ITPACKV 2D User's Guide

David R. Kincaid
Thomas C. Oppe
David M. Young

May 1989 CNA-232

Abstract

ITPACKV 2D is an adaptation for vector computers of the ITPACK 2C software package for solving large sparse linear systems of equations by adaptive accelerated iterative algorithms.

This work was supported in part by Cray Research, Inc., through Grant LTR DTD, by the Department of Energy, through Grant DE-FG05-87ER25048, and by the National Science Foundation, through Grant DCR-8518722, with The University of Texas at Austin.

ITPACKV 2D User's Guide *

David R. Kincaid
Thomas C. Oppe
David M. Young †

May 1989

Abstract

ITPACKV 2D is an adaptation for vector computers of the ITPACK 2C software package for solving large sparse linear systems of equations by adaptive accelerated iterative algorithms.

1 Introduction

ITPACKV is a collection of seven FORTRAN subroutines for solving large sparse linear systems by adaptive accelerated iterative algorithms. Basic iterative procedures, such as the Jacobi method, the Successive Overrelaxation method, the Symmetric Successive Overrelaxation method, and the RS method for the reduced system are combined, where possible, with acceleration procedures, such as Chebyshev (Semi-Iteration) and Conjugate Gradient, for rapid convergence. Automatic selection of the acceleration parameters and the use of accurate stopping criteria are major features of this software package. While the ITPACKV routines can be called with any linear system containing positive diagonal elements, they are most successful in solving systems with symmetric positive definite or mildly nonsymmetric coefficient matrices.

ITPACKV 2D is an adaptation for vector computers of the linear equation solution algorithms in the ITPACK 2C software package. See reference [**?**] for more details on the scalar version. The data structure used to contain the coefficient matrix was changed to a format allowing greater efficiency on vector computers while being flexible enough to accommodate matrix problems with general structure. The data structure in the vector version was chosen to coincide with the ELLPACK [**?**] data structure, which can efficiently store matrices arising from discretizations of partial differential equations on general domains.

*This work was supported in part by Cray Research, Inc., through Grant LTR DTD, by the Department of Energy, through Grant DE-FG05-87ER25048, and by the National Science Foundation, through Grant DCR-8518722, with The University of Texas at Austin.

†Center for Numerical Analysis, RLM Bldg. 13.150, University of Texas, Austin, TX 78712

Throughout this paper, we adopt notation such as **SOR()** when referring to a subroutine, **U(*)** when referring to a singly-dimensioned array, and **COEF(*,*)** for a doubly-dimensioned array. The residual vector is $b - Au^{(n)}$ for the linear system $Au = b$ and the pseudo-residual vector is $Gu^{(n)} + k - u^{(n)}$ for a basic iterative method of the form $u^{(n+1)} = Gu^{(n)} + k$. The smallest and largest eigenvalues of the iteration matrix G are denoted $m(G)$ and $M(G)$, respectively.

2 Applicability

The ITPACKV package is designed to solve linear systems of the form $Au = b$ where A is a symmetric and positive definite (or slightly nonsymmetric) system of **N** linear equations, b is the right hand side, and u is the desired solution vector.

The basic solution methods provided are the Jacobi, SOR, Symmetric SOR, and Reduced System methods. All methods except SOR are accelerated by either Conjugate Gradient or Chebyshev acceleration. When using the Reduced System methods, it is required that the system be reordered into a red-black system, that is, a system of the form

$$\begin{bmatrix} D_R & H \\ K & D_B \end{bmatrix},$$

where D_R and D_B are diagonal matrices. A switch to compute, if possible, the red-black indexing, permute the linear system, and permute associated vectors is provided.

3 Sparse Matrix Storage

There are two rectangular arrays, one real and one integer, which are used to store a representation of the coefficient matrix. Both arrays are of size at least **N** by **MAXNZ**, where **N** is the number of linear equations and **MAXNZ** is the maximum number of nonzeros per row in the matrix, with the maximum being taken over all rows. Each row in **COEF(*,*)** will contain the nonzeros of the respective row in the full matrix A, and the corresponding row in **JCOEF(*,*)** will contain its respective column numbers.

For example, the matrix

$$A = \begin{bmatrix} 11. & 0. & 0. & 14. & 15. \\ 0. & 22. & 0. & 0. & 0. \\ 0. & 0. & 33. & 0. & 0. \\ 14. & 0. & 0. & 44. & 45. \\ 15. & 0. & 0. & 45. & 55. \end{bmatrix}$$

would be represented in the **COEF(*,*)** and **JCOEF(*,*)** arrays as

$$\mathbf{COEF}(*,*) = \begin{bmatrix} 11. & 14. & 15. \\ 22. & 0. & 0. \\ 33. & 0. & 0. \\ 44. & 14. & 45. \\ 55. & 15. & 45. \end{bmatrix} \qquad \mathbf{JCOEF}(*,*) = \begin{bmatrix} 1 & 4 & 5 \\ 2 & 0 & 0 \\ 3 & 0 & 0 \\ 4 & 1 & 5 \\ 5 & 1 & 4 \end{bmatrix}$$

There are several remarks which should be made:

1. If a row in the matrix A has fewer than **MAXNZ** nonzeros, the corresponding rows in **COEF(*,*)** and **JCOEF(*,*)** should be padded with zeros.
2. The nonzero entries in a given row of **COEF(*,*)** may appear in any order. However, if the diagonal element is not in column 1, then ITPACKV will place it there without returning it to its original position upon exiting.
3. The diagonal element of each row should be positive. If a diagonal element is negative, ITPACKV will reverse the signs of all entries corresponding to this equation. Since this may result in a loss of symmetry of the system, convergence may be adversely affected. Hence it is better if the user enters the matrix with positive diagonal elements.
4. In ITPACKV 2D, all nonzero matrix entries must be present even if the matrix is symmetric. It does not suffice to store just the upper or lower triangle of the matrix A.

4 Usage

The user is expected to provide the coefficient matrix, the right hand side, an initial guess to the solution, and various integer and real workspace arrays.

The calling sequence is

CALL ⟨ *method* ⟩ **(N, NDIM, MAXNZ, JCOEF, COEF, RHS, U, IWKSP, NW, WKSP, IPARM, RPARM, IER)**

where ⟨ *method* ⟩ is one of

JCG	Jacobi Conjugate Gradient
JSI	Jacobi Semi-Iteration
SOR	Successive Overrelaxation
SSORCG	Symmetric SOR Conjugate Gradient
SSORSI	Symmetric SOR Semi-Iteration
RSCG	Reduced System Conjugate Gradient
RSSI	Reduced System Semi-Iteration

and where the remaining parameters are defined in the following. Here "input" means that the subroutine expects the user to provide the necessary input data and "output" means that the routine passes back information in the variable or array indicated. All parameters are arrays except variables **N**, **NDIM**, **MAXNZ**, **NW**, and **IER**. Moreover, all parameters may be altered by the subroutine call except variables **N**, **NDIM**, and **NW**. (See Section **??** for additional details.)

N is the order of the linear system. [integer; input]

NDIM is the row dimension of the arrays **COEF(*,*)** and **JCOEF(*,*)** in the routine first defining them. Note that **NDIM** $\geq$ **N**. [integer; input]

MAXNZ is the maximum number of nonzero elements per row over all rows in the matrix A. [integer; input]

JCOEF(*,*) is a rectangular array of dimension **NDIM** by **MAXNZ** containing the column numbers of the matrix coefficients given in the corresponding locations in **COEF(*,*)**. [integer array; input]

COEF(*,*) is a rectangular array of dimension **NDIM** by **MAXNZ** containing the nonzero matrix coefficients. [real array; input]

RHS(*) is a vector of length **N** containing the right-hand side of the linear system. [real array; input]

U(*) is a vector of length **N** containing the initial guess to the solution of the linear system on input and the latest approximate solution on output. Note that **U(*)** may be filled with zeros if no initial guess is known by making a call to **VFILL()** prior to calling an ITPACKV solution module. See Section **??** for details. [real array; input/output]

IWKSP(*) is a vector of length $\mathbf{3} * \mathbf{N}$ used for integer workspace. When reindexing for red-black ordering, the first **N** locations contain on output the permutation vector for the red-black indexing, the next **N** locations contain its inverse, and the last **N** are used for integer workspace.[1] [integer array; output]

NW is a scalar. On input, **NW** is the available length for **WKSP(*)**. On output, **IPARM(8)** is the actual amount used (or needed). [integer; input]

WKSP(*) is a vector used for real working space whose length depends on the iterative method being used. It must be at least **NW** entries long. (See Section **??** for the required amount of workspace for each method.) [real array]

IPARM(*) is a vector of length 12 used to initialize various integer and logical parameters. Default values may be set by calling subroutine **DFAULT()** described in Section **??**. On output, **IPARM(*)** contains the values of the parameters that were changed. (Further details are given later in Section **??**.) [integer array; input/output]

RPARM(*) is a vector of length 12 used to initialize various real parameters on input. Default values may be set by calling subroutine **DFAULT()** described in Section **??**. On output, **RPARM(*)** contains the final values of the parameters that were changed. (Further details are given later in Section **??**.) [real array; input/output]

[1] For the red-black ordering, the **I**th entry of a permutation array **P(*)** indicates the position **J** into which the **I**th unknown of the original system is being mapped, that is, if $\mathbf{P}(\mathbf{I}) = \mathbf{J}$ then unknown **I** is mapped into position **J**. The **J**th entry of an inverse permutation array **IP(*)** indicates the position **I** into which the **J**th unknown of the permuted system must be mapped to regain the original ordering, that is, $\mathbf{IP}(\mathbf{J}) = \mathbf{I}$.

IER is the error flag which is set to zero for normal convergence and to a nonzero integer when an error condition is present. (See Section **??** for the meaning of nonzero values.) [integer; output]

5 Parameter Arrays

The user must call subroutine **DFAULT()** prior to calling one of the ITPACKV iterative solution modules. The routine **DFAULT()** fills parameter arrays **IPARM(*)** and **RPARM(*)** with default values and has the syntax

CALL DFAULT (IPARM, RPARM)

The user may supply nondefault values for selected quantities in **IPARM(*)** and **RPARM(*)** by first calling **DFAULT()** and then assigning the appropriate nondefault values before calling a solution module of ITPACKV. Important variables in this package which may change adaptively are **CME** (estimate of $M(B)$, the largest eigenvalue of the Jacobi matrix), **SME** (estimate of $m(B)$, the smallest eigenvalue of the Jacobi matrix), **OMEGA** (overrelaxation parameter for the SOR and SSOR methods), **SPECR** (estimate of the spectral radius of the SSOR matrix), **BETAB** (estimate of the spectral radius of the matrix LU, where L and U are strictly lower and upper triangular matrices, respectively, such that the Jacobi matrix $B = L + U$).

The integer array **IPARM(*)** and the real array **RPARM(*)** allow the user to control certain parameters which affect the performance of the iterative algorithms. Furthermore, these arrays allow the updated parameters from the automatic adaptive procedures to be communicated back to the user. The entries in **IPARM(*)** and **RPARM(*)** are

IPARM(1) **ITMAX** is the maximum number of iterations allowed. It is reset on output to the number of iterations performed. Default: 100

IPARM(2) **LEVEL** is used to control the level of output. Each higher value provides additional information. Default: 0

[< 0: no output on unit **IPARM(4)**;
0: fatal error messages only;
1: warning messages and minimum output;
2: reasonable summary (progress of algorithm);
3: parameter values and informative comments;
4: approximate solution after each iteration (primarily useful for debugging);
5: original system]

IPARM(3) **IRESET** is the communication switch. Default: 0

[0: implies certain values of **IPARM(*)** and **RPARM(*)** will be overwritten to communicate parameters back to the user;
$\neq 0$: only **IPARM(1)** and **IPARM(8)** will be reset.]

IPARM(4) NOUT is the output unit number. Default: 6

IPARM(5) ISYM is the symmetric matrix switch. Default: 0

[0: the matrix is symmetric;
1: the matrix is nonsymmetric]

IPARM(6) IADAPT is the adaptive switch. It determines whether certain parameters have been set by the user or should be computed automatically in either a fully or partially adaptive sense. Default: 1

[0: fixed iterative parameters used for **SME**, **CME**, **OMEGA**, **SPECR**, and **BETAB** (nonadaptive);
1: fully adaptive procedures used for all parameters;
2: (SSOR methods only) **SPECR** determined adaptively and **CME**, **BETAB**, and **OMEGA** fixed;
3: (SSOR methods only) **BETAB** fixed and all other parameters determined adaptively]

(See [**?**, **?**] for details and $\mathbf{RPARM(I)}, \mathbf{I = 2, 3, 5, 6, 7}$ for **CME**, **SME**, **OMEGA**, **SPECR**, **BETAB**, respectively. These parameters are set by subroutine **DFAULT()** or by the user.)

IPARM(7) ICASE is the adaptive procedure case switch for the JSI and SSOR methods. There are two strategies, called Case I and Case II, for doing the adaptive procedure. The choice of which case to select corresponds to knowledge of the eigenvalues of the Jacobi matrix B and their estimates. Default: 1

[$\neq 2$ Case I: Fixed **SME** $\leq m(B)$ (general case);
$= 2$ Case II: Use when it is known that $|m(B)| \leq M(B)$]

The case switch determines how the estimates for **SME** and **CME** are recomputed adaptively. In Case I, **SME** is fixed throughout and should be less than or equal to $m(B)$. In Case II, **SME** is set to $-$**CME** which may adaptively change. As far as the adaptive procedure is concerned, Case I is the most general case and should be specified in the absence of specific knowledge of the relationship between the eigenvalues and their estimates. An example when Case II is appropriate occurs when the Jacobi matrix has Property A, since $m(B) = -M(B)$.[2] Also, if A is an L-matrix, then for the Jacobi matrix, we have $|m(B)| \leq M(B)$ and **SME** is always $-$**CME** (Case II). [3] Selecting the correct case may increase the rate of convergence of the iterative method. (See [**?**] for additional discussion on Case I and II. Also, see $\mathbf{RPARM(I)}, \mathbf{I = 2, 3}$ for **CME**, **SME**, respectively.)

IPARM(8) NWKSP is the amount of workspace used. It is used for output only. If **ITMAX** is set to a value just over the actual number of iterations necessary for convergence, the amount of memory for **WKSP(*)** can be reduced to just over the

[2] A matrix has Property A if and only if it is a diagonal matrix or else there exists a rearrangement of the rows and corresponding columns of the matrix which corresponds to a red-black partitioning.

[3] An L-matrix has positive diagonal elements and nonpositive off-diagonal elements.

value returned here. This may be done when rerunning a problem, for example. Default: 0

IPARM(9) **NB** is the red-black ordering switch. On output, if reindexing is done, **NB** is set to the order of the black subsystem. Default: -2

-2: skip indexing—system already in desired form and is not red-black.
-1: compute red-black indexing and permute system;
≥ 0: skip indexing—system already in red-black form, with **NB** as the order of the black subsystem.

IPARM(10) **IREMOVE** is the switch for effectively removing rows and columns when the diagonal entry is extremely large compared to the nonzero off-diagonal entries in that row. (See **RPARM(8)** for additional details.) Default: 0

[0: not done; $\neq 0$: test done]

IPARM(11) **ITIME** is the timing switch. Default: 0

[0: time method; $\neq 0$: not done]

IPARM(12) **IDGTS** is the error analysis switch. It determines if an analysis is done to determine the accuracy of the final computed solution. Default: 0

[< 0: skip error analysis
0: compute **DIGIT1** and **DIGIT2** and store in $\mathbf{RPARM}(\mathbf{I}), \mathbf{I} = \mathbf{11}, \mathbf{12}$, respectively;
1: print **DIGIT1** and **DIGIT2**;
2: print final approximate solution vector;
3: print final approximate residual vector;
4: print both solution and residual vectors;
otherwise: no printing]

(If **LEVEL** ≤ 0, no printing is done. See $\mathbf{RPARM}(\mathbf{I}), \mathbf{I} = \mathbf{11}, \mathbf{12}$ for details on **DIGIT1** and **DIGIT2**.)

RPARM(1) **ZETA** is the stopping criterion or approximate relative accuracy desired in the final computed solution. If the method did not converge in **IPARM(1)** iterations, **RPARM(1)** is reset to an estimate of the relative accuracy achieved. The stopping criterion is a test of whether **ZETA** is greater than the ratio of the two norm of the pseudo-residual vector and the two norm of the current iteration vector times a constant involving an eigenvalue estimate. (See [**?**, **?**] for details.) Default: 5×10^{-6}

RPARM(2) **CME** is the estimate of the largest eigenvalue of the Jacobi matrix. It changes to a new estimate if the adaptive procedure is used. **CME** $\leq M(B)$. Default: 0.0

RPARM(3) **SME** is the estimate of the smallest eigenvalue of the Jacobi matrix for the JSI method. In Case I, **SME** is fixed throughout at a value $\leq m(B)$. In Case II, **SME** is always set to $-$**CME** with **CME** changing in the adaptive procedure. (See **IPARM(7)** for definitions of Case I and II.) Default: 0.0

RPARM(4) FF is the adaptive procedure damping factor. Its values are in the interval $(0., 1.]$ with 1. causing the most frequent parameter changes when the fully adaptive switch **IPARM**(**6**) = **1** is used. Default: 0.75

RPARM(5) OMEGA is the overrelaxation parameter for the SOR and the SSOR methods. If the method is fully adaptive, **OMEGA** changes. Default: 1.0

RPARM(6) SPECR is the estimated spectral radius for the SSOR matrix. If the method is adaptive, **SPECR** changes. Default: 0.0

RPARM(7) BETAB is the estimate for the spectral radius of the matrix LU used in the SSOR methods. **BETAB** may change depending on the adaptive switch **IPARM(6)**. The matrix L is the strictly lower triangular part of the Jacobi matrix and U is the strictly upper triangular part. When the spectral radius of LU is less than or equal to $\frac{1}{4}$, the "SSOR condition" is satisfied for some problems provided one uses the natural ordering. (See [**?**, **?**] for additional details.) Default: 0.25.

RPARM(8) TOL is the tolerance factor near machine relative precision, **SRELPR**. In each row, if all nonzero off-diagonal row entries are less than **TOL** times the value of the diagonal entry, then this row and corresponding column are essentially removed from the system. This is done by setting the nonzero off-diagonal elements in the row and corresponding column to zero, replacing the diagonal element with 1, and adjusting the elements on the right-hand side of the system so that the new system is equivalent to to the original one. [4] If the diagonal entry is the only nonzero element in a row and is not greater than the reciprocal of **TOL**, then no elimination is done. This procedure is useful for linear systems arising from finite element discretizations of PDEs in which Dirichlet boundary conditions are handled by giving the diagonal values in the linear system extremely large values. (The installer of this package should set the value of **SRELPR**. See comments in subroutine **DFAULT()** and Section **??** for additional details.) Default: 100. × **SRELPR**

RPARM(9) TIME1 is the total time in seconds from the beginning of the iterative algorithm until convergence. (A machine dependent subprogram call for returning the time in seconds is provided by the installer of this package.) Default: 0.0

RPARM(10) TIME2 is the total time in seconds for the entire call. Default: 0.0

RPARM(11) DIGIT1 is the approximate number of digits using the estimated relative error with the final approximate solution. It is computed as the negative of the logarithm base ten of the final value of the stopping test. (See details below or [**?**].) Default: 0.0

RPARM(12) DIGIT2 is the approximate number of digits using the estimated relative residual with the final approximate solution. It is computed as the negative of the

[4] If the row and column corresponding to diagonal entry $A_{i,i}$ are to be eliminated, then the right-hand side is adjusted to $b_i \leftarrow b_i/A_{i,i}$ and $b_j \leftarrow b_j - b_i A_{i,j}$ for $j \neq i$.

logarithm base ten of the ratio of the two norm of the residual vector and the two norm of the right-hand side vector. This estimate is related to the condition number of the original linear system and, therefore, it will not be accurate if the system is ill-conditioned. (See details below or [**?**].) Default: 0.0

DIGIT1 is determined from the actual stopping test computed on the final iteration, whereas **DIGIT2** is based on the computed residual vector using the final approximate solution after the algorithm has converged. If these values differ greatly, then either the stopping test has not worked successfully or the original system is ill-conditioned. (See [**?**] for additional details.)

6 Workspace Requirements

For storage of certain intermediate results, the solution modules require a real vector **WKSP(*)** and a corresponding variable **NW** indicating the available space. The length of the workspace array varies with each solution module and the maximum amount needed is given in the following table.

Solution Module	Maximum Length of **WKSP(*)**
JCG()	$\mathbf{4 * N + 4 * ITMAX}$
JSI()	$\mathbf{2 * N}$
SOR()	$\mathbf{2 * N}$
SSORCG()	$\mathbf{6 * N + 4 * ITMAX}$
SSORSI()	$\mathbf{5 * N}$
RSCG()	$\mathbf{N + 3 * NB + 4 * ITMAX}$
RSSI()	$\mathbf{N + NB}$

where **N** is the number of linear equations, **NB** is the number of black grid points in the red-black ordering, and **ITMAX** is the maximum number of iterations allowed. The Conjugate Gradient routines require $\mathbf{4 * ITMAX}$ extra storage locations for eigenvalue calculations. It should be noted that the actual amount of workspace used may be somewhat less than these upper limits since some of the latter are dependent on the maximum number of iterations allowed, **ITMAX**, stored in **IPARM(1)**. Clearly, the array **WKSP(*)** must be dimensioned to at least the value of **NW**.

7 Error Conditions

The ITPACKV modules assign to the error flag **IER** certain values to indicate error conditions (or lack of them) upon exiting. Nonzero integer values of the error flag indicate that an error condition was detected. The values it can be assigned and the corresponding meanings are indicated in the following table.

Error Flag		Meaning
IER =	0,	Normal convergence was obtained.
=	$1+M$,	Invalid order of the system, **N**.
=	$2+M$,	Workspace array **WKSP(*)** is not large enough. **IPARM(8)** is set to the amount of required workspace, **NW**.
=	$3+M$,	Failure to converge in **IPARM(1)** iterations. **RPARM(1)** is reset to the last stopping value computed.
=	$4+M$,	Invalid order of the black subsystem, **NB**.
=	201,	Red-black indexing is not possible.
=	401,	There is a zero diagonal element.
=	402,	No diagonal element in a row.
=	501,	Failure to converge in **ITMAX** function evaluations.
=	502,	Function does not change sign at the endpoints.
=	601,	Successive iterates are not monotone increasing.
=	602,	The matrix is not positive definite.

JCG(), **JSI()**, **SOR()**, **SSORCG()**, **SSORSI()**, **RSCG()**, **RSSI()** assign values to M of 10, 20, 30, 40, 50, 60, 70, respectively. **PRBNDX()**, **SCAL()**, **ZBRENT()**, **EQRT1S()** are subroutines with error flags in the 200's, 400's, 500's, 600's, respectively. These routines perform the following functions: **PRBNDX()** determines the red-black indexing, **SCAL()** scales the system, **ZBRENT()** is a modified IMSL routine for computing a zero of a function which changes sign in a given interval, **EQRT1S()** is a modified IMSL routine for computing the largest eigenvalue of a symmetric tridiagonal matrix.[5]

8 Use of Subroutine VFILL

The array **U(*)** should contain an initial approximation to the solution of the linear system before any ITPACKV module is called. If the user has no information for making such a guess, then the zero vector may be used as the starting vector. The subroutine **VFILL()** can be used to fill a vector with a constant. For example,

CALL VFILL (N, U, VAL)

fills the array **U(*)** of length **N** with the value **VAL** in each entry.

9 Notes on Use

Each of the ITPACKV solution modules scales the linear system to a unit diagonal system prior to iterating and unscales it upon termination. This reduces the number of arithmetic operations, but it may introduce small changes in the coefficient matrix **COEF(*,*)** and

[5]IMSL (International Mathematical and Statistical Libraries, Inc.), Sixth Floor NBC Bldg., 7500 Bellaire Blvd., Houston, TX, 77036.

the right hand side vector **RHS(*)** due to round-off errors in the computer arithmetic. The scaling involves the diagonal matrix $D^{\frac{1}{2}}$ of square roots of the diagonal entries of the linear system, that is,

$$(D^{-\frac{1}{2}}AD^{-\frac{1}{2}})(D^{\frac{1}{2}}u) = (D^{-\frac{1}{2}}b).$$

The algorithms iterate until convergence is reached based on the relative accuracy requested via the stopping criterion set in **RPARM(1)** for the scaled solution vector $(D^{\frac{1}{2}}u)$. Unscaling solves for u and returns the linear system to its original form subject to roundoff errors in the arithmetic.

When requested, a red-black permutation of the linear system will be done before and after the iterative algorithm is called. Otherwise, the linear system is used in the order it is given.

If **SOR()**, **SSORCG()**, or **SSORSI()** is called, ITPACKV will attempt to segregate the parts of **COEF(*,*)** and **JCOEF(*,*)** corresponding to the upper and lower triangular parts of the matrix A into separate columns. Since this will in general take more storage, it may not be possible if **MAXNZ** (the column dimension of **COEF(*,*)** and **JCOEF(*,*)**) is too small. In this case the algorithms will operate on the existing data structure, but performance will be degraded. Segregation of L and U increases the vectorizability of these algorithms. In the case that red-black ordering is used with **SOR()**, **SSORCG()**, or **SSORSI()**, then no attempt to segregate L and U is made.

The Successive Overrelaxation (SOR) method has been shown to be more effective with the red-black ordering than with the natural ordering for some problems. In the SOR algorithm, the first iteration uses $\omega = 1$ and the stopping criterion is set to a large value so that at least one Gauss-Seidel iteration is performed before an approximate value for the optimum relaxation parameter is computed.

Optional features of this package are red-black ordering, effective removal of rows and columns when the diagonal entry is extremely large, and error analysis. In the event that one is not using some of these options and needs additional memory space for a very large linear system, the relevant subroutines which can be replaced with dummy subroutines are as follows: red-black ordering [**PRBNDX()**, **PERMAT()**, **PERVEC()**], removal of rows [**SBELM()**], error analysis [**PERROR()**].

The iterative algorithms used in ITPACKV are quite complicated and some knowledge of iterative methods is necessary to completely understand them. The interested reader should consult the references for details of the algorithms.

10 Installation Guide

This package is written in 1977 ANSI standard FORTRAN in the interests of portability. However, the following changes to the code should be considered when transporting the code to another computer.

1. This version of ITPACK was written to be efficient on vector computers having a vector cache or vector registers and whose compilers can vectorize gather/scatter programming constructs. For maximum efficiency on Cray vector computers, the FOR-

TRAN routines **ISMIN()**, **WHENIGE()**, **WHENILT()**, **SAXPY()**, **SCOPY()**, **SDOT()**, and **SNRM2()** should be deleted from the package since optimized CAL versions of these routines exist. Also, in routine **DETERM()**, loop 10 should be deleted and the line containing the call to **SOLRN()** should be activated.

2. The timing routine **TIMER()** uses the routine **SECOND()**. Since there is no standard timing routine in FORTRAN, this subroutine may have to be modified. Also, many computers have more accurate timing facilities than **SECOND()**.

3. The value of the machine relative precision is contained in the variable **SRELPR**, which is set in the **DFAULT()** subroutine. This and other default values may be permanently changed when the code is installed by changing their values in this subroutine. In particular, **SRELPR** must be changed for different computers. If the installer of this package does not know its value, an approximate value can be determined from a simple FORTRAN program given in the comment statements of subroutine **DFAULT()**.

4. Since the amount of precision may change from computer to computer, the relative accuracy requested in the stopping criterion **ZETA** must not be less than about 500 times the machine relative precision **SRELPR**. If a value of **ZETA** is requested that is too small, then the code resets it to this value. The current default value for **ZETA**, 5×10^{-6}, is set by the routine **DFAULT()** into **RPARM(1)**.

5. The program line of the ITPACKV testing program may need to be changed since different computers have different conventions for opening files.

The testing program and the routines **DFAULT()** and **TIMER()** are the only routines which require editing by the installer of the package. It is suggested that ITPACKV be made into a compiled program library although not all of it would normally be used in a particular application.

11 Changes from ITPACKV 2C

This version of ITPACKV is an update of an earlier version, ITPACKV 2C. In this version, the **SOR()**, **SSORCG()**, and **SSORSI()** modules have been modified so they are vectorizable in the case of natural ordering. These modules permute the system $Au = b$ according to a "wave front" ordering prior to iterating and unpermute the system after iterating. The wave front ordering is determined for these routines by finding the sets of unknowns that can be updated simultaneously (e.g., the unknowns along the diagonals of the mesh for a 5-point star operator on a rectangular grid). The vector lengths of these wave fronts are dependent upon the sparsity pattern of the matrix as well as the size of the matrix, but are typically much smaller than **N**. This technique of vectorization will be effective on vector computers which do not have large start-up costs for vector operations.

Also, for the **SOR()**, **SSORCG()**, and **SSORSI()** modules, the matrix is scaled by ω to reduce the operation count during the forward and backward SOR passes. The scaling

process is done every time ω is changed. Finally, the routine **DETERM()** was rewritten to use the Cray library routine **SOLRN()**, if it is available. See [**?**] for details on the vectorization techniques used in ITPACKV 2D.

References

[1] R. G. Grimes, D. R. Kincaid, and D. M. Young. "ITPACK 2.0 User's Guide." CNA-150, Center for Numerical Analysis, University of Texas, Austin, Texas, 78712, August 1979.

[2] L. A. Hageman and D. M. Young. *Applied Iterative Methods.* New York: Academic Press, 1981.

[3] D. R. Kincaid, T. Oppe, and D. M. Young. "Adapting ITPACK Routines for Use on a Vector Computer." CNA-177, Center for Numerical Analysis, University of Texas, Austin, Texas, 78712, August, 1982.

[4] D. R. Kincaid and T. Oppe. "ITPACK on Supercomputers." in *Numerical Methods* (Lecture Notes in Mathematics 1005). A. Dold and B. Eckmann (eds.), Springer-Verlag, New York, 1983, pp. 151-161.

[5] D. R. Kincaid, T. C. Oppe, and D. M. Young. "Vector Computations for Sparse Linear Systems." CNA-189, Center for Numerical Analysis, University of Texas, Austin, Texas, 78712, February, 1984.

[6] D. R. Kincaid, J. R. Respess, D. M. Young, and R. G. Grimes. "ITPACK 2C: A FORTRAN Package for Solving Large Sparse Linear Systems by Adaptive Accelerated Iterative Methods." *ACM Transactions on Mathematical Software*, Vol. 8, No. 3, 1982.

[7] D. R. Kincaid and D. M. Young. "Survey of Iterative Methods," in *Encyclopedia of Computer Sciences and Technology*, Vol. 13 (J. Belzer, A. Holzman, and A. Kent, eds.), Marcel Dekker Inc., New York, 1979, pp. 354-391.

[8] D. R. Kincaid and D. M. Young. "The ITPACK Project: Past, Present, and Future." CNA-180, Center for Numerical Analysis, University of Texas, Austin, Texas, 78712, March 1983.

[9] T. C. Oppe. "The Vectorization of ITPACK 2C." To be published in *International Journal for Numerical Methods in Engineering.*

[10] J. Rice and R. Boisvert. *Solving Elliptic Problems Using ELLPACK.* New York: Springer-Verlag, 1985.

[11] D. M. Young. *Iterative Solution of Large Linear Systems.* New York: Academic Press, 1971.