ITPACK 2C: A FORTRAN Package for Solving Large Sparse Linear Systems by Adaptive Accelerated Iterative Methods [*]

David R. Kincaid, John R. Respess, and David M. Young
University of Texas at Austin [†]

Roger G. Grimes
Boeing Computer Services Company [‡]

April 2, 2002

Abstract

ITPACK 2C 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 ITPACK routines can be called with any linear system containing positive diagonal elements, they are the most successful in solving systems with symmetric positive definite or mildly nonsymmetric coefficient matrices.

1 Introduction

For several years, we have been involved with the development and use of research-oriented programs using iterative algorithms for solving large sparse linear systems

$$Au = b$$

with positive diagonal elements. One solves for the $\mathbf{N}$ component unknown vector u given the $\mathbf{N} \times \mathbf{N}$ nonsingular coefficient matrix A and the $\mathbf{N}$ component right-hand side vector b. The

[*]Work on this project was supported in part by National Science Foundation Grant MCS 79-19829 at The University of Texas at Austin.

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

[‡]Boeing Computer Services Company, 565 Andover Park West, Tukwila, WA 98067

current ITPACK software package of subroutines, version 2C, provides for the use of seven alternative iterative procedures. While these subroutines are not designed as production software, they should successfully handle industrial problems of moderate size, that is, ones that fit in high-speed memory. This package is written in standard FORTRAN-66 code. It has been tested over a wide variety of computer systems using various FORTRAN compilers, including one which is FORTRAN-77 compatible (see Acknowledgements).

The seven iterative solution modules are based on several basic iterative procedures, such as the Jacobi method, the Successive Overrelaxation (SOR) method, the Symmetric SOR (SSOR) method, and the RS method for the reduced system. With the exception of SOR, the convergence of these basic methods are accelerated by Chebyshev (Semi-Iteration, SI) or Conjugate Gradient (CG) acceleration. All methods are available with adaptive parameter estimation and automatic stopping tests. When using the RS method it is required that the linear system be reordered into a "red-black"[1] system [6, 12]. A switch to compute, if possible, the red-black indexing, permute the linear system, and permute associated vectors is provided.

The successful convergence of iterative methods may be dependent on conditions that are difficult to determine in advance. For example, determining whether the coefficient matrix is positive definite can be as costly to check as solving the system. On the other hand, some conditions affecting convergence, such as positive diagonal elements, diagonal dominance, and symmetry are relatively easy to verify. For some applications, the theory may not exist to guarantee the convergence of an iterative method. The algorithms in ITPACK have been tested most extensively for linear systems arising from elliptic partial differential equations. The routines can be applied, formally, to any linear system which fits in high-speed memory. However, rapid convergence, and indeed convergence itself cannot be guaranteed unless the matrix of the system is symmetric and positive definite. Success can be expected, though not guaranteed, for mildly nonsymmetric systems. In other words, iterative methods may not converge when applied to systems with coefficient matrices which are completely general with no special properties.

This article discusses the usage of ITPACK and gives a few test results. The description of the iterative methods is given in [4]. The underlying theory on which the iterative algorithms are based is described in [6]. A survey of the iterative methods in ITPACK is presented in [11].

Throughout this paper, we adopt notation such as **SOR()** when referring to a subroutine and **A(*)** for a single-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

[1]In this ordering, the components of the unknown vector u are considered as either "red" or "black". A "red-black ordering" is any ordering such that every black unknown follows all of the red unknowns. This ordering of unknowns leads to a 2×2 "red-black partitioning" of the coefficient matrix, that is, a matrix of the form

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

with diagonal submatrices D_R and D_B. The original linear system may require rearrangement in order to arrive at this form.

G are denoted $m(G)$ and $M(G)$, respectively.

2 Sparse Matrix Storage

The sparse storage scheme used in ITPACK is a common one. It is a row-wise representation of the nonzero entries in the coefficient matrix of the linear system. For a nonsymmetric coefficient matrix, all of the nonzero values in each row are stored in a contiguous block of data in a real-valued array **A(*)**. If the matrix is symmetric, computer memory can be saved by storing only the nonzero entries in each row on and above the main diagonal. For either nonsymmetric or symmetric sparse storage, associated column numbers are stored in an integer-valued array **JA(*)** such that $\mathbf{JA}(\mathbf{K})$ is the column number for the value $\mathbf{A}(\mathbf{K})$. A mapping vector **IA(*)** is used to denote the starting locations of each of the contiguous blocks. The beginning of the linear block for row I is given by $\mathbf{IA}(\mathbf{I})$, the end by $\mathbf{IA}(\mathbf{I}+\mathbf{1})-\mathbf{1}$, and its length by $\mathbf{IA}(\mathbf{I}+\mathbf{1})-\mathbf{IA}(\mathbf{I})$. Thus, **IA(*)** will contain $\mathbf{N}+\mathbf{1}$ elements to accommodate a linear system of order $\mathbf{N}$. The entries for each row may be stored in any order in the contiguous block for that row.

For example, the coefficient matrix

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

would be represented in nonsymmetric sparse storage as

$$\begin{aligned} \mathbf{A}(*) &= [11., 14., 15., 22., 33., 14., 44., 45., 15., 45., 55.] \\ \mathbf{JA}(*) &= [1, 4, 5, 2, 3, 1, 4, 5, 1, 4, 5] \\ \mathbf{IA}(*) &= [1, 4, 5, 6, 9, 12] \end{aligned}$$

and in symmetric sparse storage as

$$\begin{aligned} \mathbf{A}(*) &= [11., 14., 15., 22., 33., 44., 45., 55.] \\ \mathbf{JA}(*) &= [1, 4, 5, 2, 3, 4, 5, 5] \\ \mathbf{IA}(*) &= [1, 4, 5, 6, 8, 9] \end{aligned}$$

3 Usage

The user is expected to provide the coefficient matrix and the right-hand side of the linear system to be solved. The data structure for the matrix of the system is either the symmetric or nonsymmetric sparse storage format described in Section 2. An initial guess for the solution should be provided, if one is known; otherwise, it can be set to all zero values. A series of approximations for the solution are generated iteratively until the convergence criteria is satisfied. The algorithms are performed in two work space arrays and some control over the algorithmic procedure can be obtained from switches in two parameter arrays.

There are seven main subroutines in ITPACK, each corresponding to an iterative method. They are:

Subroutine	Method
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 the calling sequence is:

CALL ⟨ *method* ⟩ **(N, IA, JA, A, RHS, U, IWKSP, NW, WKSP, IPARM, RPARM, IER)**

where the 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 linear arrays except variables **N**, **NW**, and **IER**. Moreover, all parameters may be altered by the subroutine call except variables **N** and **NW**. (See Section 7 for additional details.)

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

IA(*) is a vector of length $\mathbf{N+1}$ used in the sparse matrix storage format. It contains the row pointers into **JA(*)** and **A(*)**. [integer array; input]

JA(*) is a vector of length **NZ** (defined in **A(*)** below) used in the sparse matrix storage format. It contains the column numbers for the corresponding entries in **A(*)**. [integer array; input]

A(*) is a vector of length **NZ** used in the sparse matrix storage format. It contains the nonzero entries of the coefficient matrix with positive diagonal elements. (**NZ** is the number of nonzero entries in the upper triangular part of the coefficient matrix when symmetric storage is used and is the total number of nonzeros when nonsymmetric storage is used.) [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. [real array; input/output]

IWKSP(*) is a vector of length $\mathbf{3*N}$ 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.[2] [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 the table near the end of this 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 below. On output, **IPARM(*)** contains the values of the parameters that were changed. (Further details are given later in this 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 below. On output, **RPARM(*)** contains the final values of the parameters that were changed. (Further details are given later in this section.) [real array; input/output]

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 the table at the end of this section for the meaning of nonzero values.) [integer; output]

The user may supply nondefault values for selected quantities in **IPARM(*)** and **RPARM(*)** by first executing

CALL DFAULT (IPARM, RPARM)

and then assigning the appropriate nondefault values before calling a solution module of ITPACK.

The iterative algorithms used in ITPACK are quite complicated and some knowledge of iterative methods is necessary to completely understand them. The interested reader should consult the technical report [4] and the book [6] for details. 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 ω for the SOR and SSOR methods), **SPECR** (estimated spectral radius of the SSOR matrix), **BETAB** (estimate for the spectral

[2] 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, $\mathbf{IP}(\mathbf{J}) = \mathbf{I}$.

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 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 sparse storage format switch. Default: 0

[0: symmetric sparse storage;
1: nonsymmetric sparse storage]

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 [4, 6] for details and $\mathbf{RPARM(I), I = 2, 3, 5, 6, 7}$ 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 $m(B) = -M(B)$.[3] 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). [4] Selecting the correct case may increase the rate of convergence of the iterative method. (See [6] for additional discussion on Case I and II. Also, see $\mathbf{RPARM(I)}, \mathbf{I} = \mathbf{2}, \mathbf{3}$ 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 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: -1

[For RS methods,

< 0: compute red-black indexing and permute system;
≥ 0: skip indexing—system already in red-black form;

For other methods,

< 0: skip indexing—system already in desired form;
≥ 0: compute red-black indexing and permute system]

A negative integer value for **IPARM(9)** causes the equations to be handled in the most general way appropriate for the solution method being used. For methods other than RS methods this is the "natural order" while for RS methods it is the "red-black order." A non-negative value produces a red-black permutation for all methods except for the RS methods which are assumed to be in red-black order with the order of the black subsystem **NB** given. If reindexing is performed, **IPARM(9)** will contain the order of the black subsystem on output.

[3] 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.

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

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 $\mathbf{RPARM(I), I = 11, 12}$, 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 $\mathbf{RPARM(I), I = 11, 12}$ 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 [4, 6] 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 $\mathbf{IPARM(6) = 1}$ 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 [4, 5, 18] 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. [5] 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()** for additional details.) Default: 100. $\times$ **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 [6].) 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 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 [6].) 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

[5] 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 $j \neq i$.

stopping test has not worked successfully or the original system is ill-conditioned. (See [6] for additional details.)

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()	$\mathbf{4*N+NCG}$
JSI()	$\mathbf{2*N}$
SOR()	$\mathbf{N}$
SSORCG()	$\mathbf{6*N+NCG}$
SSORSI()	$\mathbf{5*N}$
RSCG()	$\mathbf{N+3*NB+NCG}$
RSSI()	$\mathbf{N+NB}$

The value of **NCG** is $\mathbf{2*IPARM(1)}$ for symmetric sparse storage and $\mathbf{4*IPARM(1)}$ for nonsymmetric sparse storage. 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**.

Nonzero integer values of the error flag **IER** indicate that an error condition was detected. These values are listed below according to their numerical value and to the name of the routine in which the flag was set.

Error Flag		Meaning
IER =	0,	Normal convergence was obtained.
=	1 + Mth,	Invalid order of the system, **N**.
=	2 + Mth,	Workspace array **WKSP(*)** is not large enough. **IPARM(8)** is set to the amount of required workspace, **NW**.
=	3 + Mth,	Failure to converge in **IPARM(1)** iterations. **RPARM(1)** is reset to the last stopping value computed.
=	4 + Mth,	Invalid order of the black subsystem, **NB**.
=	101,	A diagonal element is not positive.
=	102,	No diagonal element in a row.
=	201,	Red-black indexing is not possible.
=	301,	No entry in a row of the original matrix.
=	302,	No entry in a row of the permuted matrix.
=	303,	Sorting error in a row of the permuted matrix.
=	401,	A diagonal element is not positive.
=	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.

JCG(), **JSI()**, **SOR()**, **SSORCG()**, **SSORSI()**, **RSCG()**, **RSSI()** assign values to Mth of 10,20,30,40,50,60,70, respectively. **SBELM()**, **PRBNDX()**, **PERMAT()**, **SCAL()**, **ZBRENT()**, **EQRT1S()** are subroutines with error flags in the 100's, 200's, 300's, 400's, 500's, 600's, respectively. These routines perform the following functions: **SBELM()** removes rows and columns, **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.[6]

4 User-Oriented Modules

The array **U(*)** should contain an initial approximation to the solution of the linear system before any ITPACK 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:

CALL VFILL (N, U, VAL)

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

To aid the user in using the iterative methods of ITPACK, four modules for constructing the sparse matrix storage arrays are included. The modules are:

SBINI()	is called at the beginning to initialize the arrays **IA(*)**, **JA(*)**, **A(*)**, and **IWORK(*)**;
SBSIJ()	is called repeatedly to set the individual entries in the matrix and build a link list representation of the matrix structure;
SBEND()	is called at the end to restructure the link list into final sparse storage form;
SBAGN()	is called to return again to the link list representation if **SBEND()** has been called but additional elements are to be added or modified.

These modules are described below.

(a) Initialization:

CALL SBINI (N,NZ,IA,JA,A,IWORK)

Initializes **IA(*)**, **JA(*)**, **A(*)**, and **IWORK(*)** for a system of order **N**. **IA(*)**, **JA(*)**, and **IWORK(*)** are integer arrays of length at least $\mathbf{N+1}$, **NZ**, and **NZ**, respectively. **A(*)** is a real array of length at least **NZ**.

(b) Set individual entries:

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

CALL SBSIJ (N,NZ,IA,JA,A,IWORK,I,J,VAL,MODE,LEVEL,NOUT,IER)

Inserts the value, **VAL**, of the **(I,J)** entry of the user's matrix into the link list representation for that matrix. When using symmetric sparse storage, **J** must be greater than or equal to **I**. If the **(I,J)** entry has already been set then **MODE** specifies the way in which the entry is to be treated:

MODE < 0,	Current entry value is left as is;
$= 0$,	Current entry value is reset to **VAL**;
> 0,	**VAL** is added to the current entry value.

If **LEVEL** is less than 0, **SBSIJ()** causes no printing. If **LEVEL** is 0, fatal errors messages are written to output unit number **NOUT**; and if **LEVEL** is 1 or greater, a message is printed when **SBSIJ()** encounters a value it has already set with the value being reset according to the value of **MODE**. **IER** is an error parameter and returns values of

Error Flag	Meaning
IER = 0,	New **(I,J)** entry is established.
= 700,	**(I,J)** entry is already set—reset according to **MODE**.
= 701,	Improper values for either **I** or **J**.
= 702,	**NZ** is too small—no room for the new entry.

(c) Finalization:

CALL SBEND (N,NZ,IA,JA,A,IWORK)

Restructures the link list data structure built by **SBINI()** and **SBSIJ()** into the final data structure required by ITPACK.

(d) Undo Finalization:

CALL SBAGN (N,NZ,IA,JA,A,IWORK,LEVEL,NOUT,IER)

Returns to link list representation for modification or addition of elements to the system. Repeated calls to **SBSIJ()** can then be made followed by a single call to **SBEND()** to close-out the sparse matrix representation. If **LEVEL** is less than 0, no printing is done and if **LEVEL** is 0 or greater, fatal error information is written to the output unit number **NOUT**. **IER** is an error flag indicating:

Error Flag	Meaning
IER = 0,	Successful completion.
= 703,	**NZ** is too small—no room for the new entry.

Note that **SBINI()** should not be called after **SBAGN()** is called since it would destroy the previous data.

5 Examples

Given a linear system $Au = b$ with

$$A = \begin{bmatrix} 4 & -1 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{bmatrix}, \qquad b = \begin{bmatrix} 6 \\ 0 \\ 0 \\ 6 \end{bmatrix},$$

a program to solve this problem with an initial guess of $u^T = (0, 0, 0, 0)$ using **JCG()** with symmetric sparse storage and printing the final approximate solution vector follows.

```
      INTEGER IA(5), JA(8), IPARM(12), IWKSP(12)
      REAL    A(8), RHS(4), u(4), WKSP(24), RPARM(12)
      DATA A(1),A(2),A(3),A(4) / 4.0,-1.0,-1.0,4.0 /
      DATA A(5),A(6),A(7),A(8) / -1.0,4.0,-1.0,4.0 /
      DATA JA(1),JA(2),JA(3),JA(4) / 1,2,3,2 /
      DATA JA(5),JA(6),JA(7),JA(8) / 4,3,4,4 /
      DATA IA(1),IA(2),IA(3),IA(4),IA(5) / 1,4,6,8,9 /
      DATA RHS(1),RHS(2),RHS(3),RHS(4)  / 6.0,0.0,0.0,6.0 /
      DATA N /4/, NW /24/, ITMAX /4/, LEVEL/1/, IDGTS/2/
C
      CALL DFAULT (IPARM, RPARM)
      IPARM(1) = ITMAX
      IPARM(2) = LEVEL
      IPARM(12) = IDGTS
      CALL VFILL (N, U, 0.E0)
      CALL JCG (N,IA,JA,A,RHS,U,IWKSP,NW,WKSP,IPARM,RPARM,IER)
      STOP
      END
```

The output for this run would be

```
 BEGINNING OF ITPACK SOLUTION MODULE JCG

 JCG HAS CONVERGED IN     2 ITERATIONS.

      APPROX. NO. OF DIGITS (EST. REL. ERROR) = 14.6  (DIGIT1)
   APPROX. NO. OF DIGITS (EST. REL. RESIDUAL) = 14.3  (DIGIT2)


     SOLUTION VECTOR.

               1              2              3              4
      -------------------------------------------------------------
        2.00000E+00    1.00000E+00    1.00000E+00    2.00000E+00
```

Textbook methods such as the Jacobi (J), Gauss-Seidel (GS), Successive Overrelaxation (SOR—fixed relaxation factor omega), Symmetric Successive Overrelaxation (SSOR—fixed relaxation factor omega), and the RS method can be obtained from this package by resetting appropriate parameters after the subroutine **DFAULT()** is called but before ITPACK routines are called.

Method	Use	Parameters
J	**JSI()**	$\mathbf{IPARM(6) = 0, IPARM(7) = 2}$
GS	**SOR()**	$\mathbf{IPARM(6) = 0}$
SOR—fixed omega	**SOR()**	$\mathbf{IPARM(6) = 0, RPARM(5) = OMEGA}$
SSOR—fixed omega	**SSORSI()**	$\mathbf{IPARM(6) = 0, RPARM(5) = OMEGA}$
RS	**RSSI()**	$\mathbf{IPARM(6) = 0}$

These methods were not included as separate routines because they are usually slower than the accelerated methods included in this package.

On the black unknowns, the Cyclic Chebyshev Semi-Iterative (CCSI) method of Golub and Varga [2] gives the same result as the RSSI method. The CCSI and RSSI methods converge at the same rate, and each of them converges twice as fast as the JSI method. This is a theoretical result [6] and does not count the time involved in establishing the red-black indexing and the red-black partitioned system. Similarly, the Cyclic Conjugate Gradient (CCG) method with respect to the black unknowns, considered by Reid [16] (see also Hageman and Young [6]), gives the same results as the RSCG method. Also, the CCG and the RSCG methods converge at the same rate, and each of them converges, theoretically, exactly twice as fast as the JCG method. Hence, the accelerated RS methods are preferable to the accelerated J methods when using a red-black indexing.

6 Numerical Results

The iterative algorithms in ITPACK have been tested over a wide class of matrix problems arising from elliptic partial differential equations with Dirichlet, Neumann, and mixed boundary conditions on arbitrary two-dimensional regions (including cracks and holes) and on rectangular three-dimensional regions [1]. Both finite-difference and finite-element procedures have been employed to obtain the linear systems. The two sample problems presented here, while simple to pose, are representative of the behavior of the ITPACK routines for more complex problems. The iterative algorithms make no use of the constant coefficients in these two problems or of the particular structure of the resulting linear system. Because the ITPACK code is not tailored to any particular class of partial differential equations or discretization procedure, but rather to sparse linear systems, it is felt that the package can be used to solve a wider class of problems.

We now consider two simple partial differential equations which when discretized by finite-difference methods give rise to large sparse linear systems. We obtain the solution of each of these systems by the seven algorithms in ITPACK 2C. These numerical results should aid the user of ITPACK in determining the amount of time required when solving more complicated sparse systems. However, one should not interpret these execution times as conclusive by themselves. Variances introduced by different compilers, computer systems, and timing functions can sometimes be significant. Moreover, the number of iterations required by an iterative method is dependent on the problem being solved, the initial estimate for the solution, the parameter estimates used, and the relative accuracy requested in the stopping criterion **RPARM(1)**. These tests were run on the CDC Cyber 170/750 at the University of Texas with the FTN 4.8 compiler (OPT=2).

To obtain representative sparse linear systems, we discretize the following two self-adjoint elliptic partial differential equations in a region with prescribed conditions on the boundary. Here u_{xx}, u_{yy}, u_{zz} are partial derivatives and du/dn is the derivative in the normal direction.

$$\begin{aligned} u_{xx} + 2u_{yy} &= 0, \qquad (x,y) \text{ in } S = (0,1) \times (0,1) \\ u &= 1 + xy, \qquad (x,y) \text{ on the boundary of } S \end{aligned} \tag{1}$$

Using the standard 5-point symmetric finite-difference operator with $h = \frac{1}{20}$, we obtain a sparse linear system with 1729 nonzero elements and 361 unknowns.

$$\begin{aligned} u_{xx} + 2u_{yy} + 3u_{zz} &= 0, \qquad (x,y,z) \text{ in } C = (0,1) \times (0,1) \times (0,1) \\ \text{On the boundary of C:} & \\ u &= 1, \qquad (0,y,z),\ (x,0,z), \text{ or } (x,y,0) \\ du/dn &= yz(1+yz), \qquad (1,y,z) \\ du/dn &= xz(1+xz), \qquad (x,1,z) \\ du/dn &= xy(1+xy), \qquad (x,y,1) \end{aligned} \tag{2}$$

Using the standard 7-point symmetric finite difference operator with $h = \frac{1}{7}$, we obtain a sparse linear system with 1296 nonzero elements and 216 unknowns.

Tables 1 and 2 display the number of iterations and execution times (in seconds) for the seven methods in ITPACK 2C for the linear systems corresponding to problems (1) and (2), respectively, using symmetric sparse storage. Both the time for the iteration algorithm and the total time for the subroutine call are given. The stopping criterion was set to 5×10^{-6}. To illustrate how effective the adaptive procedures are, we have included in these tables the number of iterations and the time when the optimum iteration parameters were used with no adaptive procedures.

Values corresponding to the red-black ordering with the SSOR methods are omitted from the tables since it is known that these methods are ineffective with this ordering. Since the RS methods are defined for only the red-black ordering, the table entries for these methods with the natural ordering are not included.

7 Notes on Use

Before an iterative algorithm is called to solve a linear system, the values in the array **A(*)** are permuted and scaled. Afterwards, these values are unpermuted and unscaled.

Routine	Ordering	Iterations	Iteration time	Total time
JCG()	Natural	61 (61)	.250 (.247)	.281 (.271)
	Red-black	61 (61)	.232 (.246)	.402 (.413)
JSI()	Natural	108 (95)	.408 (.344)	.439 (.375)
	Red-black	108 (95)	.393 (.332)	.569 (.498)
SOR()	Natural	72 (54)	.356 (.280)	.368 (.307)
	Red-black	65 (47)	.311 (.224)	.469 (.411)
SSORCG()	Natural	17 (13)	.232 (.173)	.264 (.185)
SSORSI()	Natural	23 (22)	.242 (.213)	.273 (.244)
RSCG()	Red-black	31 (31)	.104 (.117)	.269 (.297)
RSSI()	Red-black	60 (48)	.207 (.166)	.358 (.344)

Table 1: Number of Iterations and Execution Times for Problem (1) Using Adaptive and Nonadaptive Procedures (Nonadaptive Data in Parentheses)

Routine	Ordering	Iterations	Iteration time	Total time
JCG()	Natural	28 (28)	.092 (.090)	.107 (.090)
	Red-black	28 (28)	.079 (.074)	.191 (.202)
JSI()	Natural	64 (54)	.166 (.136)	.196 (.152)
	Red-black	64 (54)	.160 (.130)	.268 (.266)
SOR()	Natural	42 (29)	.139 (.095)	.150 (.110)
	Red-black	38 (29)	.124 (.097)	.236 (.231)
SSORCG()	Natural	15 (11)	.136 (.097)	.167 (.111)
SSORSI()	Natural	19 (15)	.138 (.101)	.153 (.117)
RSCG()	Red-black	15 (15)	.032 (.051)	.150 (.169)
RSSI()	Red-black	31 (27)	.075 (.064)	.186 (.196)

Table 2: Number of Iterations and Execution Times for Problem (2) Using Adaptive and Nonadaptive Procedures (Nonadaptive Data in Parentheses)

Consequently, the values in arrays **A(*)** and **RHS(*)** may change slightly due to roundoff errors in the computer arithmetic. Moreover, since entries in each row of the linear system may be stored in any order within a contiguous block of data, the locations of elements of **A(*)** and of corresponding ones in **JA(*)** may change from those given before the permuting and unpermuting was done. The same linear system is defined by the arrays **A(*)**, **JA(*)**, and **IA(*)** whether or not corresponding elements in **A(*)** and **JA(*)** have changed locations within contiguous blocks.

Scaling of the linear system is done as follows to reduce the number of arithmetic operations. The diagonal entries of the linear system are checked for positivity and are moved to the first **N** locations of the array **A(*)**. The nonzero off-diagonal entries of the linear system $Au = b$ are scaled. 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 and to possible movement of entries within contiguous blocks of data.

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

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 [18]. 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()**, **QSORT()**], removal of rows [**SBELM()**], error analysis [**PERROR()**].

The timing routine **TIMER()** should call a routine which returns the run time in seconds.

The value of the machine relative precision is contained in the variable **SRELPR** which is set in the subroutine **DFAULT()** and in the test program. This and other default values may be permanently changed when the code is installed by changing their values in the subroutine **DFAULT()**. **SRELPR** must be changed when moving the code to another computer. 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()**.

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)**.

The distribution tape contains the ITPACK 2C software package of 71 subprograms and a testing program **MAIN()** together with its 27 subprograms. The routines **DFAULT()** and **TIMER()** in ITPACK and the program **MAIN()** are the only ones requiring editing by the installer of the package. ITPACK can be made into a compiled program library although not all of it would normally be used in a particular application.

8 ITPACK History

The 2C version of the ITPACK codes described here is the result of several years of research and development. The development of ITPACK began in the early 1970's when Professor Garrett Birkhoff suggested that general purpose software for solving linear systems should be developed for iterative methods as well as for direct methods. Initially, prototype programs were written based on preliminary iterative algorithms involving adaptive selection of parameters and automatic stopping procedures. These programs were tested on a large set of elliptic partial differential equations over domains compatible with the subroutine **REGION()** [8] which superimposed a square grid over the domain. These routines were designed for solving self-adjoint elliptic partial differential equations. Next a preliminary version of ITPACK was coded in standard FORTRAN. The ITPACK routines used iterative algorithms which were refined from the prototype programs. However, these routines were designed to solve large sparse linear systems of algebraic equations instead of partial differential equations. The use of three interchangeable symmetric sparse storage modes in ITPACK 1.0 [3] allowed for great flexibility and made it possible to solve a wider class of problems than the prototype programs and to study different storage modes for iterative methods. The next version, ITPACK 2.0 [4], was significantly faster than its predecessor since it was restricted to allow only one sparse symmetric storage format. Most of the iterative algorithms utilized in the 2.0 version of this package assume that the coefficient matrix of the linear system is symmetric positive definite. As with many packages, the need to handle a slightly larger class of problems, namely, nearly symmetric systems, soon became evident. This required adapting the routines to allow a switch for either a symmetric or nonsymmetric storage mode in ITPACK 2A [5]. Moreover, a modification of the Conjugate Gradient algorithms was developed to handle nearly symmetric systems [12]. ITPACK has been improved in the 2B version [14] by (a) writing more efficient versions of several key subroutines, (b) incorporating Basic Linear Algebra Subprograms, BLAS [15], and (c) improving the user interface with better printing and documentation. Some additional improvements and corrections were made in the 2C version. The algorithms in ITPACK are not guaranteed to converge for all linear systems but have been shown to work successfully for a large number of symmetric and nonsymmetric systems which arise from solving elliptic partial differential equations [1, 13].

The numerical algorithms in ITPACK 2C correspond to those described in the appendix of technical report [5] and outlined in the book [7]. In particular, the SOR code is based on an algorithm suggested to us by L. Hageman. Various other algorithms exist for iterative methods. For example, S. Eisenstat has an implementation of the Symmetric Successive Overrelaxation preconditioned Conjugate Gradient procedure.[7]

Modules based on the seven iterative routines in ITPACK have been incorporated into the elliptic partial differential equation solving package ELLPACK [17] together with all the necessary translation routines needed. The user-oriented modules described in Section 4 are not in ELLPACK. Moreover, if the ELLPACK system is not being used to generate the linear system for ITPACK, it is recommended that ITPACK be used as a stand-alone package apart from ELLPACK.

Acknowledgements

The authors wish to thank the referee for carefully going through the code and documentation for several different versions. Test runs were made on a variety of computer systems and helpful suggestions were made by R. Boisvert, W. Coughran, J. Dongarra, W. Dyksen, S. Eisenstat, S. Fillebrown, P. Gaffney, W. Gordon, R. Hanson, R. Lynch, J. Rice, B. Ward, and others. These suggestions and comments together with those of the referee have resulted in an improved software package. ITPACK has been tested on the following computing machines: CDC 6400, 6500, 6600, 7600, Cyber 170/750, 203, 205; Cray 1; DEC 10, 20, PDP 10, VAX 11/750, 11/780; IBM 195, 370/158, 3033; PRIME 400, 750; and others.

[7]Private communication.

References

[1] S. Eisenstat, A. George, R. Grimes, D. Kincaid, and A. Sherman. "Some Comparisons of Software Packages for Large Sparse Linear Systems," in *Advances in Computer Methods for Partial Differential Equations III*, (R. Vichnevetsky and R. Stepleman, eds.), Publ. IMACS, Department of Computer Science, Rutgers University, New Brunswick, New Jersey, 08903, 1979, pp. 98-106.

[2] G. Golub and R. Varga. "Chebyshev Semi-Iterative Methods, Successive Overrelaxation Iterative Methods, and Second-Order Richardson Iterative Methods," Parts I & II, *Numerische Mathematik*, Vol. 3, 1961, pp. 147-168.

[3] R. Grimes, D. Kincaid, W. Macgregor, and D. Young. "ITPACK Report: Adaptive Iterative Algorithms Using Symmetric Sparse Storage," CNA-139, Center for Numerical Analysis, University of Texas, Austin, Texas, 78712, August 1978.

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

[5] R. Grimes, D. Kincaid, and D. Young. "ITPACK 2A: A FORTRAN Implementation of Adaptive Accelerated Iterative Methods for Solving Large Sparse Linear Systems," CNA-164, Center for Numerical Analysis, University of Texas, Austin, Texas, 78712, October 1980.

[6] L. Hageman and D. Young. *Applied Iterative Methods*, Academic Press, New York, 1981.

[7] L. Hayes and D. Young. "The Accelerated SSOR Method for Solving Large Linear Systems: Preliminary Report," CNA-123, Center for Numerical Analysis, University of Texas, Austin, Texas, 78712, May 1977.

[8] D. Kincaid and R. Grimes. "Numerical Studies of Several Adaptive Iterative Algorithms," CNA-126, Center for Numerical Analysis, University of Texas, Austin, Texas, 78712, August 1977.

[9] D. Kincaid, R. Grimes, W. Macgregor, and D. Young. "ITPACK—Adaptive Iterative Algorithms Using Symmetric Sparse Storage," in *Symposium on Reservoir Simulation*, Society of Petroleum Engineers of AIME, 6200 North Central Expressway, Dallas, Texas, 75206, February 1979, pp. 151-160.

[10] D. Kincaid, R. Grimes, and D. Young. "The Use of Iterative Methods for Solving Large Sparse PDE-Related Linear Systems," *Mathematics and Computers in Simulation XXI*, North-Holland Publishing Company, New York, 1979, pp. 368-375.

[11] D. Kincaid and D. 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.

[12] D. Kincaid and D. Young. "Adapting Iterative Algorithms Developed for Symmetric Systems to Nonsymmetric Systems," in *Elliptic Problem Solvers*, (M. Schultz, ed.), Academic Press, New York, 1981, p. 353-359.

[13] D. Kincaid. "Acceleration Parameters for a Symmetric Successive Overrelaxation Conjugate Gradient Method for Nonsymmetric Systems," in *Advances in Computer Methods for Partial Differential Equations IV*, (R. Vichnevetsky and R. Stepleman, eds.), Publ. IMACS, Department of Computer Science, Rutgers University, New Brunswick, New Jersey, 08903, 1981, pp. 294-299.

[14] D. Kincaid, R. Grimes, J. Respess, and D. Young. "ITPACK 2B: A FORTRAN Implementation of Adaptive Accelerated Iterative Methods for Solving Large Sparse Linear Systems," CNA-173, Center for Numerical Analysis, University of Texas, Austin, Texas, 78712, September 1981.

[15] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh. "Basic Linear Algebra Subprograms for FORTRAN Usage," *ACM Transactions on Mathematical Software*, Vol. 5., No. 3, September 1979, pp. 308-323.

[16] J. Reid. "The Use of Conjugate Gradients for Systems of Linear Equations Possessing Property A," *SIAM Journal of Numerical Analysis*, Vol. 9, 1972, pp. 325-332.

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

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

[19] D. Young and D. Kincaid. "The ITPACK Package for Large Sparse Linear Systems," in *Elliptic Problem Solvers*, (M. Schultz, ed.), Academic Press, New York, 1981, pp. 163-185.