M z = d$ at each iteration. The matrix - vector multiplications required can be computed efficiently on a parallel computer. The idea of approximating the inverse of A was further developed by Johnson and Paul (1980, 1981). In 1980 they introduced a new class of methods called incomplete inverse conjugate gradient Jacobi (IICGJ) methods, while in 1981 they

suggested a parameterized form of the incomplete inverse and showed how to select the parameters in order to optimize the convergence of the algorithm. For our test series in the numerical chapter we use the diagonal - block inverses defined in the previous chapter.

Finally we remark that the choice $M = I$, $N = I - A$ leads to the basic unmodified CG algorithm.

V. NUMERICAL EXPERIMENTS

We carried out numerical experiments to demonstrate and compare the efficiency of the methods presented in the previous chapters. For the purpose of comparison we used the one and two dimensional model problems of appendix A . Each iterative process was started with the same initial guess $x^T = (1,\ldots,1)$.

For each test problem the exact solution of the linear system is the zero vector, so that the absolute error of the current iterate can be easily computed. We stopped the iterations when this error was reduced below a specified value.

The methods using an approximate inverse allow us to reduce the number of iterations necessary to achieve a specified accuracy, but often at the expense of increasing the work involved in each iteration (see tables 5.1-5.3). Therefore we used two measures, the number of iterations and the number of multiplications (including divisions), to compare the methods. We ignored the number of additions (including subtractions), since a rough estimate of the work per iteration is sufficient for our purposes.

We also ignored in our work measures the calculations required to establish the iterative process. This greatly simplifies our estimates and is a reasonable approach when the linear system must be solved for many right hand sides. We remark, however, that the small linear systems involved in the calculation of each row of the approximate inverse B can be solved in parallel. For our model problems the diagonal block systems are easy to solve and, except for the ones at the edges, are all alike (see (3.16)). Also, these systems can be solved independently of one another. Thus, if the cost for computing B is ignored, the set-up work consists solely of forming the matrix $G = I - B A$, when methods of the form (3.6) are used. This can also be avoided by rewriting the update formula in the form $u_{k+1} = u_k + B(-Au_k + b)$. But this would increase the amount of work per iteration for the situations we consider. However, we note that there exist approximate inverses where it is more advantageous to use the above iteration formula instead of using the iteration matrix G.

A parallel machine as described in chapter 2 with P independent processors can calculate P multiplications simultaneously with no overhead for data transfer or any other data manipulation. This leads us to define a parallel time unit $T(P)$ as the time required for P independent multiplications. An almost immediate observation concerning each of the Jacobi like methods is that they can benefit from as many parallel processors as the number of

multiplications involved in computing Gu . The methods of chapter 2, PGS1 and PGS2, and the Gauss-Seidel like methods for our dimensional model problem consist of two sequential update passes per iteration. However, a small number of processors (less than N, the number of unknowns) can be equally exploited by the Gauss-Seidel like and the Jacobi like methods using the same approximate inverse in the one dimensional case (see table 5.1). The Gauss-Seidel like methods that we are using for the two dimensional model problem are calculating each component of Gu sequentially, so that the parallelism of these methods is limited to the number of nonzeros in a row of the iteration matrix.

In the one dimensional case we find that there are about 2N multiplications necessary for computing the matrix-vector product Gx , where G = I-BA , if the Diagonal-Block (DB) approximate inverses are used (N). Thus, assuming that N is an upper bound for the number of independent processors, the work necessary for one iteration of the parallel Gauss-Seidel method PGS1 (see chapter 2) is the same as the work involved for one iteration of the JDB1(I) or the GDB1(I) methods (see chapter 3). If more than 2N parallel processors are available, the work needed for the Jacobi like iterations is only half of the work needed for the Gauss-Seidel like iterations (see table 5. 5.1).

Compared to the DB-technique, the Least-Squares (LSQ) technique creates more nonzeros in the iteration matrix G . The number of multiplications per iteration of the JLSQ1(I) (I = 3,5,7,...) methods is (I+2)*N . The numerical experiments showed that the JLSQ1(I) methods converged about as fast as the corresponding JDB1(I) methods, so we conclude that it does not pay to use the iteration matrix I-BA with a least-squares approximate inverse for our one dimensional model problem. The same conclusion applies to our two dimensional model problem. There the costs C(I) for the JDB2(I) (I = 5,11,17,..) methods are $C(I) = (8 + \{ (I+1)/6 - 1 \} 4) N$ multiplications per iteration (see table 5.2), while the costs per iteration for the JLSQ2(I) methods are C(I) + I . However, as mentioned above, there are cases where the iteration formula $u_{k+1} = u_k + B(-Au_k + b)$ is less costly than the formula involving I-BA .

The conjugate gradient method (CG) needs 5N +2 multiplications per iteration plus a matrix-vector product involving the matrix A of the linear system to be solved. The preconditioned conjugate gradient methods (DBCG(I) or LSQCG(I), I = 5,11,17...) need one additional matrix-vector product involving the approximate inverse B (see tables 5.1 and 5.3). To initiate the iterative process the CG method requires one matrix-vector product Au in order to calculate the first residual as well as a vector inner-product. The overhead for the DBCG(I) or LSQCG(I)

methods consists of the computation of the approximate inverse B plus the two matrix-vector products Au and Bu plus one vector inner-product. As above, we choose to neglect this overhead.

The CG and the preconditioned conjugate gradient methods are less parallel than the above Jacobi like methods. Only step (2) and step (3) in algorithm 2 of chapter 4 can be computed in parallel, so that we have five sequential steps that have to be computed for each iteration. Also, the two divisions in step (1) and (5) require two full time units no matter how many parallel processors are available. We may therefore have severe unemployment among the processors. The parallelism of these methods is limited to the calculation of the matrix-vector products, the vector inner-products and the scalar-vector products (see tables 5.4-5.6).

In tables 5.1 - 5.3 we summarize the operation counts for the methods under consideration. In our tables we do not include the operation counts for the methods using the least-squares technique. The number of multiplications per iteration for linear stationary methods using this technique is obtained by simply adding the factor I*N to the number of multiplications necessary for one iteration of the corresponding method using a DB approximate inverse with I stripes of non-zero entries. For the preconditioned conjugate gradient methods the cost for a matrix-vector product involving a LSQ-approximate inverse is the same as

if a DB-approximate inverse were used, so that the total cost per iteration does not change either. Furthermore, since N is large we ignored edges effects in the operation counts. Also, for simplicity the size of the linear system N in each column of any table is always assumed to be a multiple of the number P of independent processors. Thus each given fraction of N is an integer value. The tables 5.4 -5.6 exemplify the different degrees of parallelism (as discussed above) of the methods under consideration. The functions T(P),S(P),E(P) and F(P) are the parallel time unit, the speed-up ratio, the efficiency and the effectiveness as defined in chapter 2 .

In order to graph computational work versus convergence rate, we have selected the methods PGS1 and PGS2 as the basis of comparison for the linear stationary methods of the form (3.6) . All these methods are shown in terms of equivalent iterations of PGS1 or PGS2 respectively. For example, the JDB2(5) method in two dimensions, which requires 8 time units per iteration using N processors, is allowed 1/2 iteration per computational work unit, since one iteration of PGS2 requires just 4 time units in that case. The CG method serves as a basis of comparison for the preconditioned conjugate gradient methods.

In tables 5.7 - 5.11 we give the number of iterations to reduce the error by a specified factor for the methods under consideration for various numbers of unknowns. The JDB1(3) and the JDB2(5) methods reduce the number of iterations compared to the PGS1 and PGS2 methods respectively roughly by a factor of 1/2. The methods JDB1(5) and JDB2(11) further reduce the iteration number roughly by the same factor. However, table 5.11 indicates that as the number of stripes in the approximate inverses increases the advantage gained by including still more stripes diminishes. The method JDB2(35) , for example, reduces the number of iterations compared to the JDB2(29) method only by a factor of 0.98 .

The convergence rates of the JLSQ2(I) methods are somewhat worse than the convergence rates of the JDB2(I) methods. In the one dimensional case, however, the JLSQ1(I) methods converge significantly slower than the corresponding JDB1(I) methods. In particular, the JLSQ1(3) method converges only as fast as the parallel Gauss-Seidel version PGS1. This is still an improvement by a factor of 2 over the convergence rate of the standard Jacobi method in the cases under consideration (The Jacobi method takes 5260,20832 and 46729 iterations for 100, 200 and 300 unknowns respectively to reduce the maximum norm of the error to less than .1).

Figures 5.12 and 5.13 show the computational work in terms of equivalent iterations of the PGS1 and PGS2 methods respectively

versus the error of the Jacobi like methods with various DB approximate inverses. We observe that the JDB2(17) method does not give an improvement over the JDB2(11) method.

Tables 5.7 and 5.9 indicate that the Gauss-Seidel like methods, GDB1(I) and GDB2(I) , improve the convergence rate of the corresponding JDB1(I) and JDB2(I) methods by a factor of 2. The parallelism of the GDB2(I) methods is limited to the number of non-zeros per row in the iteration matrix I-BA. Thus we have to assume a small number of processors in order to gain an advantage over the JDB2(I) methods. In figure 5.14 we compare the GDB2(I) and JDB2(I) methods assuming 16 independent processors. The methods GDB2(11) and GDB2(17) are more efficient than the corresponding methods JDB2(11) and JDB2(17) in this case.

Tables 5.8 and 5.10 show the rate of convergence of the conjugate gradient and the preconditioned conjugate gradient methods. It was found that preconditioning improves the rate of convergence. This is shown for the DBCG2(I) (I=5,11,17) methods in figure 5.15. Similar results have been obtained for the LSQCG2(I) methods. The improvement is smaller than the increase in the computational work unless the number of processors is assumed to be on the order of N (the number of unknowns). This is shown in figures 5.16-5.18.

In conclusion approximate inversion techniques provide an effective means for developing parallel algorithms for the iterative solution of

large sparse linear systems arising in connection with the numerical solution of boundary value problems. In particular the diagonal-block extensions of the Jacobi method performed very well in a parallel environment. The Gauss-Seidel methods in the one dimensional case usually produced an additional improvement. The same result could be accomplished in the two dimensional case when the number of processors was assumed to be small. Finally, in special cases, our approximate inversion techniques proved effective for preconditioning the conjugate gradient method.

WORK REQUIREMENTS IN ONE DIMENSION

	T(1)	T(15)	T(N)	T(2N)	T(7N)
PGS1	2N	2/15 N	2	2	2
JDB1(I)	2N	2/15 N	2	1	1
GDB1(I)	2N	2/15 N	2	2	2
CG	8N + 2	8/15 N + 2	10	8	7
DBCG1(3)	11N + 2	11/15 N + 2	13	10	8
DBCG1(5)	13N + 2	13/15 N + 2	15	11	8
DBCG1(7)	15N + 2	N + 2	17	12	8

Table 5.1 : Work requirements for the methods solving the one dimensional model problem. T(1) is the number of multiplications per iteration. T(P) is the number of time units per iteration using P processors. N is the order of the linear system.

WORK REQUIREMENTS IN TWO DIMENSIONS

	T(1)	T(16)	T(N)	T(16N)
PGS2	4N	1/4 N	4	2
JDB2(5)	8N	1/2 N	8	1
JDB2(11)	12N	3/4 N	12	1
JDB2(17)	16N	N	16	1
GDB2(5)	8N	N	N	N
GDB2(11)	12N	N	N	N
GDB2(17)	16N	N	N	N

Table 5.2 : Work requirements in the two dimensional case ($N = n^2$) for linear stationary methods. T(P) is number of time units per iteration using P processors.

	T(1)	T(N)	T(2N)	T(5N)	T(11N)	T(17N)
CG	10N + 2	12	9	7	7	7
DBCG2(5)	15N + 2	17	12	8	8	8
DBCG2(11)	21N + 2	23	15	10	8	8
DBCG2(17)	27N + 2	29	18	11	9	8

Table 5.3 : Work requirements in the two dimensional case ($N = n^2$) for the preconditioned conjugate gradient methods. T(P) is the number of time steps per iteration using P processors.

MEASURES OF PARALLELISM

	S(15)	S(N)	S(2N)	T(7N)
PGS1	15	N	N	N
JDB1(I)	15	N	2 N	2 N
GDB1(I)	15	N	N	N
CG	15	.8 N	N	1.142 N
DBCG1(3)	15	.846 N	1.1 N	1.375 N
DBCG1(5)	15	.886 N	1.18 N	1.625 N
DBCG1(7)	15	.882 N	1.25 N	1.875 N

Table 5.4 : Estimates of the speed-up ratios in the one dimensional case (see chapter 2 for definition of S(P)).

	E(15)	E(N)	E(2N)	E(7N)
PGS1	1	1	.5	.1428
JDB1(I)	1	1	1	.2856
GDB1(I)	1	1	.5	.1428
CG	1	.8	.5	.1632
DBCG1(3)	1	.845	.55	.1964
DBCG1(5)	1	.866	.59	.2321
DBCG1(7)	1	.882	.625	.2678

Table 5.5 : Estimates of efficiency in the one dimensional case (see chapter 2 for definition of E(P)).

	F(15)	F(N)	F(2N)	F(7N)
PGS1	7.5/N	.5	.25	.Ø714
JDB1(I)	7.5/N	.5	1	.285
GDB1(I)	7.5/N	.5	.25	.Ø714
CG	1.875/N	.Ø8	.Ø625	.Ø233
DBCG1(3)	1.366/N	.Ø65	.Ø55	.Ø245
DBCG1(5)	1.153/N	.Ø577	.Ø537	.Ø29Ø
DBCG1(7)	1/N	.Ø52Ø8	.Ø521	.Ø334

Table 5.6 : Estimates of effectiveness in the one dimensional case (see chapter 2 for definition of F(P)).

CONVERGENCE RATE IN ONE DIMENSION
FOR LINEAR STATIONARY METHODS

	Number of Iterations		
Size N	100	200	300
PGS1	2630	10416	23365
JDB1(3)	1315	5207	11678
JDB1(5)	585	2314	5190
JDB1(7)	329	1302	2920
GDB1(3)	658	2604	5840
GDB1(5)	293	1158	2596
GDB1(7)	165	652	1460
JLSQ1(3)	2629	10413	23354
JLSQ1(5)	1314	5206	11677
JLSQ1(7)	788	3124	7006
GLSQ1(3)	1578	6249	14013
GLSQ1(5)	752	2976	6673
GLSQ1(7)	438	1736	3892

Table 5.7 : Number of iterations to reduce the maximum norm of the error to at least .1 for the one dimensional model problem is given. The size N denotes the number of unknowns.

CONVERGENCE RATE IN ONE DIMENSION
FOR THE CONJUGATE GRADIENT METHODS

	Number of Iterations		
Size N	100	200	300
CG	50	100	150
DBCG1(3)	44	75	105
DBCG1(5)	35	55	89
DBCG1(7)	33	62	82
LSQCG1(3)	34	65	96
LSQCG1(5)	33	58	83
LSQCG1(7)	40	59	82

Table 5.8 : Number of iterations to reduce the maximum norm of the error to at least .01 for the one dimensional model problem is given. The size N denotes the number of unknowns.

CONVERGENCE RATE IN TWO DIMENSIONS
FOR LINEAR STATIONARY METHODS

	Number of Iterations		
Size $N = n^2$	225	400	625
PGS2	479	833	1282
JDB2(5)	187	323	496
JDB2(11)	99	171	263
JDB2(17)	76	132	203
GDB2(5)	95	163	249
GDB2(11)	51	87	133
GDB2(17)	39	67	102
JLSQ2(5)	262	453	696
JLSQ2(11)	150	260	401
JLSQ2(17)	124	216	332
GLSQ2(5)	143	247	378
GLSQ2(11)	80	138	212
GLSQ2(17)	66	114	175

Table 5.9 : Number of iterations to reduce the maximum norm of the error to at least .0001 for the two dimensional model problem is given. The size N denotes the number of grid points.

CONVERGENCE RATE IN TWO DIMENSIONS
FOR THE CONJUGATE GRADIENT METHODS

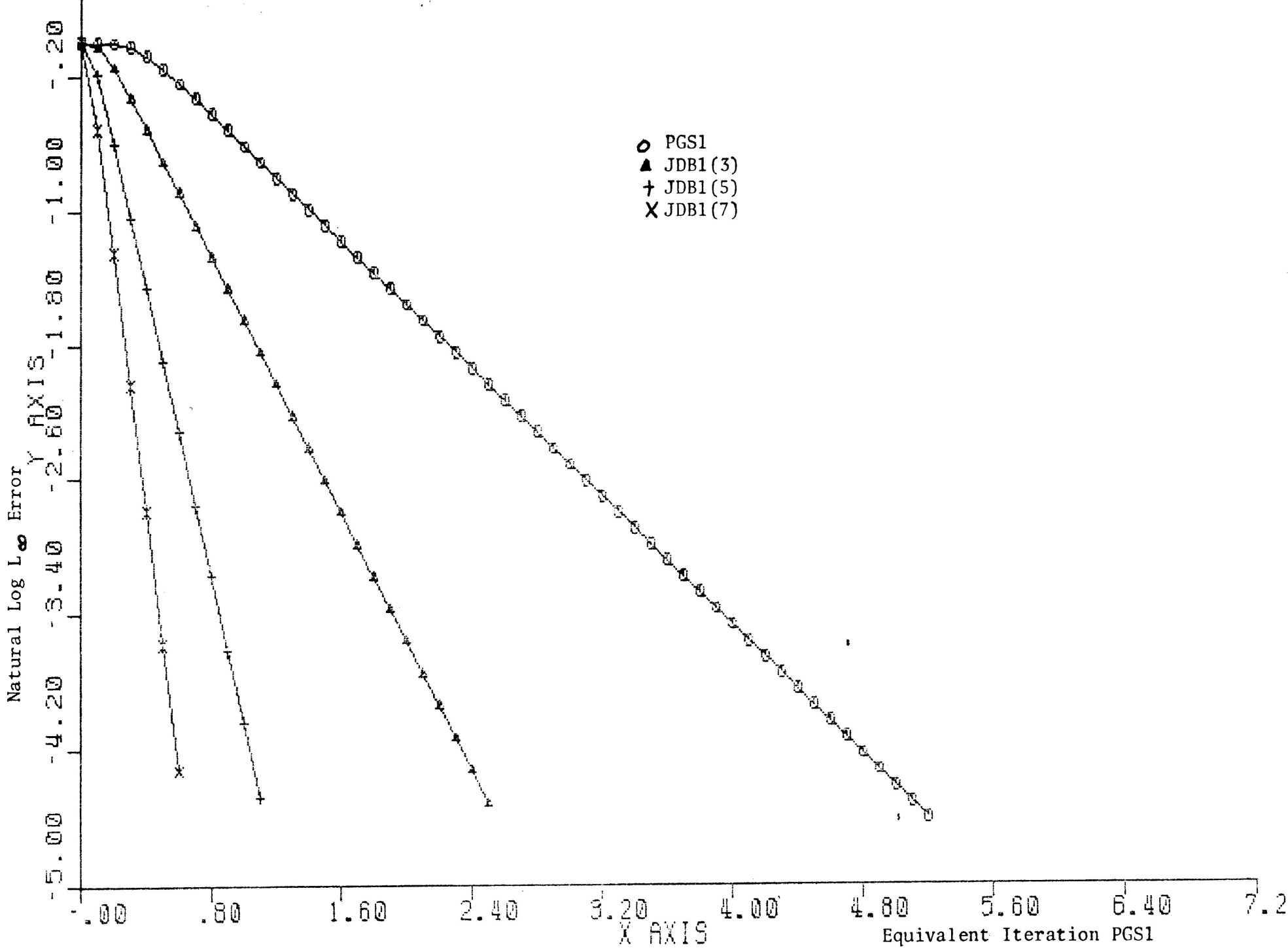
	Number of Iterations		
Size $N = n^2$	225	400	625
CG	23	30	39
DBCG2(5)	17	25	28
DBCG2(11)	15	19	22
DBCG2(17)	16	20	24
LSQCG2(5)	18	21	23
LSQCG2(11)	15	20	24
LSQCG2(17)	16	21	24

Table 5.10 : Number of iterations to reduce the maximum norm of the error to at least .00001 for the two dimensional model problem is given. The size N denotes the number of grid points.

COMPARISON OF DB APPROXIMATE
INVERSES IN TWO DIMENSIONS

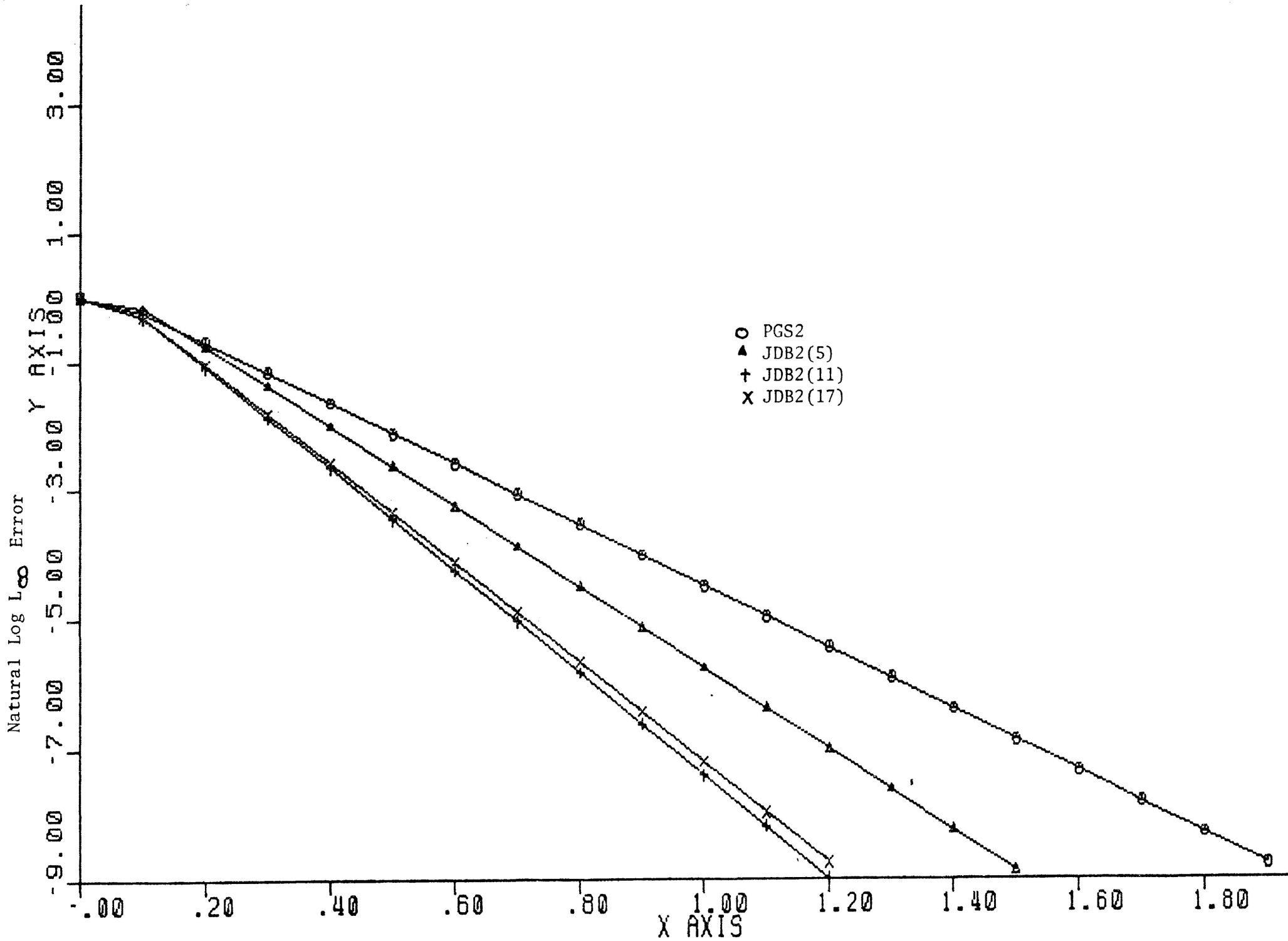
	Number of Iterations		
Size $N = n^2$	225	400	625
JDB2(5)	187	323	496
JDB2(11)	99	171	263
JDB2(17)	76	132	203
JDB2(23)	69	118	182
JDB2(29)	66	113	173
JDB2(35)	65	111	170

Table 5.11 : Number of iterations to reduce the maximum norm of the error to at least .0001 for the two dimensional model problem is given. The size N denotes the number of grid points.



Comparison with Vector Methods,
Convergence versus Computational Work,
100 unknowns, less than N processors.

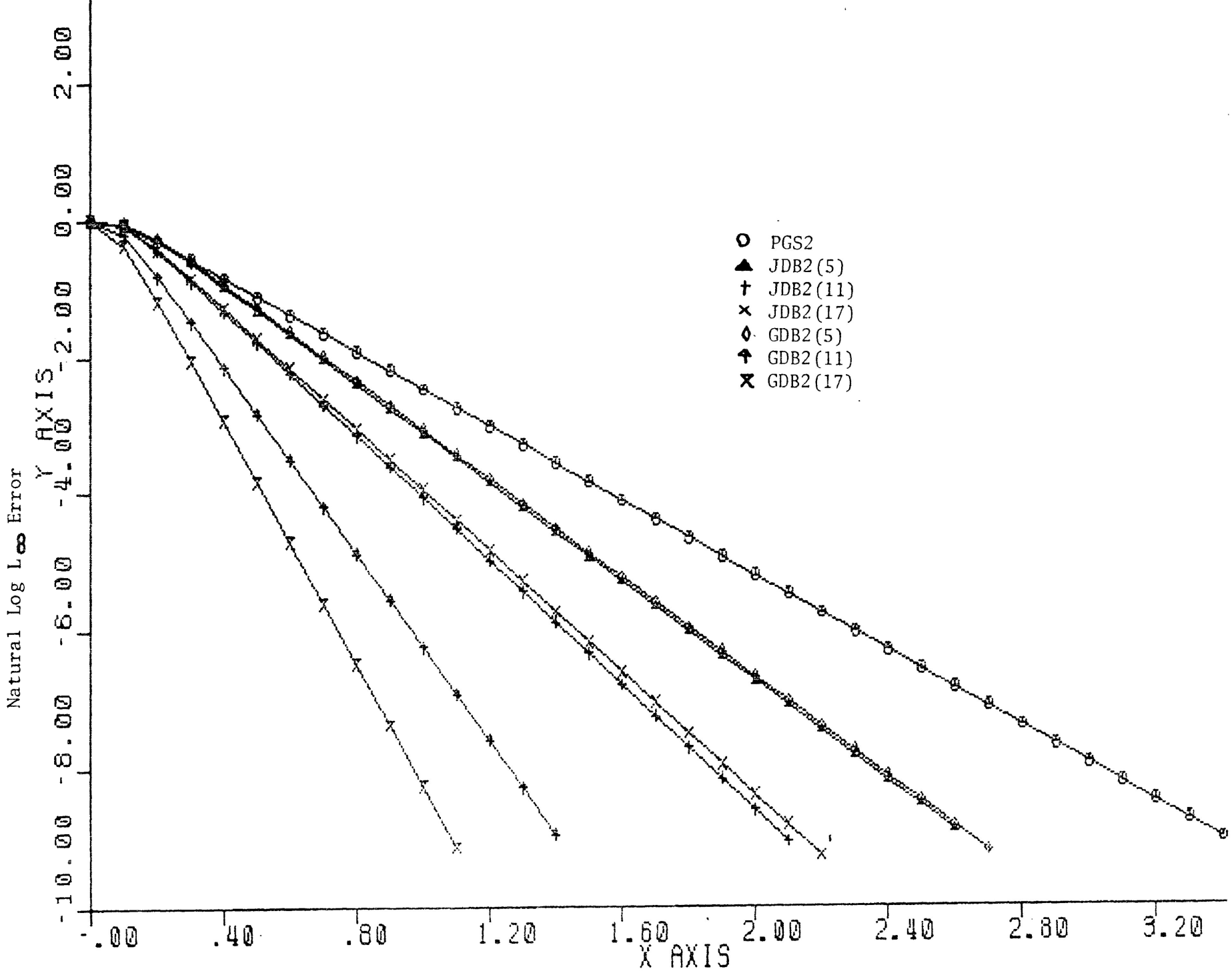
FIGURE 5.12



Comparison with Vector Methods,
Convergence versus Computational Work,
15 x 15 Grid, less than 2N processors.

Equivalent Iteration PGS2

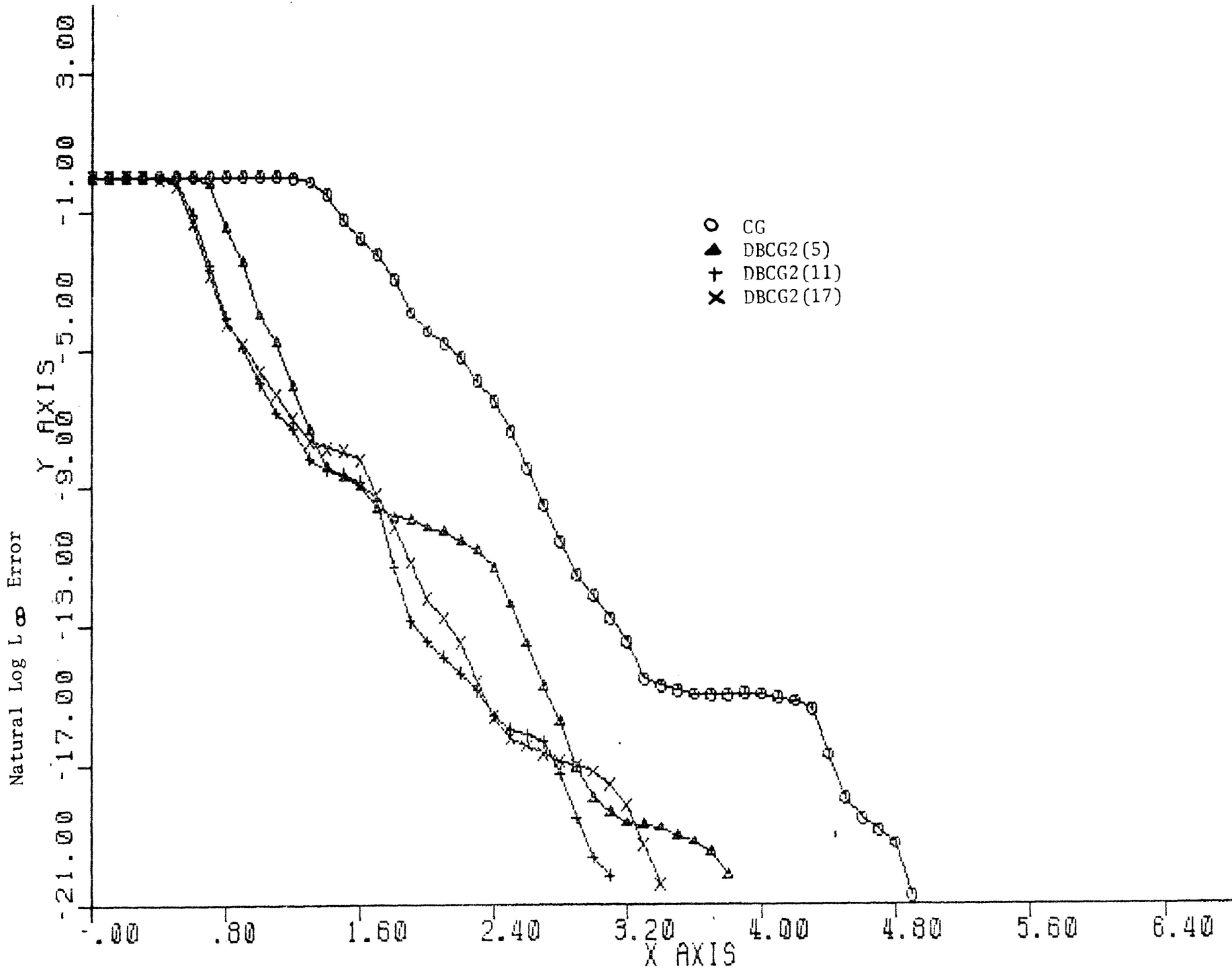
FIGURE 5.13



Comparison with Vector Methods,
Convergence versus Computational Work,
20 x 20 Grid, 16 processors.

Equivalent Iteration PGS2

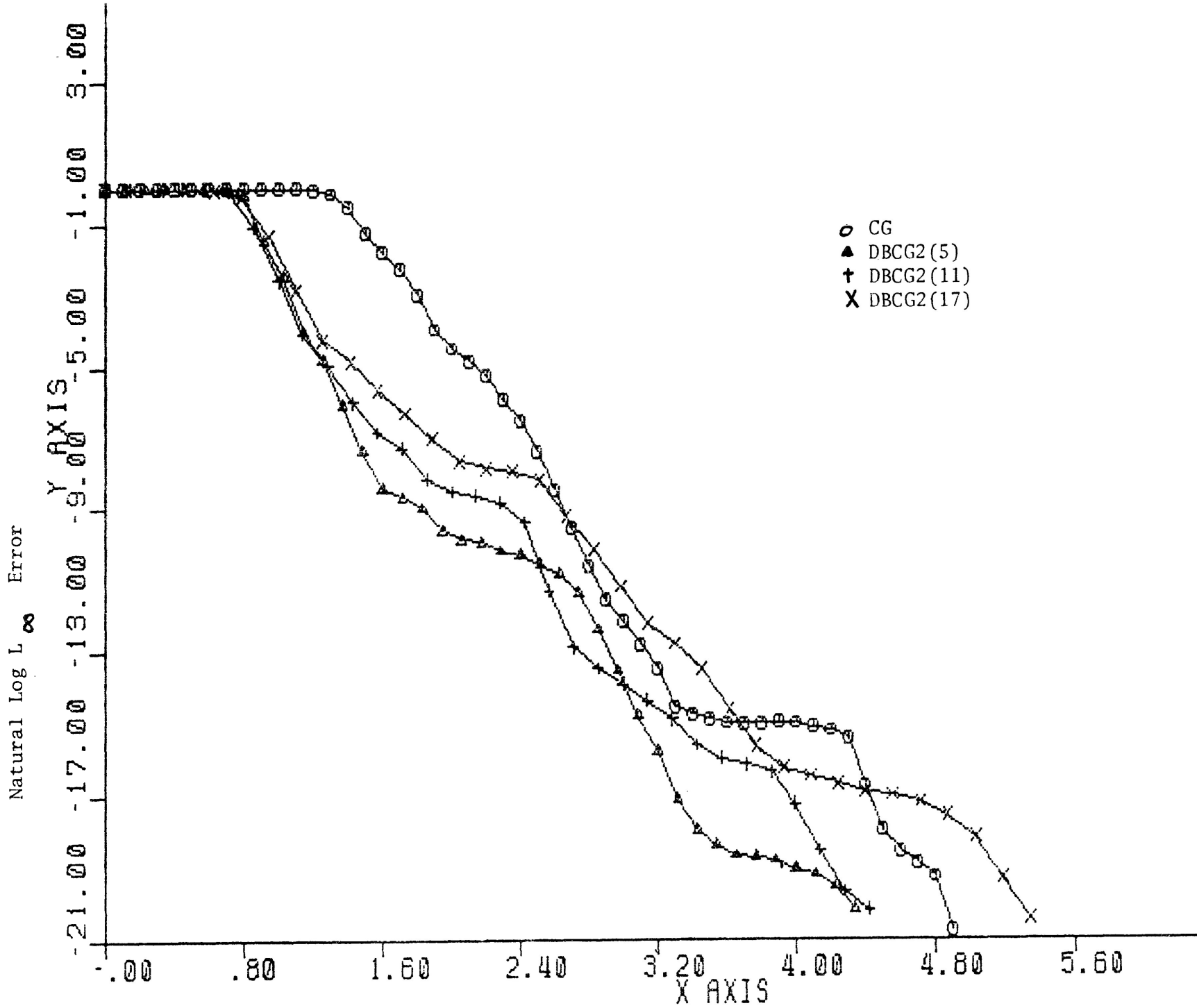
FIGURE 5.14



Comparison with Vector Methods,
Convergence rate,
20 x 20 Grid.

Equivalent Iteration CG.

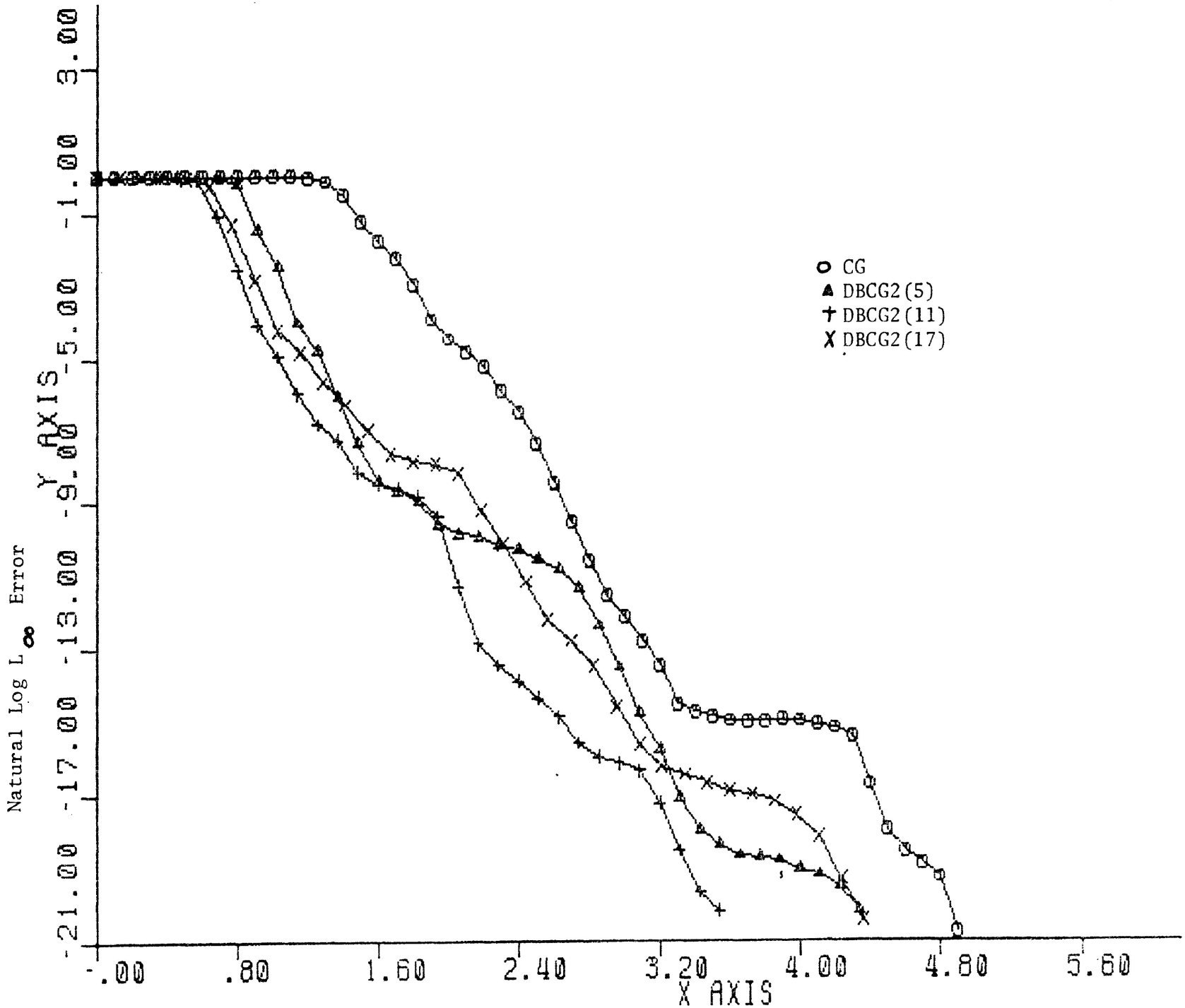
FIGURE 5.15



Comparison with Vector Methods,
Convergence versus Computational Work,
20 x 20 Grid, $5 \cdot 15^2$ processors.

Equivalent Iteration CG.

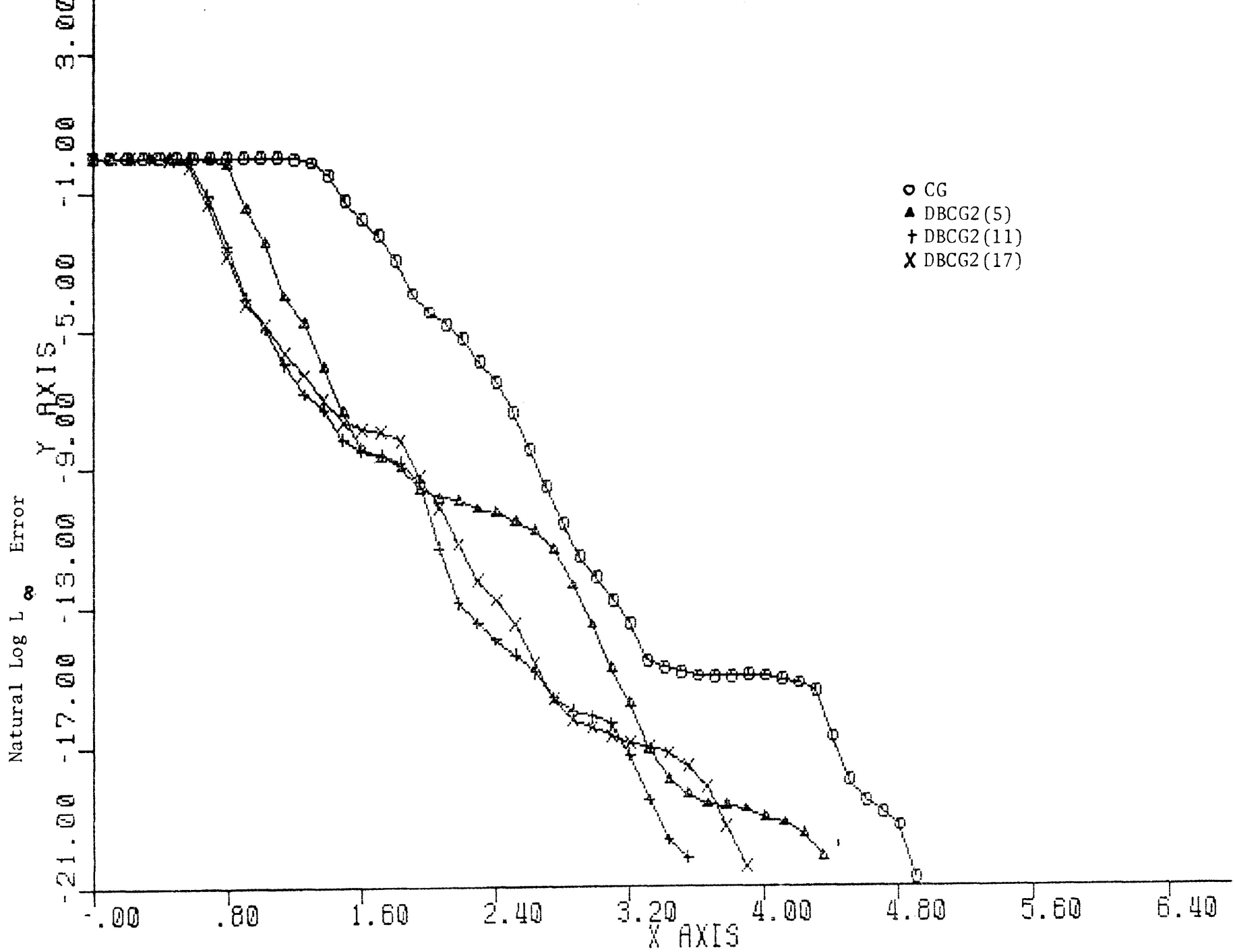
FIGURE 5.16



Comparison with Vector Methods,
Convergence versus Computational Work,
20 x 20 Grid, $11 \cdot 15^2$ processors.

Equivalent Iteration CG.

FIGURE 5.17



Comparison with Vector Methods,
Convergence versus Computational Work,
20 x 20 Grid, $17 \cdot 15^2$ processors.

Equivalent Iteration CG.

FIGURE 5.18

BIBLIOGRAPHY

M.W. Benson, P.O. Frederickson, 1981, Iterative Solution of Large Sparse Linear Systems arising in certain Multidimensional Approximation Problems, to appear in Util. Math.

M.W. Benson, 1973, Iterative Solution of Large Scale Linear Systems, Mathematics Report No. 17-73, Lakehead University,Thunder Bay, Canada.

R.H. Barlow, D.J.Evans, 1982, Parallel Agorithms for the Iterative Solution to Linear Systems, The Computer Journal, Vol.25, No.1, pp.56-60.

P. Budnik, D.J. Kuck, 1971, The Organization and Use of Parallel Memories, IEEE Transations on Computers, Vol. C-20, No. 12, pp.1566-1569.

R. Chandra, 1978, Conjugate Gradient Methods for Partial Differential Equations, Ph.D dissertation Department of Computer Science, Research Report No.129, Yale University, New Haven, CT.

P. Concus, G.H. Golub, D.P.O'Leary, 1976, A Generalized Conjugate Gradient Method for the Numerical Solution of Elliptic Partial Differential Equations in: Sparse Matrix Computation, Academic Press, eds.: J.R. Bund, D.J. Rose, pp.1-22.

P.E. Dubois, A. Greenbaum, G.H. Rodrigue, 1979, Approximating the Inverse of a Matrix for Use in Iterative Algorithms on Vector Processors, Computing 22, pp.257-268.

M.J. Flynn, 1966, Very High-Speed Computing Systems, Proc. IEEE, Vol. 54, pp.1901-1909.

L.S. Haynes, 1982, Highly Parellel Computing:Guest Editor's Introduction, Computer, Vol.15, No.1, pp.7-8.

L.S. Haynes, R.L.Lau, D.P.Siewiork, D.W. Mizell, 1982, A Survey of Highly Parallel Computing, Computer, Vol.15, No.1, pp.9-24.

D. Heller, 1978, A Survey of Parallel Algorithms in Numerical Linear Algebra, Siam Review, Vol.20, No.4, pp740-773.

M.R. Hestenes, E. Stiefel,52, Methods of Conjugate Gradients for Solving Linear Systems, Journal Research of the National Bureau Standards, Vol.49, No.6, Research Paper 2379, pp.409-436.

O. Johnson, G.Paul, 1980, Vector Algorithms for Elliptic Partial Differential Equations Based on the Jacobi Method, Research Rep. RC-8372, IBM, T.J. Wilson Research Center, Yorktown, N.Y., pp.35-351.

O. Johnson, G. Paul, 1981, Optimal Parameterized Incomplete Inverse Preconditioning for Conjugate Gradient Calculations, RC. 8644, IBM, Research Rep., pp.1-9.

H.T. Kung, 1981, Why Systolic Architectures?, Computer, Vol.15, No.1., pp.37-46.

J.A. Meijerink, H.A. van der Vorst, 1977, An Iterative Solution Method for Linear Systems of which the Coefficient Matrix is a Symmetric M-Matrix, MOC, Vol.31, No.137, pp.148-162.

U. Schendel, 1981, Einfuehrung in die Parallele Numerik Oldenburg Verlag, Muenchen, Wien.

H.J. Siegel, 1979, A Model of SIMD Machines and a Comparison of various Interconnection Networks, IEEE Trans. on Comp.,Vol. C-28, No.12, pp.907-917.

H. Stone, 1973, Problems of Parallel Computation: in Complexity of Sequential and Parallel Numerical Algorithms, ed. by J.F. Traub, Academic Press, pp.1-16.

H. Stone, 1968, Iterative Solution of Implicit Approximations of Multidimensional Partial Differential Equations, SIAM J. Numer. Anal., Vol. 5, No. 3, pp.530-558.

R.S. Varga, 1962, Matrix Iterative Analysis, Prentice-Hall, Englewood Cliffs, N.J.

Young, 1971, Iterative Solution of Large Linear Systems, Academic Press.

APPENDIX A

MODEL PROBLEMS

Throughout this thesis we use the following two examples to illustrate our results. The first model problem is the one-dimensional analog of Young's model problem (see Young,1971). This is a simple example of large sparse systems and it is used for testing and comparing the methods proposed in this thesis. It arises from the application of a simple finite difference approximation to the following one-dimensional boundary value problem:

$$(A.1) \qquad u''(x) = \emptyset \ , \ u(a) = u(b) = \emptyset \ , \qquad a \leq x \leq b \ ,$$

where $u : [a,b] \longrightarrow R$ is a twice continously differentiable real valued function and u'' denotes the second derivative of u . The unique analytic solution to this problem is $u(x) = \emptyset$ for all $x \in [a,b]$. We seek a numerical solution to the function that satisfies (A.1) at equally spaced points in $[a,b]$. We first subdivide the interval $[a,b]$ into N+1 intervals of equal lengh, say h , and then replace the second derivative by central differences

$$(A.2) \qquad u''(x) \cong [\, u_{i-1} - 2\, u_i + u_{i+1} \,]/\, h \ , \qquad i = 1,..,N \ ,$$

where $x_i = a + i*h$, $i=1,..,N$ and $x_o = a$, $x_{N+1} = b$ and u_i denotes the approximation of $u(x_i)$.

Thus using (A.1) and (A.2) we get

$$
(A.3) \quad \begin{bmatrix} -2 & 1 & & & & \\ 1 & -2 & 1 & & & \\ & \cdot & \cdot & \cdot & & \\ & & & & & \\ & & & \cdot & \cdot & \cdot \\ & & & 1 & -2 & 1 \\ & & & & 1 & -2 \end{bmatrix} \begin{bmatrix} u_1 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ u_N \end{bmatrix} = \begin{bmatrix} \emptyset \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \emptyset \end{bmatrix},
$$

The matrix of the above linear system is nonsingular, so that the unique solution of the approximate equations (A.3) is the zero-vector. We call this the one-dimensional model problem.

In two dimensions we choose Young's model problem (Young,1971,pp. 2-3). We seek an approximation to the function $u(x,y)$ defined on the unit square which satisfies Laplace's equation

$$
(A.4) \quad u_{xx}(x,y) + u_{yy}(x,y) = \emptyset \ , \quad \emptyset < x,y < 1 \ ,
$$

and

$$
(A.5) \quad u(x,y) = \emptyset
$$

on the boundary of square, where $u_{xx} = \partial^2 u / \partial x^2$ and $u_{yy} = \partial^2 u / \partial y^2$. The solution to this problem is the zero-function, but we will use it to illustrate our

numerical techniques. We superimpose a mesh of horizontal and vertical lines over the square with a uniform spacing $h = (N+1)^{-1}$, for some integer N, and seek to determine approximate values of $u(x,y)$ at the mesh-points. For each mesh-point we replace the differential operator by the usual five-point star difference operator :

$$\text{(A.6)} \quad [\, u_{i-1,j} + u_{i,j-1} - 4\, u_{i,j} + u_{i,j+1} + u_{i+1,j} \,]/\, h^2 \cong u_{xx}(x_i, y_j) + u_{yy}(x_i, y_j),$$

where $i,j = 1,\ldots,N$, $x_i = i*h$, $y_j = j*h$ and u_{ij} denotes the approximation to $u(\, i*h, j*h\,)$. Using (A.4) - (A.6) we get the following sparse linear system of equations

$$\text{(A.7)} \quad \begin{bmatrix} -4 & 1 & & & & 1 & & & & & \\ 1 & -4 & 1 & & & & 1 & & & 0 & \\ & \cdot & \cdot & \cdot & & & & \cdot & & & \\ & & & & & & 0 & & & & \\ 1 & & & \cdot & \cdot & \cdot & & & & \cdot & \\ & & & & 1 & -4 & 0 & & & 1 & \\ & & & & & 0 & -4 & 1 & & & 1 \\ - & & & 0 & & & & \cdot & \cdot & \cdot & 1 \\ & & & 1 & & & & & 1 & -4 & 1 \\ & 0 & & & & & & & & & \\ & & & & 1 & & & & & 1 & -4 \end{bmatrix} \hat{u} = \begin{bmatrix} 0 \\ \cdot \\ \\ \cdot \\ 0 \\ \cdot \\ \\ \cdot \\ 0 \end{bmatrix},$$

where $\hat{u}$ is an N^2-vector, and every N'th element of

the subdiagonal and superdiagonal is a zero and the other two diagonals are N matrix elements apart from the main diagonal. The matrix of the system (A.7) is nonsingular, so that the solution is the zero-vector.

APPENDIX B

FORTRAN SUBROUTINES FOR DATA ORGANIZATION

For our test series we used a data structure that allows for experiments with any striped matrix. Each matrix has its non-zero entries stored in an N by NBND array, where N is the size of the matrix and NBND is the number of stripes with non-zero entries. The column indices of the non-zero entries are stored in an N by NBND integer array. Corresponding elements of these two arrays are in the same position. When we refer to an approximate inverse B , each i'th row of these arrays contains non-zeros in its first c_i locations, where c_i is the number of non-zeros in the i'th row of B . The data is organized somewhat differently for the coefficient matrix A of the linear system to be solved, since we need to fetch columns of A when computing the approximate inverse. Therefore each diagonal of A containing non-zero elements is stored in one column as are the corresponding indices.

In this appendix we list the FORTRAN subroutines that create the data structure described above and also the subroutines that operate on this structure. The subroutine INPUT2 initializes an array that indicates which diagonals of the coefficient matrix under consideration contain non-zero elements. The subroutine INPUT3 finds the corresponding array for the approximate inverse according to

the definitions in chapter 3. These arrays are used by the subroutines BSTRUCT and ASTRUCT . These subroutines initialize the index-arrays for the approximate inverse and the coefficient matrix respectively. The subroutines INIT and INIT2D initialize the coefficient matrices of the one- and two-dimensional model problem respectively. The subroutines MATVEC , ZEILVEC and AX2MAT perform a matrix-vector and restricted matrix-vector multiplications respectivly. The subroutine ZEILVEC multiplies a specified row of a matrix times a vector. This subroutine is used for the Gauss-Seidel like methods (see appendix F). The subroutine AX2MAT performs a restricted matrix-vector multiplication in the sense that only the components between certain bounds are computed.

```
      SUBROUTINE INPUT2( NBNDA,INFOA,ND )
C THIS SUBROUTINE INITIALIZES AN ARRAY
C THAT CONTAINS THE INFORMATION ABOUT
C THE POSITIONS OF THE DIAGONALS OF THE
C WITH NON-ZERO ENTRIES OF THE COEFFICIENT
C MATRIX. IT HANDLES BOTH THE ONE AND
C THE TWO DIMENSIONAL CASE.
C DESCRIPTION OF PARAMETERS.
C NBNDA : NUMBER OF NON-ZERO DIAGONALS OF A.
C         ODD INTEGER VALUE.
C ND    : NUMBER OF GRID POINTS PER ROW IN THE
C         TWO-DIMENSIONAL CASE.
C INFOA : INTEGER ARRAY OF SIZE (NBNDA-1). CONTAINS
C         ON OUTPUT THE DISTANCES OF THE DIAGONALS
C         OF  A  FROM ONE ANOTHER.
      INTEGER NBNDA,ND
      DIMENSION INFOA(NBNDA-1)
      DO  5  I = 1,NBNDA-1
      INFOA(I) = 1
5     CONTINUE
      IF ( NBNDA .EQ. 3 ) GOTO 11
      INFOA(2) = ND - 2
      INFOA(4) = ND - 2
11    CONTINUE
      RETURN
      END
```

```
      SUBROUTINE INPUT3( N,ND,NBNDB,NBNDA,INFOB )
C THIS SUBROUTINE INITIALIZES AN ARRAY
C THAT CONTAINS THE INFORMATION ABOUT
C THE POSITIONS OF THE DIAGONALS WITH NON-ZERO
C ENTRIES IN THE APPROXIMATE INVERSE  B .
C IT HANDLES BOTH, THE ONE AND
C THE TWO DIMENSIONAL CASE.
C DESCRIPTION OF PARAMETERS.
C NBNDB : NUMBER OF NON-ZERO DIAGONALS OF B .
C         ODD INTEGER VALUE.
C NBNDA : NUMBER OF NON-ZERO DIAGONALS OF THE
C         COEFFICIENT MATRIX. ODD INTEGER VALUE.
C ND    : NUMBER OF GRID POINTS PER ROW IN THE
C         TWO-DIMENSIONAL CASE.
C INFOB : INTEGER ARRAY OF SIZE (NBNDB-1). CONTAINS
C         ON OUTPUT THE DISTANCES OF THE DIAGONALS OF
C          B FROM ONE ANOTHER.
      INTEGER N,ND,NBNDB,NBNDA
      DIMENSION INFOB(2,NBNDB-1)
      IF ( NBNDA .EQ. 5 ) GOTO 11
      DO 10  I = 1,NBNDB - 1
      INFOB(1,I) = 1
10    CONTINUE
      GOTO 22
11    CONTINUE
      IHILF = ( NBNDB + 1 )/6
      JPOINT = 1
230   CONTINUE
      DO 200  I = 1,IHILF
      INFOB(1,JPOINT) = 1
      JPOINT = JPOINT + 1
200   CONTINUE
      INFOB( 1,JPOINT ) = ND - 2 * IHILF
      JPOINT = JPOINT + 1
      JH = 2 * IHILF - 2
      IF  (  JH .EQ. 0 ) GOTO 220
      DO 210  I = 1,JH
      INFOB(1,JPOINT) = 1
      JPOINT = JPOINT + 1
210   CONTINUE
220   IF ( JPOINT .NE. NBNDB ) GOTO 230
22    CONTINUE
      RETURN
      END
```

```
      SUBROUTINE BSTRUCT ( N,NBND,INFO,NRAND,IRAND,INDEXB )
C THIS SUBROUTINE INITIALIZES THE INDEX MATRIX OF THE APPROXIMATE
C INVERSE  B .
C DESCRIPTION OF PARAMETERS.
C N     : SIZE OF THE APPROXIMATE INVERSE ( EQUALS NUMBER THE
C         OF UNKNOWNS ).
C NBND  : NUMBER OF DIAGONALS OF  B  CONTAINING NON-ZEROS.
C INFO  : TWO DIMENSIONAL INTEGER ARRAY OF SIZE (2,NBND-1).
C         ON ENTRY, THE ABSOLUTE VALUES OF THE FISRT ROW ARE
C         THE DISTANCES OF THE DIAGONALS OF  B  WITH NON-ZEROS FROM
C         ONE ANOTHER. A NEGATIVE VALUE IN THE FIRST ROW INDICATES
C         THAT CERTAIN VALUES IN THE DIAGONAL CORRESPONDING TO
C         THAT VALUE ARE ZEROS. THE CORRESPONDING VALUE IN THE
C         SECOND ROW OF  INFO  CONTAINS A POINTER TO THE
C         ARRAY  NRAND  WHERE THE ZERO POSITIONS OF THAT
C         PARTICULAR DIAGONAL ARE GIVEN.
C IRAND : NUMBER OF DIAGONALS WITH NON-ZERO ENTRIES WHERE
C         CERTAIN ELEMENTS ARE ZERO.
C NRAND : TWO DIMENSIONAL ARRAY OF SIZE  N,IRAND .
C         CONTAINS INFORMATION ABOUT CERTAIN ZERO VALUES
C         OF THE STORED DIAGONALS OF  B (SEE PARAMETER INFO).
C INDEXB: TWO DIMENSIONAL INTEGER ARRAY OF SIZE  N,NBNDB .
C         CONTAINS ON OUTPUT THE INDEX MATRIX OF THE
C         APPROXIMATE INVERSE.
      INTEGER N,NBND,NRAND
      DIMENSION INDEXB(N,NBND), INFO(2,5Ø),IRAND(N,NRAND)
      IHELP = ( NBND - 1 ) / 2
      DO 1Ø  I = 1,NBND
      DO 5   J = 1,N
      INDEXB(J,I) = Ø
5     CONTINUE
1Ø    CONTINUE
      DO 2Ø  J = 1,N
      ISUM1 = Ø
      ISUM2 = Ø
      IZ1 = Ø
      IZ2 = Ø
      DO 3Ø  K = 1,IHELP
      INF = INFO( 1,K )
      IF  (  INF .EQ. Ø  ) GOTO 11
      ISUM1 = ISUM1 + ABS( INF )
      IF  (  ISUM1 .GT. ( J-1 )  ) GOTO 11
      IZ1 = K
3Ø    CONTINUE
      GOTO 42
11    CONTINUE
      ISUM1 = ISUM1 - ABS( INF )
42    CONTINUE
      DO 4Ø  K = 1,IHELP
      INF = INFO( 1,K+IHELP )
      IF  (  INF .EQ. Ø ) GOTO 22
      ISUM2 = ISUM2 + ABS( INF )
```

```
      IF  (  ISUM2 .GT. ( N-J )  ) GOTO 22
      IZ2 = K
40    CONTINUE
22    CONTINUE
      ICOUNT = Ø
      IF  (  IZ1 .EQ. Ø  ) GOTO 5Ø
      DO 5Ø  I = 1,IZ1
      ICOUNT = ICOUNT + 1
      IK = IZ1 + 1 - I
      INF = INFO( 1,I )
      IF  (  INF .LT. Ø  ) GOTO 33
55    CONTINUE
      INDEXB( J,ICOUNT ) = J - ISUM1
      ISUM1 = ISUM1 - ABS( INFO(1,IK) )
      GOTO 44
33    CONTINUE
      INF = INFO( 2,I )
      IR  = IRAND( J,INF )
      IF  (  IR .EQ. 1 ) GOTO 55
      ICOUNT = ICOUNT - 1
44    CONTINUE
5Ø    CONTINUE
      ICOUNT = ICOUNT + 1
      INDEXB( J,ICOUNT ) = J
      IF  (  IZ2 .EQ. Ø  ) GOTO 6Ø
      ISUM2 = Ø
      DO 6Ø  I = 1,IZ2
      ICOUNT = ICOUNT + 1
      INF = INFO( 1,I+IHELP )
      ISUM2 = ISUM2 + ABS( INF )
      IF  (  INF .LT. Ø  ) GOTO 66
88    CONTINUE
      INDEXB( J,ICOUNT ) = J + ISUM2
      GOTO 77
66    CONTINUE
      INF = INFO( 2,I+IHELP )
      IR  = IRAND( J,INF )
      IF  (  IR  .EQ. 1  ) GOTO 88
      ICOUNT = ICOUNT - 1
77    CONTINUE
6Ø    CONTINUE
2Ø    CONTINUE
      RETURN
      END
```

```
      SUBROUTINE ASTRUCT ( N,NBND,INFO,INDEXA )
C THIS SUBROUTINE INITIALIZES THE INDEX MATRIX OF THE
C COEFFICIENT MATRIX  A .
C DESCRIPTION OF PARAMETERS.
C N      : NUMBER OF UNKNOWNS , SIZE OF  INDEXA.
C NBND   : NUMBER OF DIAGONALS OF  A  CONTAINING NON-ZEROS.
C INFO   : ONE DIMENSIONAL INTEGER ARRAY OF SIZE (NBND-1).
C          ON ENTRY, IT CONTAINS
C          THE DISTANCES OF THE  NON-ZERO DIAGONALS OF  A
C          FROM ONE ANOTHER.
C INDEXA: TWO DIMENSIONAL INTEGER ARRAY OF SIZE  N,NBNDA .
C          CONTAINS ON OUTPUT THE INDEX-MATRIX OF THE
C          COEFFICIENT MATRIX  A .
      INTEGER N,NBND
      DIMENSION INDEXA(N,NBND), INFO(NBND-1)
      IHELP = ( NBND - 1 )/2
      DO 1Ø  I=1,NBND
      DO 5   J=1,N
      INDEXA(J,I) = Ø
5     CONTINUE
1Ø    CONTINUE
      DO 2Ø  I = 1,N
      INDEXA(I,IHELP+1) = I
      ISUM = Ø
      DO 3Ø  J = 1,IHELP
      IK = IHELP + 1 - J
      INF = INFO(J)
      IF  (  INF  .EQ.  Ø  ) GOTO 11
      ISUM = ISUM + INF
      IF  (  (I-ISUM)  .LE.  Ø  ) GOTO 11
      INDEXA( I,IK ) = I - ISUM
3Ø    CONTINUE
11    CONTINUE
      ISUM = Ø
      DO 4Ø  J = 1,IHELP
      IK = IHELP + 1 + J
      INF = INFO ( IHELP + J )
      IF  (  INF .EQ. Ø  ) GOTO 22
      ISUM = ISUM + INF
      IF  (  ISUM .GT. (N-I)  ) GOTO 22
      INDEXA( I,IK ) = I + ISUM
4Ø    CONTINUE
22    CONTINUE
2Ø    CONTINUE
      RETURN
      END
```

```
      SUBROUTINE COLFETCH ( N,NBND,INFO,J,A,COL,ICOL )
C THIS SUBROUTINE FETCHES THE J'TH COLUMN OF THE ARRAY
C STORING THE COEFFICIENT MATRIX AND ALSO THE J'TH COLUMN
C OF THE CORRESPONDING INDEX-MATRIX. REMARK, THE INDEX-MATRIX
C IS NOT NEEDED AS INPUT PARAMETER.
C DESCRIPTION OF PARAMETERS.
C N     : NUMBER OF UNKNOWNS .
C NBND  : NUMBER OF DIAGONALS OF  A  CONTAINING NON-ZEROS.
C INFO  : ONE DIMENSIONAL INTEGER ARRAY OF SIZE (NBND-1).
C         ON ENTRY, IT CONTAINS
C         THE DISTANCES OF THE  'NON-ZERO' DIAGONALS OF  A
C         FROM ONE ANOTHER.
C A     : TWO DIMENSIONAL INTEGER ARRAY OF SIZE  N,NBNDA .
C         CONTAINS ON OUTPUT THE 'NON-ZERO' DIAGONALS OF THE
C         COEFFICIENT MATRIX  A .
C J     : INTEGER VALUE. DETERMINES THE COLUMN TO BE FETCHED.
C COL   : REAL ARRAY OF SIZE  NBND. CONTAINS ON OUPUT THE
C       : J'TH COLUMN OF  A .
C ICOL  : INTEGER ARRAY OF SIZE  NBND. CONTAINS ON OUTPUT THE
C         THE COLUMN INDICES OF THE ELEMENTS IN THE ARRAY COL.
      IMPLICIT REAL *8 (A-H,O-Z)
      INTEGER N,NBND,J
      DIMENSION INFO(NBND-1),COL(NBND),ICOL(NBND),A(N,NBND)
      IHELP = (NBND-1)/2
      DO 1Ø  I = 1,NBND
      COL(I) = Ø.Ø
      ICOL(I) = Ø
1Ø    CONTINUE
      COL(IHELP+1) = A(J,IHELP+1)
      ICOL(IHELP+1) = J
      ISUM = Ø
      DO 2Ø  I = 1,IHELP
      IK = IHELP + 1 - I
      INF = INFO(I)
      IF  (  INF .EQ. Ø  ) GOTO 11
      ISUM = ISUM + INF
      IF  (  ISUM .GT. (N-J)  ) GOTO 11
      COL( IK ) = A( J+ISUM,IK )
      ICOL( IK ) = ISUM + J
2Ø    CONTINUE
11    CONTINUE
      ISUM = Ø
      DO 3Ø  I = 1,IHELP
      IK = IHELP + 1 + I
      INF = INFO( IHELP+I )
      IF  (  INF .EQ. Ø  ) GOTO 22
      ISUM = ISUM + INF
      IF  (  (J-ISUM) .LE. Ø  ) GOTO 22
      COL( IK ) = A( J-ISUM,IK )
      ICOL( IK ) = J - ISUM
3Ø    CONTINUE
22    CONTINUE
      RETURN
      END
```

```
      SUBROUTINE INIT( ND,A,RH,XK )
C THIS SUBROUTINE INITIALIZES THE TRIDIAGONAL MATRIX A AND
C THE RIGHT-HAND SIDE RH OF THE LINEAR SYSTEM TO BE SOLVED.
C IT ALSO INITIALIZES THE STARTING VECTOR XK FOR THE ITERATION.
C ND IS THE NUMBER OF SUBINTERVALS AFTER DISCRETIZATION.
      INTEGER ND
      DIMENSION A(ND-1,3),RH(ND-1),XK(ND-1)
      DO 10  I = 1,ND-1
      A(I,1) = 1.
      A(I,2) = - 2.
      A(I,3) = 1.
      RH(I) = 0.0
      XK(I) = 1.0
10    CONTINUE
      A(1,1) = 0.0
      A(ND-1,3) = 0.0
      RETURN
      END
```

```
      SUBROUTINE INIT2D ( ND,UNY,A,RH,XK,N )
C THIS SUBROUTINE INITIALIZES THE FIVE-DIAGONAL MATRIX A AND THE
C RIGHT-HAND SIDE RH OF THE LINEAR SYSTEM RESULTING FROM THE DISCRE-
C TIZATION OF LAPLACE'S EQUATION ON A NxN SQUARE WITH THE FOLLOWING
C BOUNDARY CONDITIONS :
C       U(X,Ø) = Ø
C       U(X,L2) = Ø
C       U(Ø,Y) = Ø
C       U(L1,Y) = UNY.
C IT INITIALIZES ALSO THE STARTING VECTOR XK FOR THE ITERATION.
C ND IS THE NUMBER OF SUBINTERVALS ALONG THE X-AXIS.
      IMPLICIT REAL *8 (A-H,O-Z)
      INTEGER ND,N
C     REAL L1,L2,UNY
      DIMENSION A(N,5),RH(N),XK(N)
      N1 = ND - 1
      DO 1Ø  I = 1,N
      RH(I) = Ø.Ø
      XK(I) = 1.Ø
      DO 2Ø  J = 1,5
      A(I,J) = Ø.Ø
2Ø    CONTINUE
1Ø    CONTINUE
      DO 3Ø  I = 1,N-1
      IF (  MOD(I,N1) .EQ. Ø ) GOTO 11
      A(I,4) = 1.
      A(I+1,2) = 1.
      GOTO 22
11    CONTINUE
      RH(I) = - UNY
22    CONTINUE
      A(I,3) = -4.
3Ø    CONTINUE
      A(N,3) = -4.
      RH(N)  = - UNY
      DO 4Ø  I = 1,N-N1
      IK = I + N1
      A(IK,1) = 1.
      A(I,5)  = 1.
4Ø    CONTINUE
      RETURN
      END
```

```
      SUBROUTINE MATVEC( N,NBND,RM,INDEXM,X )
C THIS SUBROUTINE PERFORMS A MATRIX-VECTOR MULTIPLICATION
C FOR SPARSE MATRICES STORED IN THE FOLLOWING WAY :
C THE NON-ZEROS OF THE I'TH ROW ARE STORED
C IN THE I'TH ROW OF THE MATRIX  RM  AND THE COLUMN INDICES
C OF THESE NON-ZEROS ARE STORED IN THE CORRESPONDING POSITIONS
C OF THE I'TH ROW OF  INDEXM.
C DESCRIPTION OF PARAMETERS.
C N      : NUMBER OF COLUMNS.
C NBND   : MAXIMUM NUMBER OF NON-ZEROS PER ROW.
C RM     : TWO DIMENSIONAL REAL ARRAY OF SIZE  (N,NBND).
C          CONTAINS ON ENTRY THE NON-ZERO ELEMENTS OF
C          MATRIX INVOLVED (SEE ABOVE).
C INDEXM: TWO DIMENSIONAL INTEGER ARRAY OF SIZE  N,NBND.
C          CORRESPONDING INDEX-MATRIX TO RM AS DESCRIBED ABOVE.
C X      : REAL ARRAY OF SIZE  N. CONTAINS ON ENTRY THE VECTOR
C          OPERAND AND ON OUTPUT THE RESULT.
      IMPLICIT REAL *8 (A-H,O-Z)
      INTEGER N,NBND
      DIMENSION RM(N,NBND), INDEXM(N,NBND), X(N)
C     REAL SCRATCH(4900),SUM
      REAL *8 SCRATCH(2500),SUM
      DO  10  I = 1,N
      SUM = 0.0
      DO  20  J = 1,NBND
      INF = INDEXM(I,J)
      IF  (  INF .EQ. 0  ) GOTO 11
      SUM = SUM + RM(I,J) * X(INF)
11    CONTINUE
20    CONTINUE
      SCRATCH(I) = SUM
10    CONTINUE
      DO  30  I = 1,N
      X(I) = SCRATCH(I)
30    CONTINUE
      RETURN
      END
```

```
      SUBROUTINE ZEILVEC(I,N,NBND,RM,INDEXM,X )
C THIS SUBROUTINE PERFORMS THE MULTIPLICATION OF A VECTOR AND THE
C I'TH ROW OF A SPARSE MATRIX STORED IN THE FOLLOWING WAY :
C THE NON-ZEROS OF THE I'TH ROW ARE STORED
C IN THE I'TH ROW OF THE MATRIX  RM  AND THE COLUMN INDICES
C OF THESE NON-ZEROS ARE STORED IN THE CORRESPONDING POSITIONS
C OF THE I'TH ROW OF  INDEXM.
C DESCRIPTION OF PARAMETERS.
C I      : INTEGER VALUE. DETERMINES THE ROW TO BE SELECTED.
C N      : NUMBER OF COLUMNS.
C NBND   : MAXIMUM NUMBER OF NON-ZEROS PER ROW.
C RM     : TWO DIMENSIONAL REAL ARRAY OF SIZE  (N,NBND).
C          CONTAINS ON ENTRY THE NON-ZERO ELEMENTS OF
C          MATRIX INVOLVED (SEE ABOVE).
C INDEXM: TWO DIMENSIONAL INTEGER ARRAY OF SIZE  N,NBND.
C          CORRESPONDING INDEX-MATRIX TO RM AS DESCRIBED ABOVE.
C X      : REAL ARRAY OF SIZE  N. CONTAINS ON ENTRY THE VECTOR
C          OPERAND AND ON OUTPUT THE RESULT.
      IMPLICIT REAL *8 (A-H,O-Z)
      INTEGER N,NBND,I
      DIMENSION RM(N,NBND), INDEXM(N,NBND), X(N)
C     REAL SUM
      REAL *8 SUM
      SUM = Ø.Ø
      DO  2Ø  J = 1,NBND
      INF = INDEXM(I,J)
      IF  (  INF .EQ. Ø  ) GOTO 11
      SUM = SUM + RM(I,J) * X(INF)
11    CONTINUE
2Ø    CONTINUE
      X(I) = SUM
      RETURN
      END
```

```
      SUBROUTINE AX2MAT(KI,DIM,N,NBND,RM,INDEXM,X )
C THIS SUBROUTINE PERFORMS A RESTRICTED MULTIPLICATION OF A VECTOR
C AND A SPARSE MATRIX STORED IN THE FOLLOWING WAY :
C THE NON-ZEROS OF THE I'TH ROW ARE STORED
C IN THE I'TH ROW OF THE MATRIX  RM  AND THE COLUMN INDICES
C OF THESE NON-ZEROS ARE STORED IN THE CORRESPONDING POSITIONS
C OF THE I'TH ROW OF  INDEXM. THE MULTIPLICATION IS RESTRICTED IN
C THE SENSE THAT ONLY THE COMPONENTS BETWEEN CERTAIN BOUNDS ARE
C COMPUTED.
C DESCRIPTION OF PARAMETERS.
C KI     : INTEGER VALUE. USED FOR CALCULATING THE BOUNDS FOR THE
C          MULTIPLICATION.
C DIM    : INTEGER VALUE. USED FOR CALCULATING THE BOUNDS FOR THE
C          MULTIPLICATION.
C N      : NUMBER OF COLUMNS.
C NBND   : MAXIMUM NUMBER OF NON-ZEROS PER ROW.
C RM     : TWO DIMENSIONAL REAL ARRAY OF SIZE  (N,NBND).
C          CONTAINS ON ENTRY THE NON-ZERO ELEMENTS OF
C          MATRIX INVOLVED (SEE ABOVE).
C INDEXM: TWO DIMENSIONAL INTEGER ARRAY OF SIZE  N,NBND.
C          CORRESPONDING INDEX-MATRIX TO RM AS DESCRIBED ABOVE.
C X      : REAL ARRAY OF SIZE  N. CONTAINS ON ENTRY THE VECTOR
C          OPERAND AND ON OUTPUT THE RESULT.
      IMPLICIT REAL *8 (A-H,O-Z)
      INTEGER N,NBND,KI,DIM
      DIMENSION RM(N,NBND), INDEXM(N,NBND), X(N)
C     REAL SCRATCH(4900),SUM
      REAL *8 SCRATCH(2500),SUM
      JL = KI - DIM - 2
      JU = KI + DIM + 2
      ILOWER = MAX0( 1,JL)
      IUPPER = MIN0( N,JU )
      DO  10  I = ILOWER,IUPPER
      SUM = 0.0
      DO  20  J = 1,NBND
      INF = INDEXM(I,J)
      IF  (  INF .EQ. 0  ) GOTO 11
      SUM = SUM + RM(I,J) * X(INF)
11    CONTINUE
20    CONTINUE
      SCRATCH(I) = SUM
10    CONTINUE
      DO  30  I = ILOWER,IUPPER
      X(I) = SCRATCH(I)
30    CONTINUE
      RETURN
      END
```

APPENDIX C

FORTRAN SUBROUTINES FOR FINDING THE DB AND LSQ APPROXIMATE INVERSES

In this appendix, we list FORTRAN subroutines for finding Diagonal Block and Least-Squares approximate inverses in the one and two-dimensional cases (see chapter 3). The subroutine APPRINV handles the one-dimensional case and the subroutine AINV2 handles the two-dimensional case. The arguments are explained in the programs.

```
      SUBROUTINE APPRINV (N,NBNDA,NBNDB,A,INDEXA,INDEXB,INFOA,IFLAG,B)
C THIS SUBROUTINE FINDS A BANDED DIAGONAL OR LEAST-SQUARES APPROXIMATE
C INVERSE  B  OF BANDWIDTH  NBNDB .
C DESCRIPTION OF PARAMETERS.
C N      : NUMBER OF GRID POINTS.
C NBNDA  : NUMBER OF 'NON-ZERO' DIAGONALS OF THE COEFFICIENT MATRIX  A.
C NBNDB  : NUMBER OF 'NON-ZERO' DIAGONALS OF THE APPROXIMATE INVERSE B.
C A      : TWO DIMENSIONAL REAL ARRAY OF SIZE  N,NBNDA. CONTAINS ON
C          ENTRY THE NON-ZERO DIAGONALS OF THE COEFFICIENT MATRIX IN
C          ITS COLUMNS.
C INDEXA: TWO DIMENSIONAL INTEGER ARRAY OF SIZE  N,NBNDA. CONTAINS
C          THE COLUMN INDICES OF THE ELEMENTS THAT ARE STORED IN  A
C          MATRIX IN  THE CORRESPONDING LOCATIONS.
C INDEXB: TWO DIMENSIONAL INTEGER ARRAY OF SIZE  N,NBNDB. CONTAINS,
C          ON ENTRY THE COLUMN INDICES OF NON-ZERO ELEMENTS OF THE
C          APPROXIMATE INVERSE  B.
C INFOA : INTEGER ARRAY OF SIZE  (NBNDA-1). CONTAINS ON ENTRY THE
C          DISTANCES OF THE 'NON-ZERO' DIAGONALS.
C B      : TWO DIMENSIONAL REAL ARRAY OF SIZE  N,NBNDB . CONTAINS ON
C          OUTPUT THE NON-ZERO ELEMENTS OF THE APPROXIMATE INVERSE
C          IN THE LOCATIONS GIVEN BY THE ARRAY  INDEXB .
C IFLAG : INTEGER VALUE. IF' IFLAG=1 A DIAGONAL-BLOCK APPROXIMATE
C          INVERSE IS FOUND, A LEAST SQUARES APPROXIMATE INVERSE
C          IF IFLAG=Ø.
      INTEGER N,NBNDA,NBNDB,IFLAG
      DIMENSION A(N,NBNDA),B(N,NBNDB),INDEXA(N,NBNDA),INDEXB(N,NBNDB),
     1 INFOA(NBNDA-1)
      REAL MAT(49ØØ,52),RH1(49ØØ),COL(51),X(51)
      INTEGER ICOL(51),IVEC(49ØØ),IX(51)
      DO 1Ø  I = 1,N
      DO 2Ø  J = 1,NBNDB
      B(I,J) = Ø.Ø
2Ø    CONTINUE
1Ø    CONTINUE
      DO 3Ø  J = 1,N
      IZ = 1
      DO  35  I = 1,N
      IVEC(I) = Ø
      RH1(I) = Ø.Ø
      DO  36  II = 1,NBNDB
      MAT(I,II) = Ø.Ø
36    CONTINUE
35    CONTINUE
C FIND DIMENSIONS OF THE OVERDETERMINED SYSTEM AND
C THE COLUMNS OF A THAT ARE INVOLVED.
      DO 4Ø  K = 1,NBNDB
      INFB = INDEXB(J,K)
      IF  (  INFB .EQ. Ø  ) GOTO 11
      ICOUNT = K
      DO 5Ø  KK = 1,NBNDA
      INFA = INDEXA( INFB,KK )
      DO 6Ø  II = 1,N
      IF  (  INFA .EQ. IVEC( II )  ) GOTO 22
```

```
60      CONTINUE
        IVEC( IZ ) = INFA
        IZ = IZ + 1
22      CONTINUE
50      CONTINUE
40      CONTINUE
11      CONTINUE
        IDIM = IZ - 1
        IF  (  IFLAG .EQ. 0  ) GOTO 55
C FIND COLUMNS OF A FOR QUADRATIC DIAGONAL-BLOCK SYSTEM.
C SORT OF IVEC.
        DO  110  I = 1,IDIM-1
        IZ = IVEC(I)
        IZ1 = I
        DO  120  II = I+1,IDIM
        IF  (  IZ .LE. IVEC(II)  ) GOTO 120
        IZ = IVEC(II)
        IZ1 = II
120     CONTINUE
        IVEC(IZ1) = IVEC(I)
        IVEC(I) = IZ
110     CONTINUE
C FIND INDEX OF DIAGONAL.
        DO  130  I = 1,N
        IF  (  IVEC(I) .EQ. J  ) IDIAG = I
130     CONTINUE
        JHELP = ( NBNDB - 1 )/2  +  1
        DO  140  I = 1,NBNDB
        IX(I) = 0
140     CONTINUE
        IX(JHELP) = J
        I = 0
        JC = ICOUNT - 1
66      IF  (  JC  .EQ. 0  ) GOTO 77
        I = I + 1
        ID = IDIAG - I
        IF  (  ID .LE. 0  ) GOTO 88
        IX( JHELP-I ) = IVEC( ID )
        JC = JC - 1
        IF  (  JC .EQ. 0  ) GOTO 77
88      CONTINUE
        ID = IDIAG + I
        IF  (  ID .GT. IDIM  ) GOTO 99
        IX( JHELP+I ) = IVEC( ID )
        JC = JC - 1
99      CONTINUE
        GOTO 66
77      CONTINUE
        JKZ = 1
        DO  150  I = 1,NBNDB
        IF  (  IX(I) .EQ. 0  ) GOTO 150
        IVEC(JKZ) = IX(I)
        JKZ = JKZ + 1
150     CONTINUE
        IDIM = ICOUNT
```

```
55      CONTINUE
C FIND DIAGONAL BLOCK SYSTEM.
        DO 7Ø  I = 1,IDIM
        IV = IVEC(I)
        CALL COLFETCH( N,NBNDA,INFOA,IV,A,COL,ICOL )
        DO 8Ø  II = 1,ICOUNT
        INFB = INDEXB(J,II)
        DO 9Ø  IJ = 1,NBNDA
        IK = NBNDA + 1 - IJ
        INFA = ICOL(IK)
        IF  (  INFA .EQ. INFB  ) GOTO 33
9Ø      CONTINUE
        GOTO 44
33      CONTINUE
        MAT( I,II ) = COL( IK )
44      CONTINUE
8Ø      CONTINUE
        IF  (  IV .EQ. J  ) RH1(I) = 1.Ø
7Ø      CONTINUE
C SOLVE DIAGONAL BLOCK SYSTEM
        IF  (  IFLAG .EQ. 1 ) GOTO 111
        CALL HOUSE( MAT,RH1,IDIM,ICOUNT,X,JFLAG,49ØØ,51 )
        GOTO 222
111     CONTINUE
        DO  16Ø  I = 1,ICOUNT
        MAT(I,ICOUNT+1) = RH1(I)
16Ø     CONTINUE
        CALL GAUSS( MAT,X,ICOUNT,ICOUNT+1,49ØØ,51)
222     CONTINUE
        DO 1ØØ  I = 1,ICOUNT
        B(J,I) = X(I)
1ØØ     CONTINUE
3Ø      CONTINUE
        RETURN
        END
```

```
      SUBROUTINE AINV2 ( N,NBNDA,NBNDB,A,INDEXA,INDEXB,INFOA,IFLAG,
     1 B,ND )
C THIS SUBROUTINE FINDS A DIAGONAL-BLOCK OR A LEAST-SQUARES APPROXIMATE
C INVERSE IN THE TWO DIMENSIONAL CASE.
C DESCRIPTION OF PARAMETERS.
C N     : NUMBER OF GRID POINTS.
C ND    : NUMBER OF GRID POINT IN A ROW ( ND*ND = N ).
C NBNDA : NUMBER OF 'NON-ZERO' DIAGONALS OF THE COEFFICIENT MATRIX  A.
C NBNDB : NUMBER OF 'NON-ZERO' DIAGONALS OF THE APPROXIMATE INVERSE B.
C A     : TWO DIMENSIONAL REAL ARRAY OF SIZE  N,NBNDA. CONTAINS ON
C         ENTRY THE NON-ZERO DIAGONALS OF THE COEFFICIENT MATRIX IN
C         ITS COLUMNS.
C INDEXA: TWO DIMENSIONAL INTEGER ARRAY OF SIZE  N,NBNDA. CONTAINS
C         THE COLUMN INDICES OF THE ELEMENTS THAT ARE STORED IN  A
C         MATRIX IN  THE CORRESPONDING LOCATIONS.
C INDEXB: TWO DIMENSIONAL INTEGER ARRAY OF SIZE  N,NBNDB. CONTAINS,
C         ON ENTRY THE COLUMN INDICES OF NON-ZERO ELEMENTS OF THE
C         APPROXIMATE INVERSE  B.
C INFOA : INTEGER ARRAY OF SIZE  (NBNDA-1). CONTAINS ON ENTRY THE
C         DISTANCES OF THE 'NON-ZERO' DIAGONALS.
C B     : TWO DIMENSIONAL REAL ARRAY OF SIZE  N,NBNDB . CONTAINS ON
C         OUTPUT THE NON-ZERO ELEMENTS OF THE APPROXIMATE INVERSE
C         IN THE LOCATIONS GIVEN BY THE ARRAY  INDEXB .
C IFLAG : INTEGER VALUE. IF IFLAG=1 A DIAGONAL-BLOCK APPROXIMATE
C         INVERSE IS FOUND, A LEAST SQUARES APPROXIMATE INVERSE
C         IF IFLAG=0.
      IMPLICIT REAL *8 (A-H,O-Z)
      INTEGER N,NBNDA,NBNDB,IFLAG,ND
      DIMENSION A(N,NBNDA),B(N,NBNDB),INDEXA(N,NBNDA),
     1 INDEXB(N,NBNDB),INFOA(NBNDA-1)
C     REAL MAT(4900,52),RH1(4900),COL(51),X(51)
      REAL *8 MAT(2500,52),RH1(2500),COL(51),X(51)
      INTEGER ICOL(51),IVEC(4900),IX(51)
      DO 10  I = 1,N
      DO 20  J = 1,NBNDB
      B(I,J) = 0.0
20    CONTINUE
10    CONTINUE
      DO 30  J = 1,N
      IZ = 1
      DO  35  I = 1,N
      IVEC(I) = 0
      RH1(I) = 0.0
      DO  36  II = 1,NBNDB
      MAT(I,II) = 0.0
36    CONTINUE
35    CONTINUE
C FIND DIMENSIONS OF THE OVERDETERMINED SYSTEM AND
C THE COLUMNS OF A THAT ARE INVOLVED.
      DO 40  K = 1,NBNDB
      INFB = INDEXB(J,K)
      IF  (  INFB .EQ. 0  ) GOTO 11
      ICOUNT = K
```

```
      IFFO1 = 1
      DO 1000  KK = 1,NBNDA
      IF  (  INDEXA(INFB,KK) .NE. 0  ) GOTO 1111
      IFFO1 = IFFO1 + 1
1000  CONTINUE
1111  CONTINUE
      IFFO2 = NBNDA
      DO 2000  KK = 1,NBNDA
      KKK = NBNDA + 1 - KK
      IF  (  INDEXA(INFB,KKK) .NE. 0  ) GOTO 2222
      IFFO2 = IFFO2 - 1
2000  CONTINUE
2222  CONTINUE
      DO 50  KK = IFFO1,IFFO2
      INFA = INDEXA( INFB,KK )
      DO 60  II = 1,N
      IF  (  INFA .EQ. IVEC( II )  ) GOTO 22
      IF  (  A(INFB,KK) .EQ. 0    ) GOTO 22
60    CONTINUE
      IVEC( IZ ) = INFA
      IZ = IZ + 1
22    CONTINUE
50    CONTINUE
40    CONTINUE
11    CONTINUE
      IDIM = IZ - 1
      IF  (  IFLAG .EQ. 0  ) GOTO 55
C FIND COLUMNS OF A FOR THE QUADRARIC DIAGONAL-BLOCK SYSTEM.
      JZ = ( NBNDB + 1 )/6
C     WRITE(5,*) JZ
C THE NUMBER OF NON-ZERO STRIPES OF THE APPROXIMATE INVERSE
C PLUS 1 IS REQUIRED TO BE DIVISIBLE BY 6.
      IPOINT = 1
      DO 110  I = 1,JZ
      II = JZ + 1 - I
      IS1 = J -( ND - 1 )- II + 1
      IF (  IS1 .LE. 0  ) GOTO 115
      IVEC( IPOINT ) = IS1
      IPOINT = IPOINT + 1
115   CONTINUE
110   CONTINUE
      IF ( JZ .EQ. 1 ) GOTO 120
      DO  125  I = 1,JZ-1
      IS1 = J -( ND - 1 ) + I
      IF (  IS1 .LE. 0  ) GOTO 130
      IVEC( IPOINT ) = IS1
      IPOINT = IPOINT + 1
130   CONTINUE
125   CONTINUE
120   CONTINUE
      DO 135  I = 1,JZ
      II = JZ + 1 - I
      IS1 = J - II
      IF ( IS1 .LE. 0 ) GOTO 140
      IVEC( IPOINT ) = IS1
```

```
      IPOINT = IPOINT + 1
140   CONTINUE
135   CONTINUE
      IVEC( IPOINT ) = J
      IPOINT = IPOINT + 1
      DO 145  I = 1,JZ
      IS1 = J + I
      IF ( IS1 .GT. N ) GOTO 150
      IVEC( IPOINT ) = IS1
      IPOINT = IPOINT + 1
150   CONTINUE
145   CONTINUE
      DO 152  I = 1,JZ
      II = JZ + 1 - I
      IS1 = J + ( ND - 1 ) - II + 1
      IF ( IS1 .GT. N ) GOTO 153
      IVEC( IPOINT ) = IS1
      IPOINT = IPOINT + 1
153   CONTINUE
152   CONTINUE
      IF (  JZ .EQ. 1  ) GOTO 154
      DO 155  I = 1,JZ-1
      IS1 = J + ( ND - 1 ) + I
      IF (  IS1  .GT. N  ) GOTO 156
      IVEC( IPOINT ) = IS1
      IPOINT = IPOINT + 1
156   CONTINUE
155   CONTINUE
154   CONTINUE
      IDIM = ICOUNT
55    CONTINUE
C FIND DIAGONAL BLOCK SYSTEM.
      DO 70  I = 1,IDIM
      IV = IVEC(I)
      CALL COLFETCH( N,NBNDA,INFOA,IV,A,COL,ICOL )
      DO 80  II = 1,ICOUNT
      INFB = INDEXB(J,II)
      DO 90  IJ = 1,NBNDA
      IK = NBNDA + 1 - IJ
      INFA = ICOL(IK)
      IF  (  INFA .EQ. INFB  ) GOTO 33
90    CONTINUE
      GOTO 44
33    CONTINUE
      MAT( I,II ) = COL( IK )
44    CONTINUE
80    CONTINUE
      IF  (  IV .EQ. J  ) RH1(I) = 1.0
70    CONTINUE
C SOLVE DIAGONAL BLOCK SYSTEM
      IF  (  IFLAG .EQ. 1 ) GOTO 111
C     CALL HOUSE( MAT,RH1,IDIM,ICOUNT,X,JFLAG,4900,51 )
      CALL HOUSE( MAT,RH1,IDIM,ICOUNT,X,JFLAG,2500,51 )
      GOTO 222
111   CONTINUE
```

```
      DO  16Ø  I = 1,ICOUNT
      MAT(I,ICOUNT+1) = RH1(I)
16Ø   CONTINUE
C     CALL GAUSS( MAT,X,ICOUNT,ICOUNT+1,49ØØ,51)
      CALL GAUSS( MAT,X,ICOUNT,ICOUNT+1,25ØØ,51)
222   CONTINUE
      DO 1ØØ  I = 1,ICOUNT
      B(J,I) = X(I)
1ØØ   CONTINUE
3Ø    CONTINUE
      RETURN
      END
```

APPENDIX D

FORTRAN SUBROUTINES FOR THE PARALLEL GAUSS-SEIDEL VERSIONS PGS1 AND PGS2 SIMULATED ON A UNIPROCESSOR

This appendix contains FORTRAN subroutines for performing the parallel Gauss-Seidel methods of chapter 2 that serve as a basis for comparison with the algorithms using approximate inverses. The subroutine PGAUSS performs the PGS1 method and the subroutine PGAUS2 performs the PGS2 method. The subroutines UPDATE and UPDAT2 are used in PGAUSS and PGAUS2 respectively.

```
      SUBROUTINE PGAUSS( N,EPS,X,ERR,ICOUNT,XPL,YPL,IPO,IWO,COMP )
C THIS SUBROUTINE COMPUTES ITERATIVELY AN APPROXIMATE SOLUTION
C OF THE LINEAR SYSTEM IN THE ONE DIMENSIONAL CASE.
C A PARALLEL GAUSS-SEIDEL METHOD IS PERFORMED.
C DESCRIPTION OF PARAMETER.
C N      : NUMBER OF UNKNOWNS.
C EPS    : ACCURACY. THE ITERATION IS STOPPED WHEN THE ABSOLUTE ERROR
C          IS LESS THAN  EPS .
C X      : REAL ARRAY OF SIZE  N . ITERATION VECTOR.
C ERR    : REAL VALUE. CONTAINS THE ABSOLUTE ERROR TO
C          THE SOLUTION ON OUTPUT.
C ICOUNT: INTEGER VALUE. ITERATION COUNTER.
C THE FOLLOWING PARAMETERS ARE NOT USED FOR THE ITERATIVE
C PROCESS. THEY ARE USED IN SUBROUTINE  INITPLOT
C FOR PLOTTING THE ERROR.
C XPL    : TWO DIMENSIONAL ARRAY OF SIZE  (N,4). CONTAINS
C          VALUES OF X-AXIS IN COLUMN ONE.
C YPL    : TWO DIMENSIONAL ARRAY OF SIZE  (N,4). CONTAINS
C          VALUES OF Y-AXIS IN COLUMN ONE
C IPO    : INTEGER VALUE. NUMBER OF PLOT-POINTS.
C IWO    : INTEGER VALUE. USED FOR SCALING PURPOSES.
C COMP   : REAL VALUE. USED FOR SCALING PURPOSES.
         INTEGER N,ICOUNT,IPO
         REAL EPS,ERR,NORM,NORME,COMP
         DIMENSION X(N),XPL(61Ø,4),YPL(61Ø,4)
         ICOUNT = Ø
         IPO = 1
         ERR = NORM(X,N)
C        ERR = NORME(X,N)
         CALL INITPLOT(N,ERR,XPL,YPL,IPO,1,COMP)
         IFLA = MOD(N,2)
1        CONTINUE
C FIRST UPDATE SWEEP : 'ODD' COMPONENTS
         K = INT( N/2 )
         IF ( IFLA .EQ. Ø ) K = K - 1
         DO 11  I = Ø,K
         II = 2 * I + 1
         CALL UPDATE( II,X,N )
11       CONTINUE
C END OF FIRST UPDATE SWEEP.
C SECOND UPDATE SWEEP : 'EVEN' COMPONENTS.
         K = INT( N/2 )
         DO 12  I = 1,K
         II = 2 * I
         CALL UPDATE( II,X,N )
12       CONTINUE
C END OF SECOND UPDATE SWEEP.
         ERR = NORM( X,N )
C        ERR = NORME(X,N)
         IF ( IPO .GT. 61Ø ) GOTO 333
         IF ( MOD(ICOUNT,IWO) .NE. Ø ) GOTO 333
         CALL INITPLOT(N,ERR,XPL,YPL,IPO,1,COMP)
333      CONTINUE
         ICOUNT = ICOUNT + 1
```

```
C       IF ( MOD(ICOUNT,5Ø) .EQ. Ø ) WRITE(6,*) ICOUNT
        IF (  ERR .LE. EPS  ) GOTO 99
        IF (  ICOUNT .GE. 1ØØØØ  ) GOTO 99
        GOTO 1
99      CONTINUE
        RETURN
        END
```

```
        SUBROUTINE UPDATE( I,X,N )
C UPDATE FORMULA FOR THE GAUSS-SEIDEL METHOD IN ONE DIMENSION.
C USED IN SUBROUTINE  PGAUSS.
C DESCRIPTION OF PARAMETERS.
C N     : INTEGER VALUE
C I     : INTEGER VALUE. REFERS TO THE COMPONENT
C         OF THE ARRAY  X  TO BE UPDATED.
C X     : REAL ARRAY OF SIZE N. CONTAINS ITERATION VECTOR.
C         ON OUTPUT, THE I'TH COMPONENT IS UPDATED..
        INTEGER I,N
        DIMENSION X(N)
        IF (  I .EQ. 1  ) GOTO 1
        IF (  I .EQ. N  ) GOTO 2
        X(I) = ( X( I-1 ) + X( I+1 ) ) / 2
        GOTO 3
1       CONTINUE
        X(1) = X(2)/2
        GOTO 3
2       CONTINUE
        X(N) = X(N-1)/2
3       CONTINUE
        RETURN
        END
```

```
      SUBROUTINE PGAUS2(N,DIM,EPS,X,ERR,ICOUNT,XPL,YPL,IPO,IWORK,COMP)
C THIS SUBROUTINE COMPUTES ITERATIVELY AN APPROXIMATE SOLUTION
C OF THE LINEAR SYSTEM IN THE TWO DIMENSIONAL CASE.
C A PARALLEL GAUSS-SEIDEL METHOD IS PERFORMED.
C DESCRIPTION OF PARAMETERS.
C DIM    : NUMBER OF GRID POINTS ( NUMBER OF UNKNOWNS ).
C N      : NUMBER OF GRID POINTS PER ROW.
C EPS    : ACCURACY. THE ITERATION IS STOPPED WHEN THE ABSOLUTE ERROR
C          IS LESS THAN  EPS .
C X      : REAL ARRAY OF SIZE  N . ITERATION VECTOR.
C ERR    : REAL VALUE. CONTAINS THE ABSOLUTE ERROR TO
C          THE SOLUTION ON OUTPUT.
C ICOUNT: INTEGER VALUE. ITERATION COUNTER.
C THE FOLLOWING PARAMETERS ARE NOT USED FOR THE ITERATIVE
C PROCESS. THEY ARE USED IN SUBROUTINE  INITPLOT
C FOR PLOTTING THE ERROR FUNCTION.
C XPL    : TWO DIMENSIONAL ARRAY OF SIZE  (N,4). CONTAINS
C          VALUES OF X-AXIS IN COLUMN ONE.
C YPL    : TWO DIMENSIONAL ARRAY OF SIZE  (N,4). CONTAINS
C          VALUES OF Y-AXIS IN COLUMN ONE
C IPO    : INTEGER VALUE. NUMBER OF PLOT-POINTS.
C IWORK  : INTEGER VALUE. USED FOR SCALING PURPOSES.
C COMP   : REAL VALUE. USED FOR SCALING PURPOSES.
         IMPLICIT REAL *8 (A-H,O-Z)
         INTEGER N,ICOUNT,DIM,IPO,IWORK
C        REAL EPS,ERR,NORM,NORME,COMP
C        DIMENSION X(DIM),XPL(61Ø,4),YPL(61Ø,4)
         DIMENSION X(DIM)
         REAL *8 EPS,ERR,NORM,NORME
         REAL XPL(61Ø,4),YPL(61Ø,4),COMP
         ICOUNT = Ø .
         ERR = NORM(X,DIM)
C        ERR = NORME(X,DIM)
         IPO = 1
         CALL INITPLOT(DIM,ERR,XPL,YPL,IPO,1,COMP)
         IFLA = MOD(N,2)
1        CONTINUE
C FIRST UPDATE SWEEP : THE 'ODD' DIAGONALS.
C N IS EXPECTED TO BE GREATER THAN 2.
C BELOW MAIN DIAGONAL.
         K = INT(N/2)
         IF (  IFLA .EQ. Ø  ) K = K-1
         DO 11  I = Ø,K
         II = 2 * I + 1
C        WRITE(6,*) II
         CALL UPDAT2( II,X,N,DIM  )
         JK = II
         DO 12  J = 1,N-1
         JK = JK + N - 1
         JJ = J * N
         IF (  JK .LE. JJ  ) GOTO 13
C        WRITE(6,*) JK
         CALL UPDAT2( JK,X,N,DIM )
```

```
12      CONTINUE
13      CONTINUE
11      CONTINUE
C ABOVE MAIN DIAGONAL.
        IF ( IFLA .EQ. Ø  ) K = K + 1
        DO 14  I = Ø,K-1
        II = 2 * I + 1
        KK = DIM - II + 1
C       WRITE(6,*) KK
        CALL UPDAT2( KK,X,N,DIM )
        JK = KK
        DO 15  J = 1,N-1
        JK = JK - (N-1)
        JJ = (N-J) * N
        IF (  JK .GT. JJ  ) GOTO 16
C       WRITE(6,*) JK
        CALL UPDAT2( JK,X,N,DIM )
15      CONTINUE
16      CONTINUE
14      CONTINUE
C SECOND UPDATE SWEEP : 'EVEN' DIAGONALS.
C BELOW THE MAIN DIAGONAL.
        DO 17  I = 1,K
        II = 2 * I
C       WRITE(6,*) II
        CALL UPDAT2( II,X,N,DIM )
        JK = II
        DO 18  J = 1,N-1
        JK = JK + N - 1
        JJ = J * N
        IF (  JK .LE. JJ  ) GOTO 19
C       WRITE(6,*) JK
        CALL UPDAT2( JK,X,N,DIM )
18      CONTINUE
19      CONTINUE
17      CONTINUE
C ABOVE THE MAIN DIAGONAL.
        IF (  IFLA .EQ. Ø  ) K = K - 1
        DO 11Ø  I = 1,K
        II = 2 * I - 1
        KK = DIM - II
C       WRITE(6,*) KK
        CALL UPDAT2( KK,X,N,DIM )
        JK = KK
        DO 111  J = 1,N-1
        JK = JK - ( N - 1 )
        JJ = ( N - J ) * N
        IF ( JK .GT. JJ ) GOTO 112
C       WRITE(6,*) JK
        CALL UPDAT2( JK,X,N,DIM )
111     CONTINUE
112     CONTINUE
11Ø     CONTINUE
        ICOUNT = ICOUNT + 1
C       IF ( MOD(ICOUNT,5Ø) .EQ. Ø ) WRITE(6,*) ICOUNT
```

```
      ERR = NORM( X,DIM )
C     ERR = NORME(X,DIM)
      IF (MOD(ICOUNT,IWORK).NE.Ø) GOTO 1ØØØ
1ØØØ  CALL INITPLOT(DIM,ERR,XPL,YPL,IPO,1,COMP)
      IF ( ERR .LT. EPS ) GOTO 99
      IF ( ICOUNT .GE. 6ØØØ ) GOTO 99
      GOTO 1
99    CONTINUE
      RETURN
      END
```

```
      SUBROUTINE UPDAT2( I,X,N,DIM )
C UPDATE FORMULA FOR THE GAUSS-SEIDEL METHOD IN TWO DIMENSIONS.
C USED IN SUBROUTINE  PGAUS2.
C DESCRIPTION OF PARAMETERS.
C N     : INTEGER VALUE. NUMBER OF GRID POINTS.
C DIM   : INTEGER VALUE. NUMBER OF GRID POINTS PER ROW.
C I     : INTEGER VALUE. REFERS TO THE COMPONENT
C         OF THE ARRAY X TO BE UPDATED.
C X     : ARRAY OF SIZE  N . CONTAINS ITERATION VECTOR.
C         ON OUTPUT, THE I'TH COMPONENT IS UPDATED.
      IMPLICIT REAL *8 (A-H,O-Z)
      INTEGER I,N,DIM
      DIMENSION X(DIM)
      IF (  I .EQ. 1  ) GOTO 1
      IF (  I .EQ. DIM  ) GOTO 2
      IF (  (I .NE. 1) .AND. (I .LE. N)  ) GOTO 3
      IF (  (I .NE. DIM) .AND. (I .GE. DIM-N ) ) GOTO 4
      X(I) = (  X(I-N) + X(I-1) + X(I+1) + X(I+N)  ) / 4
      GOTO 5
1     CONTINUE
      X(1) = (  X(I+1) + X(I+N)  ) / 4
      GOTO 5
2     CONTINUE
      X(DIM) = (  X(DIM-N) + X(DIM-1)  ) / 4
      GOTO 5
3     CONTINUE
      X(I) = (  X(I-1) + X(I+1) + X(I+N)  ) / 4
      GOTO 5
4     CONTINUE
      X(I) = (  X(I-N) + X(I-1) + X(I+1)  ) / 4
5     CONTINUE
      RETURN
      END
```

APPENDIX E

A FORTRAN SUBROUTINE FOR JACOBI LIKE METHODS USING APPROXIMATE INVERSES SIMULATED ON A UNIPROCESSOR

This appendix contains the FORTRAN subroutine DBLOCK for performing the Jacobi like iterations defined in chapter 3. These methods use a Diagonal-Block or a Least-Squares approximate inverse. Depending on the input parameters the subroutine DBLOCK performs the JDB1(I) , JLSQ1(I) , JDB2(I) or the JLSQ2(I) iterations.

```
      SUBROUTINE DBLOCK(N,NBNDA,NBNDB,A,RH,B,INDEXA,INDEXB,EPS,
     1 ERR,XK,JCOUNT,XPLOT,YPLOT,IP,JPOINT,IWOR,COMP)
C THIS SUBROUTINE PERFORMS THE ITERATION: X(k+1) = X(k)+B(-A X(k)+RH )
C TO SOLVE THE LINEAR SYSTEM  A X = RH.
C ALL COMPONENTS ARE UPDATED WITH THE COMPONENTS OF  X(k) (JACOBI LIKE).
C DESCRIPTION OF PARAMETERS.
C N      : NUMBER OF UNKNOWNS.
C NBNDA  : NUMBER OF 'NON-ZERO' DIAGONALS OF THE COEFFICIENT MATRIX  A.
C NBNDB  : NUMBER OF 'NON-ZERO' DIAGONALS OF THE APPROXIMATE INVERSE B.
C A      : TWO DIMENSIONAL REAL ARRAY OF SIZE  N,NBNDA. CONTAINS ON
C          ENTRY THE NON-ZERO DIAGONALS OF THE COEFFICIENT MATRIX IN
C          ITS COLUMNS.
C RH     : REAL ARRAY OF SIZE  N. CONTAINS ON ENTRY THE RIGHT-HAND
C          SIDE OF THE LINEAR SYSTEM.
C B      : TWO DIMENSIONAL REAL ARRAY OF SIZE  N,NBNDB . CONTAINS ON
C          INPUT THE NON-ZERO ELEMENTS OF THE APPROXIMATE INVERSE
C INDEXA: TWO DIMENSIONAL INTEGER ARRAY OF SIZE  N,NBNDA. CONTAINS
C          THE COLUMN INDICES OF THE ELEMENTS THAT ARE STORED IN  THE
C          MATRIX  A  IN  THE CORRESPONDING LOCATIONS.
C INDEXB: TWO DIMENSIONAL INTEGER ARRAY OF SIZE  N,NBNDB. CONTAINS,
C          ON ENTRY THE COLUMN INDICES OF NON-ZERO ELEMENTS OF THE
C          APPROXIMATE INVERSE  B.
C EPS    : ACCURACY. THE ITERATION IS STOPPED WHEN THE ABSOLUTE ERROR
C          IS LESS THAN  EPS .
C XK     : REAL ARRAY OF SIZE  N . ITERATION VECTOR.
C ERR    : REAL VALUE. CONTAINS THE ABSOLUTE ERROR TO
C          THE SOLUTION ON OUTPUT.
C JCOUNT: INTEGER VALUE. ITERATION COUNTER.
C THE FOLLOWING PARAMETERS ARE NOT USED FOR THE ITERATIVE
C PROCESS. THEY ARE USED IN SUBROUTINE  INITPLOT
C FOR PLOTTING THE ERROR FUNCTION.
C XPLOT : TWO DIMENSIONAL ARRAY OF SIZE  (N,4). CONTAINS
C          VALUES OF X-AXIS IN COLUMN ONE.
C YPLOT : TWO DIMENSIONAL ARRAY OF SIZE  (N,4). CONTAINS
C          VALUES OF Y-AXIS IN COLUMN ONE
C JPOINT: INTEGER VALUE. DETERMINES THE COLUMN OF YPLOT AND XPLOT
C          TO BE INITIALIZED WITH THE PLOT-POINTS..
C IP     : INTEGER VALUE. NUMBER OF PLOT-POINTS.
C IWOR   : INTEGER VALUE. USED FOR SCALING PURPOSES.
C COMP   : REAL VALUE. USED FOR SCALING PURPOSES.
      IMPLICIT REAL *8 (A-H,O-Z)
      INTEGER N,NBNDA,NBNDB,JCOUNT,IP,JPOINT,IWOR
      DIMENSION INDEXA(N,NBNDA),INDEXB(N,NBNDB),A(N,NBNDA),B(N,NBNDB),
     1 RH(N), XK(N)
      REAL YPLOT(610,4),XPLOT(610,4)
C     REAL ERR,NORM,NORME,EPS, SCRAT(4900),COMP
      REAL COMP
      REAL *8 ERR,NORM,NORME,EPS, SCRAT(2500)
      JCOUNT = 0
      IP = 1
      ERR = NORM( XK,N )
C     ERR = NORME(XK,N)
      CALL INITPLOT(N,ERR,XPLOT,YPLOT,IP,JPOINT,COMP)
```

```
C FIND RESIDUAL  B * ( - A * XK  +  RH )
88      DO  1Ø  J = 1,N
        SCRAT(J) = XK(J)
1Ø      CONTINUE
        CALL MATVEC( N,NBNDA,A,INDEXA,XK )
        DO  2Ø  I = 1,N
        XK(I) = - XK(I) + RH(I)
2Ø      CONTINUE
        CALL MATVEC( N,NBNDB,B,INDEXB,XK )
C UPDATE ITERATION VECTOR.
        DO  3Ø  I = 1,N
        XK(I) = SCRAT(I) + XK(I)
        SCRAT(I) = XK(I) - SCRAT(I)
3Ø      CONTINUE
C FIND ERROR.
C       ERR = NORM( SCRAT,N )
        ERR = NORM( XK,N )
C       ERR = NORME(XK,N)
        JCOUNT = JCOUNT + 1
        IF(MOD(JCOUNT,IWOR).EQ.Ø) CALL INITPLOT(N,ERR,XPLOT,YPLOT,
     1  IP,JPOINT,COMP)
C       IF (  MOD(JCOUNT,5Ø) .EQ. Ø ) WRITE(6,*) JCOUNT
        IF  (  ERR .LE. EPS  ) GOTO 99
        IF  (  JCOUNT .GE. 6ØØØ  ) GOTO 99
C       WRITE(6,*) JCOUNT
        GOTO 88
99      CONTINUE
        RETURN
        END
```

APPENDIX F

FORTRAN SUBROUTINES FOR GAUSS-SEIDEL LIKE METHODS USING APPROXIMATE INVERSES SIMULATED ON A UNIPROCESSOR

This appendix contains FORTRAN subroutines for performing the Gauss-Seidel like iterations defined in chapter 3. These subroutines use a Diagonal-Block or a Least-Squares approximate inverse. The subroutine GGS1 performs the GDB1(I) or the GLSQ1(I) iterations for the one dimensional case. The subroutine GGS performs the GDB2(I) and GLSQ2(I) iterations for the two dimensional case.

```
      SUBROUTINE GGS1(N,NBNDA,NBNDB,A,RH,B,INDEXA,INDEXB,EPS,
     1 ERR,XK,JCOUNT,XPLOT,YPLOT,IP,JPOINT,IWOR,COMP )
C THIS SUBROUTINE PERFORMS THE ITERATION:X(k+1)=X(k)+B(- A X(k) + RH)
C TO SOLVE THE LINEAR SYSTEM  A X = RH.
C THERE ARE TWO UPDATE SWEEPS PER ITERATION. THE SET OF COMPONENTS IS
C DIVIDED INTO TWO DISJOINT SUBSETS. THE COMPONENTS WITHIN ONE SUBSET
C CAN BE UPDATED IN PARALLEL.
C DESCRIPTION OF PARAMETERS.
C N      : NUMBER OF GRID POINTS.
C NBNDA  : NUMBER OF 'NON-ZERO' DIAGONALS OF THE COEFFICIENT MATRIX  A.
C NBNDB  : NUMBER OF 'NON-ZERO' DIAGONALS OF THE APPROXIMATE INVERSE B.
C A      : TWO DIMENSIONAL REAL ARRAY OF SIZE  N,NBNDA. CONTAINS ON
C          ENTRY THE NON-ZERO DIAGONALS OF THE COEFFICIENT MATRIX IN
C          ITS COLUMNS.
C RH     : REAL ARRAY OF SIZE  N. CONTAINS ON ENTRY THE RIGHT-HAND
C          SIDE OF THE LINEAR SYSTEM.
C B      : TWO DIMENSIONAL REAL ARRAY OF SIZE  N,NBNDB . CONTAINS ON
C          INPUT THE NON-ZERO ELEMENTS OF THE APPROXIMATE INVERSE
C INDEXA: TWO DIMENSIONAL INTEGER ARRAY OF SIZE  N,NBNDA. CONTAINS
C          THE COLUMN INDICES OF THE ELEMENTS THAT ARE STORED IN  A
C          MATRIX IN  THE CORRESPONDING LOCATIONS.
C INDEXB: TWO DIMENSIONAL INTEGER ARRAY OF SIZE  N,NBNDB. CONTAINS,
C          ON ENTRY THE COLUMN INDICES OF NON-ZERO ELEMENTS OF THE
C          APPROXIMATE INVERSE  B.
C EPS    : ACCURACY. THE ITERATION IS STOPPED WHEN THE ABSOLUTE ERROR
C          IS LESS THAN  EPS .
C XK     : REAL ARRAY OF SIZE  N . ITERATION VECTOR.
C ERR    : REAL VALUE. CONTAINS THE ABSOLUTE ERROR TO
C          THE SOLUTION ON OUTPUT.
C JCOUNT: INTEGER VALUE. ITERATION COUNTER.
C THE FOLLOWING PARAMETERS ARE NOT USED FOR THE ITERATIVE
C PROCESS. THE ARE USED IN SUBROUTINE  INITPLOT
C FOR PLOTTING THEY ERROR FUNCTION.
C XPLOT : TWO DIMENSIONAL ARRAY OF SIZE  (N,4). CONTAINS
C          VALUES OF X-AXIS IN COLUMN  JPOINT .
C YPLOT : TWO DIMENSIONAL ARRAY OF SIZE  (N,4). CONTAINS
C          VALUES OF Y-AXIS IN COLUMN  JPOINT .
C JPOINT: INTEGER VALUE. DETERMINES THE COLUMN OF YPLOT AND XPLOT
C          TO BE INITIALIZED WITH THE PLOT-POINTS..
C IP     : INTEGER VALUE. NUMBER OF PLOT-POINTS.
C IWOR   : INTEGER VALUE. USED FOR SCALING PURPOSES.
C COMP   : REAL VALUE. USED FOR SCALING PURPOSES.
      IMPLICIT REAL *8 (A-H,O-Z)
      INTEGER N,NBNDA,NBNDB,JCOUNT,IP,JPOINT,IWOR
      DIMENSION INDEXA(N,NBNDA),INDEXB(N,NBNDB),A(N,NBNDA),B(N,NBNDB),
     1 RH(N), XK(N)
      REAL YPLOT(610,4),XPLOT(610,4)
C     REAL ERR,NORM,NORME,EPS, SCRAT(4900),COMP
      REAL COMP
      REAL *8 ERR,NORM,NORME,EPS, SCRAT(2500)
      IBOUND = ( NBNDB - 1 )/2 + 1
      JCOUNT = 0
      IP = 1
      ERR = NORM( XK,N )
```

```
C       ERR = NORME(XK,N)
        CALL INITPLOT(N,ERR,XPLOT,YPLOT,IP,JPOINT,COMP)
88      KZEIG = Ø
        JZAEHL = Ø
3       IF  ( JZAEHL .GT. N ) GOTO 1
        DO  1Ø  I = 1,IBOUND
        JZAEHL = KZEIG + I
        IF (  JZAEHL .GT. N ) GOTO 1
        DO  2Ø  J = 1,N
        SCRAT(J) = XK(J)
2Ø      CONTINUE
        CALL AX2MAT(JZAEHL,IBOUND,N,NBNDA,A,INDEXA,SCRAT )
        DO  3Ø  J = 1,N
        SCRAT(J) = - SCRAT(J) + RH(J)
3Ø      CONTINUE
        CALL ZEILVEC( JZAEHL,N,NBNDB,B,INDEXB,SCRAT )
        XK(JZAEHL) = SCRAT(JZAEHL) + XK(JZAEHL)
1Ø      CONTINUE
        KZEIG = KZEIG + ( NBNDB + 1 )
        GOTO 3
1       CONTINUE
        KZEIG = IBOUND
        JZAEHL = Ø
4       IF  (JZAEHL .GT. N) GOTO 2
        DO  4Ø  I = 1,IBOUND
        JZAEHL = KZEIG + I
        IF (  JZAEHL .GT. N ) GOTO 2
        DO  5Ø  J = 1,N
        SCRAT(J) = XK(J)
5Ø      CONTINUE
        CALL AX2MAT(JZAEHL,IBOUND,N,NBNDA,A,INDEXA,SCRAT )
        DO  6Ø  J = 1,N
        SCRAT(J) = - SCRAT(J) + RH(J)
6Ø      CONTINUE
        CALL ZEILVEC( JZAEHL,N,NBNDB,B,INDEXB,SCRAT )
        XK(JZAEHL) = SCRAT(JZAEHL) + XK(JZAEHL)
4Ø      CONTINUE
        KZEIG = KZEIG + ( NBNDB + 1 )
        GOTO 4
2       CONTINUE
        ERR = NORM( XK,N )
C       ERR = NORME(XK,N)
        JCOUNT = JCOUNT + 1
        WRITE(6,*) JCOUNT
        IF(MOD(JCOUNT,IWOR).EQ.Ø) CALL INITPLOT(N,ERR,XPLOT,YPLOT,
     1  IP,JPOINT,COMP)
C       IF (  MOD(JCOUNT,5Ø) .EQ. Ø ) WRITE(6,*) JCOUNT
        IF  (  ERR .LE. EPS  ) GOTO 99
        IF  (  JCOUNT .GE. 6ØØØ  ) GOTO 99
C       WRITE(6,*) JCOUNT
        GOTO 88
99      CONTINUE
        RETURN
        END
```

```
      SUBROUTINE GGS(N,NBNDA,NBNDB,A,RH,B,INDEXA,INDEXB,EPS,
     1 ERR,XK,JCOUNT,XPLOT,YPLOT,IP,JPOINT,IWOR,COMP )
C THIS SUBROUTINE PERFORMS THE ITERATION : X(k+1)=X(k)+B(- A X(k)+RH)
C TO SOLVE THE LINEAR SYSTEM  A X = RH.
C ALL COMPONENTS ARE UPDATED SEQUENTIALLY (GAUSS-SEIDEL LIKE).
C DESCRIPTION OF PARAMETERS.
C N      : NUMBER OF UNKNOWNS.
C NBNDA  : NUMBER OF 'NON-ZERO' DIAGONALS OF THE COEFFICIENT MATRIX  A.
C NBNDB  : NUMBER OF 'NON-ZERO' DIAGONALS OF THE APPROXIMATE INVERSE B.
C A      : TWO DIMENSIONAL REAL ARRAY OF SIZE  N,NBNDA. CONTAINS ON
C          ENTRY THE NON-ZERO DIAGONALS OF THE COEFFICIENT MATRIX IN
C          ITS COLUMNS.
C RH     : REAL ARRAY OF SIZE  N. CONTAINS ON ENTRY THE RIGHT-HAND
C          SIDE OF THE LINEAR SYSTEM.
C B      : TWO DIMENSIONAL REAL ARRAY OF SIZE  N,NBNDB . CONTAINS ON
C          INPUT THE NON-ZERO ELEMENTS OF THE APPROXIMATE INVERSE
C INDEXA: TWO DIMENSIONAL INTEGER ARRAY OF SIZE  N,NBNDA. CONTAINS
C          THE COLUMN INDICES OF THE ELEMENTS THAT ARE STORED IN  A
C          MATRIX IN  THE CORRESPONDING LOCATIONS.
C INDEXB: TWO DIMENSIONAL INTEGER ARRAY OF SIZE  N,NBNDB. CONTAINS,
C          ON ENTRY THE COLUMN INDICES OF NON-ZERO ELEMENTS OF THE
C          APPROXIMATE INVERSE  B.
C EPS    : ACCURACY. THE ITERATION IS STOPPED WHEN THE ABSOLUTE ERROR
C          IS LESS THAN  EPS .
C XK     : REAL ARRAY OF SIZE  N . ITERATION VECTOR.
C ERR    : REAL VALUE. CONTAINS THE ABSOLUTE ERROR TO
C          THE SOLUTION ON OUTPUT.
C JCOUNT: INTEGER VALUE. ITERATION COUNTER.
C THE FOLLOWING PARAMETERS ARE NOT USED FOR THE ITERATIVE
C PROCESS. THEY ARE USED IN SUBROUTINE  INITPLOT
C FOR PLOTTING THE ERROR FUNCTION.
C XPLOT : TWO DIMENSIONAL ARRAY OF SIZE  (N,4). CONTAINS
C          VALUES OF X-AXIS IN COLUMN  JPOINT .
C YPLOT : TWO DIMENSIONAL ARRAY OF SIZE  (N,4). CONTAINS
C          VALUES OF Y-AXIS IN COLUMN  JPOINT .
C JPOINT: INTEGER VALUE. DETERMINES THE COLUMN OF YPLOT AND XPLOT
C          TO BE INITIALIZED WITH THE PLOT-POINTS.
C IP     : INTEGER VALUE. NUMBER OF PLOT-POINTS.
C IWOR   : INTEGER VALUE. USED FOR SCALING PURPOSES.
C COMP   : REAL VALUE. USED FOR SCALING PURPOSES.
      IMPLICIT REAL *8 (A-H,O-Z)
      INTEGER N,NBNDA,NBNDB,JCOUNT,IP,JPOINT,IWOR
      DIMENSION INDEXA(N,NBNDA),INDEXB(N,NBNDB),A(N,NBNDA),B(N,NBNDB),
     1 RH(N), XK(N)
      REAL YPLOT(610,4),XPLOT(610,4)
C     REAL ERR,NORM,NORME,EPS, SCRAT(4900),COMP
      REAL COMP,S
      REAL *8 ERR,NORM,NORME,EPS, SCRAT(2500)
      S = N
      S = SQRT(S)
      ID = INT(S)
      JCOUNT = 0
      IP = 1
```

```
      ERR = NORM( XK,N )
C     ERR = NORME(XK,N)
      CALL INITPLOT(N,ERR,XPLOT,YPLOT,IP,JPOINT,COMP)
88    DO  1Ø  I = 1,N
      DO  2Ø  J = 1,N
      SCRAT(J) = XK(J)
2Ø    CONTINUE
      CALL AX2MAT(I,ID,N,NBNDA,A,INDEXA,SCRAT )
      DO  3Ø  J = 1,N
      SCRAT(J) = - SCRAT(J) + RH(J)
3Ø    CONTINUE
      CALL ZEILVEC( I,N,NBNDB,B,INDEXB,SCRAT )
      XK(I) = SCRAT(I) + XK(I)
1Ø    CONTINUE
      ERR = NORM( XK,N )
C     ERR = NORME(XK,N)
      JCOUNT = JCOUNT + 1
      WRITE(6,*) JCOUNT
      IF(MOD(JCOUNT,IWOR).EQ.Ø) CALL INITPLOT(N,ERR,XPLOT,YPLOT,
     1 IP,JPOINT,COMP)
C     IF (  MOD(JCOUNT,5Ø) .EQ. Ø ) WRITE(6,*) JCOUNT
      IF  (  ERR .LE. EPS  ) GOTO 99
      IF  (  JCOUNT .GE. 6ØØØ  ) GOTO 99
C     WRITE(6,*) JCOUNT
      GOTO 88
99    CONTINUE
      RETURN
      END
```

APPENDIX G

A FORTRAN SUBROUTINE FOR THE PRECONDITIONED CONJUGATE GRADIENT ALGORITHM SIMULATED ON A UNIPROCESSOR

This appendix contains the FORTRAN subroutine PCG that performs the preconditioned conjugate gradient algorithm of chapter 4. The subroutine PCG expects an approximate inverse as an input parameter. If the identity is used as an approximate inverse, the plain conjugate gradient algorithm is performed. The subroutines DOTRPRO and COPY are used by PCG . DOTPRO performs the usual vector inner-product and COPY copies one vector into another.

```
      SUBROUTINE PCG ( N,NBNDA,NBNDB,A,RH,B,INDEXA,INDEXB,EPS,
     1 ERR,XK,JCOUNT,XPLO,YPLO,JPO,IWORK,COMP )
C DESCRIPTION OF PARAMETERS.
C N      : NUMBER OF UNKNOWNS.
C NBNDA  : NUMBER OF 'NON-ZERO' DIAGONALS OF THE COEFFICIENT MATRIX  A.
C NBNDB  : NUMBER OF 'NON-ZERO' DIAGONALS OF THE APPROXIMATE INVERSE B.
C A      : TWO DIMENSIONAL REAL ARRAY OF SIZE  N,NBNDA. CONTAINS ON
C          ENTRY THE NON-ZERO DIAGONALS OF THE COEFFICIENT MATRIX IN
C          ITS COLUMNS.
C RH     : REAL ARRAY OF SIZE  N. CONTAINS ON ENTRY THE RIGHT-HAND
C          SIDE OF THE LINEAR SYSTEM.
C B      : TWO DIMENSIONAL REAL ARRAY OF SIZE  N,NBNDB . CONTAINS ON
C          INPUT THE NON-ZERO ELEMENTS OF THE APPROXIMATE INVERSE
C INDEXA: TWO DIMENSIONAL INTEGER ARRAY OF SIZE  N,NBNDA. CONTAINS
C          THE COLUMN INDICES OF THE ELEMENTS THAT ARE STORED IN  A
C          MATRIX IN  THE CORRESPONDING LOCATIONS.
C INDEXB: TWO DIMENSIONAL INTEGER ARRAY OF SIZE  N,NBNDB. CONTAINS,
C          ON ENTRY THE COLUMN INDICES OF NON-ZERO ELEMENTS OF THE
C          APPROXIMATE INVERSE  B.
C EPS    : ACCURACY. THE ITERATION IS STOPPED WHEN THE ABSOLUTE ERROR
C          IS LESS THAN  EPS .
C XK     : REAL ARRAY OF SIZE  N . ITERATION VECTOR.
C ERR    : REAL VALUE. CONTAINS THE ABSOLUTE ERROR TO
C          THE SOLUTION ON OUTPUT.
C JCOUNT: INTEGER VALUE. ITERATION COUNTER.
C THE FOLLOWING PARAMETERS ARE NOT USED FOR THE ITERATIVE
C PROCESS. THEY ARE USED IN SUBROUTINE  INITPLOT
C FOR PLOTTING THE ERROR FUNCTION.
C XPLO   : TWO DIMENSIONAL ARRAY OF SIZE  (N,4). CONTAINS
C          VALUES OF X-AXIS IN COLUMN  JPO .
C YPLO   : TWO DIMENSIONAL ARRAY OF SIZE  (N,4). CONTAINS
C          VALUES OF Y-AXIS IN COLUMN  JPO .
C JPO    : INTEGER VALUE. DETERMINES THE COLUMN OF YPLO AND XPLO
C          TO BE INITIALIZED WITH THE PLOT-POINTS. IF  IWORK = 1,
C          THE PLAIN CONJUGATE GRADIENT METHOD IS PERFORMED.
C IPO    : INTEGER VALUE. NUMBER OF PLOT-POINTS.
C IWORK  : INTEGER VALUE. USED FOR SCALING PURPOSES.
C COMP   : REAL VALUE. USED FOR SCALING PURPOSES.
         IMPLICIT REAL *8 (A-H,O-Z)
         INTEGER N,NBNDA,NBNDB,JCOUNT,IPO,JPO,IWORK,IW
         DIMENSION INDEXA(N,NBNDA),INDEXB(N,NBNDB),A(N,NBNDA),
     1   B(N,NBNDB),RH(N),XK(N)
         REAL XPLO(610,4),YPLO(610,4)
C        REAL ERR,EPS,NORM,NORME,AI,BI,DOTPRO,COMP
C        REAL RES(4900),SCRATP(4900),SCRATD(4900),
C      1 SHELP(4900),SHELP2(4900),SHELP3(4900),SXK(4900)
         REAL COMP
         REAL *8 ERR,EPS,NORM,NORME,AI,BI,DOTPRO
         REAL *8 RES(4900),SCRATP(2500),SCRATD(2500),SHELP(2500)
         REAL *8 SHELP2(2500)
         REAL *8 SHELP3(2500),SXK(2500)
C        DO 500 I = 1,N
C        WRITE(3,*) ( INDEXB( I,J),J=1,5 )
C500     CONTINUE
```

```
      IF ( JPO .NE. 1 ) GOTO 11
      B(1,1) = 1.Ø
      B(1,2) = Ø.Ø
      B(1,3) = Ø.Ø
      DO 5  I = 2,N
      B(I,1) = Ø.Ø
      B(I,2) = 1.Ø
      B(I,3) = Ø.Ø
5     CONTINUE
11    CONTINUE
      IPO = 1
      IW = Ø
      ERR = NORM(XK,N)
C     ERR = NORME(XK,N)
      CALL INITPLOT(N,ERR,XPLO,YPLO,IPO,JPO,COMP)
      CALL COPY(N,XK,RES)
      CALL MATVEC(N,NBNDA,A,INDEXA,RES)
C     DO 51Ø I = 1,N
C     WRITE(3,*) RES(I)
C51Ø  CONTINUE
      DO  1Ø  I =1,N
      RES(I) = RH(I) - RES(I)
1Ø    CONTINUE
      CALL COPY(N,RES,SCRATD)
C     DO 53Ø I = 1,N
C     WRITE(3,*) ( B(I,J),J=1,5 )
C53Ø  CONTINUE
      CALL MATVEC(N,NBNDB,B,INDEXB,SCRATD)
      CALL COPY(N,SCRATD,SCRATP)
      JCOUNT = Ø
88    CALL COPY(N,SCRATP,SHELP)
      CALL COPY(N,XK,SXK)
      CALL MATVEC(N,NBNDA,A,INDEXA,SCRATP)
      AI = DOTPRO(N,RES,SCRATD)/DOTPRO(N,SCRATP,SHELP)
C     ERR = ABS(AI) * NORM(SHELP,N)
      DO  2Ø  I = 1,N
      XK(I) = XK(I) + AI * SHELP(I)
      SHELP3(I) = RES(I)
      RES(I) = RES(I) - AI * SCRATP(I)
2Ø    CONTINUE
      ERR = NORM(XK,N)
C     ERR = NORME(XK,N)
      CALL COPY(N,SCRATD,SHELP2)
      CALL COPY(N,RES,SCRATD)
      CALL MATVEC(N,NBNDB,B,INDEXB,SCRATD)
      BI = DOTPRO(N,RES,SCRATD)/DOTPRO(N,SHELP3,SHELP2)
      DO  3Ø  I = 1,N
      SCRATP(I) = SCRATD(I) + BI * SHELP(I)
3Ø    CONTINUE
      JCOUNT = JCOUNT + 1
C     IF ( MOD(JCOUNT,5Ø) .EQ. Ø ) WRITE(6,*) ERR
C     IF ( MOD(JCOUNT,5Ø) .EQ. Ø ) WRITE(3,*) ERR
C WORK-SCALING IN ONE DIMENSION FOR PCG WITH DBQ(7) COMPARED WITH CG,
C PCG WITH DBQ(3) AND DBQU(5) .
C     IF ( IW .NE. 3 ) GOTO 22
```

```
C       IF ( MOD(ICOUNT,IWORK) .NE. (IWORK-1) ) GOTO 33
C       CALL INITPLOT(N,ERR,XPLO,YPLO,IPO,JPO)
C       IW = Ø
C       GOTO 33
22      CONTINUE
        IF (MOD( JCOUNT,IWORK) .NE. Ø ) GOTO 33
        CALL INITPLOT(N,ERR,XPLO,YPLO,IPO,JPO,COMP)
C       IF ( JPO .EQ. 4 ) IW = IW + 1
33      CONTINUE
C       IF (  MOD(JCOUNT,5Ø) .EQ. Ø ) WRITE(6,*) JCOUNT
        IF (  ERR  .LE.  EPS  ) GOTO 99
        IF (  JCOUNT  .GE. 6ØØØ  ) GOTO 99
C       IF ( ICOUNT .GT. 1 ) GOTO 333
C       IF ( JPO .EQ. 2 ) COMP = 17/12
C       IF ( JPO .EQ. 3 ) COMP = 23/12
C       IF ( JPO .EQ. 4 ) COMP = 29/12
C333    CONTINUE
        GOTO 88
99      CONTINUE
        RETURN
        END


        REAL FUNCTION DOTPRO(N,VEC1,VEC2)
C THIS SUBROUTINE PERFORMS THE USUAL VECTOR
C INNER-PRODUCT.
C DESCRIPTION OF PARAMETERS.
C N     : SIZE OF THE VECTORS.
C VEC1  : REAL ARRAY OF SIZE  N. FIRST VECTOR.
C VEC2  : REAL ARRAY OF SIZE  N. SECOND VECTOR.
C USED BY SUBROUTINE  PCG.
        IMPLICIT REAL *8 (A-H,O-Z)
        INTEGER N,NORG
        DIMENSION VEC1(N),VEC2(N)
        DOTPRO = Ø.Ø
        DO  1Ø  I = 1,N
        DOTPRO = DOTPRO + VEC1(I) * VEC2(I)
1Ø      CONTINUE
        RETURN
        END


        SUBROUTINE COPY ( N,VEC1,VEC2 )
C THIS SUBROUTINE COPIES TWO VECTORS OF SIZE  N.
C VEC1 IS COPIED INTO VEC2.
C USED BY SUBROUTINE  PCG.
        IMPLICIT REAL *8 (A-H,O-Z)
        INTEGER N
        DIMENSION VEC1(N),VEC2(N)
        DO  1Ø  I =1,N
        VEC2(I) = VEC1(I)
1Ø      CONTINUE
        RETURN
        END
```

APPENDIX H

MISCELLANEOUS SUBROUTINES AND FUNCTIONS

This appendix contains subroutines and functions that are called by the above subroutines. The subroutine HOUSE performs the Householder transformation to solve an overdetermined linear system in the least-squares sense. The function NORM1 is used by subroutine HOUSE . The subroutine GAUSS performs the Gauss elimination process. Both subroutines are called by the subroutines APPRINV and AINV2 . The subroutine INITPLOT is used by the subroutines that perform the iterative processes (see appendix E-F). This subroutine writes the abscissas and ordinates of the error function into a two dimensional array. The functions NORM and NORME comput the maximum norm and the Euclidian norm of a vector.

```
      SUBROUTINE HOUSE(A,B,N,M,X,IFLAG,ORN,ORM)
C THIS SUBROUTINE USES A HOUSEHOLDER TRANSFORMATION
C TO SOLVE AN OVERDETERMINED SYSTEM OF LINEAR EQUATIONS
C IN THE LEAST SQUARES SENSE.
C DESCRIPTION OF PARAMETERS.
C M      NUMBER OF EQUATIONS.
C N      NUMBER OF UNKNOWNS.
C ORN    MAXIMUM NUMBER OF EQUATIONS.
C ORM    MAXIMUM NUMBER OF UNKNOWNS.
C A      TWO DIMENSIONAL ARRAY OF SIZE (M,N).ON ENTRY,
C        THE MATRIX OF THE COEFFICIENTS OF THE OVERDETERMINED
C        SYSTEM MUST BE STORED IN A.ON EXIT,A CONTAINS THE
C        UPPER TRIANGULAR MATRIX RESULTING FROM THE
C        HOUSEHOLDER TRANSFORMATION.
C B      ONE DIMENSIONAL ARRAY OF SIZE (M).ON ENTRY,B MUST
C        CONTAIN THE RIGHT HAND SIDE OF THE EQUATIONS.
C X      ONE DIMENSIONAL ARRAY OF SIZE (N).ON EXIT,
C        X CONTAINS A SOLUTION TO THE PROBLEM.
C IFLAG AN EXIT CODE WITH VALUES ..
C        Ø - NO PROBLEM OCCURRED.
C        1 - ZERO NORM WHILE UPDATING THE ELEMENTARY
C            REFLECTOR MATRIX.
C        2 - ZERO DIVISION OCCURRED IN BACK SUBSTITUTION.
         IMPLICIT REAL *8 (A-H,O-Z)
         INTEGER I,J,K,M,N,II,JJ,ORN,ORM
C        REAL A(ORN,ORM+1),B(ORN),X(ORM)
C        REAL V(8ØØØ),W(6Ø),NORM1,TEM,Y,T,H
         REAL *8 A(ORN,ORM+1),B(ORN),X(ORM)
         REAL *8 V(25ØØ),W(6Ø),NORM1,TEM,Y,T,H
         DO 7Ø J=1,M
         DO 1Ø I=J,N
1Ø       V(I)=A(I,J)
         II=J
C
C        FIND V=(X-Y)/NORM1(X-Y)
C
         Y=NORM1(V,N,II)
         IF (A(J,J).GE.Ø.) V(II)=V(II)+Y
         IF (A(J,J).LT.Ø.) V(II)=V(II)-Y
         TEM=NORM1(V,N,II)
         IF (TEM.EQ.Ø.) GOTO 9Ø
         DO 15 I=J,N
15       V(I)=V(I)/TEM
C
C        FIND A=A-2*V*VT*A
C
         DO 25 JJ=1,M
         W(JJ)=Ø.Ø
         DO 2Ø I=J,N
2Ø       W(JJ)=W(JJ)+2.*V(I)*A(I,JJ)
25       CONTINUE
         DO 4Ø I=J,N
         DO 3Ø JJ=1,M
3Ø       A(I,JJ)=A(I,JJ)-V(I)*W(JJ)
```

```
40      CONTINUE
C
C       FIND B=B-2*V*VT*B
C
        T=0.0
        DO 50 I=J,N
50      T=T+V(I)*B(I)
        DO 60 I=J,N
60      B(I)=B(I)-2.*V(I)*T
70      CONTINUE
C       BACK SUBSTITUTION
C
        MM=M
        X(MM)=B(MM)/A(MM,M)
        DO 80 K=1,MM-1
        I=MM-K
        DO 75 JJ=I+1,M
75      B(I)=B(I)-A(I,JJ)*X(JJ)
C       WRITE(3,*) I
        X(I)=B(I)/A(I,I)
80      CONTINUE
        RETURN
90      IFLAG=1
        RETURN
        END


        REAL *8 FUNCTION NORM1(X,N,J)
C THIS FUNCTION COMPUTES A THE EUCLIDIAN
C VECTOR NORM.
C USED IN SUBROUTINE HOUSE.
C DESCRIPTION OF PARAMETERS.
C X     : REAL ARRAY OF SIZE  N. THE NORM
C         OF A SUBVECTOR OF X IS COMPUTED.
C N     : INTEGER VALUE. SIZE OF VECTOR X.
C J     : INTEGER VALUE. DETERMINES THE
C         SUBVECTOR OF X.
        IMPLICIT REAL *8 (A-H,O-Z)
        REAL *8 X(N),SUM
C       REAL X(N),SUM
        INTEGER I,J,N
        SUM=0.0
        DO 100 I=J,N
100     SUM=SUM+X(I)**2
        NORM1=SQRT(SUM)
        RETURN
        END
```

```
      SUBROUTINE GAUSS(A,X,N,NP1,NN,NNP1)
C  THIS SUBROUTINE PERFORMS THE GAUSS ELIMINATION
C  PROCESS. IT ACCEPTS THE DIMENSIONS OF
C  THE MATRIX AS VARIABLES.
C  DESCRIPTION OF PARAMETERS.
C  A     : TWO DIMENSIONAL ARRAY OF SIZE (NN,NNP1).
C          CONTAINS ON ENTRY THE COEFFICIENT MATRIX
C          OF THE LINEAR SYSTEM IN THE FIRST  N
C          ROWS AND THE FIRST  NP1-1 COLUMNS OF  A
C          AND ALSO THE RIGHT-HAND SIDE IN COLUMN NP1.
C N      : INTEGER VALUE. ACTUAL NUMBER OF ROWS OF A.
C NP1    : INTEGER VALUE. ACTUAL NUMBER OF COLUMNS OF A.
C NN     : INTEGER VALUE. MAXIMUM NUMBER OF ROWS OF A.
C NNP1   : INTEGER VALUE. MAXIMUM NUMBER OF COLUMNS OF A.
      IMPLICIT REAL *8 (A-H,O-Z)
      DIMENSION A(NN,NNP1+1),X(NP1)
C  BEGIN THE PIVOTAL CONDENSATION
C  K NAMES THE PIVOTAL ROW
      NM1=N-1
      DO 600 K=1,NM1
      KP1=K+1
      L=K
C  FIND TERM IN COLUMN K ON OR BELOW MAIN DIAGONAL, THAT
C  IS LARGEST IN ABSOLUTE VALUE.  AFTER THE SEARCH,
C  L IS THE ROW NUMBER OF THE LARGEST ELEMENT
      DO 400 I=KP1,N
400   IF(ABS(A(I,K)).GT.ABS(A(L,K))) L=I
C  CHECK IF L=K WHICH MEANS THAT THE LARGEST ELEMENT IN
C  COLUMN K WAS ALREADY THE DIAGONAL TERM, MAKING
C  ROW INTERCHANGE UNNECESSARY
      IF(L.EQ.K) GO TO 500
C  INTERCHANGE ROWS L AND K, FROM DIAGONAL RIGHT
      DO 410 J=K,NP1
      TEMP=A(K,J)
      A(K,J)=A(L,J)
410   A(L,J)=TEMP
C  ELIMINATE ALL ELEMENTS IN COLUMN K BELOW MAIN DIAGONAL
C  ELEMENTS ELIMINATED ARE NOT ACTUALLY CHANGED
500   DO 600 I=KP1,N
      FACTOR=A(I,K)/A(K,K)
      DO 600 J=KP1,NP1
600   A(I,J)=A(I,J)-FACTOR*A(K,J)
C  BACK SOLUTION
      X(N)=A(N,NP1)/A(N,N)
      I=NM1
710   IP1=I+1
      SUM=0.0
      DO 700 J=IP1,N
700   SUM=SUM+A(I,J)*X(J)
      X(I)=(A(I,NP1)-SUM)/A(I,I)
      I=I-1
      IF(I.GE.1) GO TO 710
      RETURN
      END
```

```
      SUBROUTINE INITPLOT( N,ERR,XPLOT,YPLOT,IPP,I,COMP )
C THIS SUBROUTINE WRITES THE COORDINATES OF A POINT TO BE
C PLOTTED INTO THE ARRAYS  XPLOT  AND  YPLOT.
C DESCRIPTION OF PARAMETERS.
C N      : INTEGER VALUE. NUMBER OF UNKNOWNS.
C ERR    : REAL VALUE. CONTAINS THE ERROR.
C XPLOT  : TWO DIMENSIONAL ARRAY OF SIZE (61Ø,4).
C          CONTAINS ON OUTPUT THE X-VALUE OF THE
C          PLOT-POINT IN POSITION  (IPP,I).
C YPLOT  : TWO DIMENSIONAL ARRAY OF SIZE (61Ø,4).
C          CONTAINS ON OUTPUT THE Y-VALUE OF THE
C          PLOT-POINT IN POSITION  (IPP,I).
C IPP    : INTEGER VALUE (SEE ABOVE).
C I      : INTEGER VALUE (SEE ABOVE).
C COMP   : REAL VALUE. USED FOR SCALING THE X-AXIS.
      INTEGER N,IPP,I
C     REAL ERR,COMP
      REAL *8 ERR
      DIMENSION XPLOT(61Ø,4),YPLOT(61Ø,4)
      REAL COMP,SS
      SS = ERR
      YPLOT(IPP,I) = ALOG( SS )
      XPLOT(IPP,I) = ( IPP - 1 ) * .1 * COMP
      IPP = IPP + 1
      RETURN
      END


      REAL FUNCTION NORME( X,N )
C THIS FUNCTION COMPUTES THE EUCLIDIAN
C NORM OF THE VECTOR  X  OF SIZE  N .
      INTEGER N
      REAL SUM
      DIMENSION X(N)
      SUM = Ø.Ø
      DO 1Ø  I = 1,N
      SUM = SUM + X(I) ** 2
1Ø    CONTINUE
      NORM = SQRT(SUM)
      RETURN
      END
```

```
      REAL FUNCTION  NORM( X,N )
C THIS SUBROUTINE COMPUTES THE MAXIMUM
C NORM OF THE VECTOR  X  OF SIZE  N.
      IMPLICIT REAL *8 (A-H,O-Z)
      INTEGER N
      DIMENSION X( N )
C     REAL SUM
      REAL *8 SUM
      SUM = ABS( X(1) )
      DO  1Ø  J = 2,N
      IF ( SUM .LT. ABS( X(J) ) ) SUM = ABS( X(J) )
1Ø    CONTINUE
      NORM = SUM
      RETURN
      END
```