# Sparse Matrix Test Problems

IAIN S. DUFF
Harwell Laboratory
and
ROGER G. GRIMES
and
JOHN G. LEWIS
Boeing Computer Services

We describe the Harwell–Boeing sparse matrix collection, a set of standard test matrices for sparse matrix problems. Our test set comprises problems in linear systems, least squares, and eigenvalue calculations from a wide variety of scientific and engineering disciplines. The problems range from small matrices, used as counter-examples to hypotheses in sparse matrix research, to large test cases arising in large-scale computation. We offer the collection to other researchers as a standard benchmark for comparative studies of algorithms. The procedures for obtaining and using the test collection are discussed. We also describe the guidelines for contributing further test problems to the collection.

Categories and Subject Descriptors: G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra—*sparse and very large systems*; G.4 [**Mathematics of Computing**]: Mathematical Software

General Terms: Measurement, Performance

Additional Key Words and Phrases: Matrix collection, sparse matrices, test matrices

## 1. INTRODUCTION

Research in solving problems involving sparsity has always been motivated by the need to solve practical large-scale problems. Thus it is important that the evaluation of techniques for exploiting sparsity should be strongly influenced by the performance of these techniques on realistic test problems. It is often hard for research workers in universities to get access to such problems. People working in a laboratory or industrial environment will usually be exposed to relatively

Part of this work was performed while I. S. Duff was visiting Argonne National Laboratory. The work was supported in part by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under contract W-31-109-Eng-38. The work of R. G. Grimes and J. G. Lewis was supported in part by AFOSR grant F49620-87-C-0037.
Authors' addresses: I. S. Duff, Computer Science and Systems Division, Harwell Laboratory, Oxfordshire OX11 0RA, UK; R. G. Grimes and J. G. Lewis, Engineering and Scientific Services Division, Boeing Computer Services, P.O. Box 24346, Seattle, WA 98124-0346.

few applications and so will not easily know how general their techniques are. Published results are often difficult to compare because the problems used for evaluation are different. For these reasons we have organized and continue to maintain a database of sparse matrices that we offer to other researchers.

The main goal of our work is to provide:

(i) an extensive collection of interesting and real test problems in a common, general, machine-readable form with detailed documentation on the problems;
(ii) easy access to the test problems and to interesting subsets of the collection; and
(iii) a simple mechanism for adding new problems to the collection.

Our intention is to allow this test set to be disseminated as widely as possible. For most readers the most important issues are the contents of the collection and how to access the collection. The current contents are outlined in Section 2. The mechanisms for obtaining all or part of the collection are presented in Section 3. Section 4 contains the general forms of the matrix representations in the collection. Those three sections provide an overview of the current collection. In Section 5 we offer readers the opportunity of contributing to the long-term value of the collection, and we give guidelines for adding new problems to the collection. We conclude with a discussion of the history of this collection and remarks on some related areas. We also include as an appendix full details of the storage structures and formats used to hold the data in our test set.

By the publication of this report, we intend to broadcast the availability of our set of sparse matrix test problems and to provide information on what is contained in our set. The nature of scientific computation changes rapidly, and so we do not envisage this current collection as static or complete. For this reason full details of the collection are not given in this paper. Instead we are publishing separately a Users' Guide for the Harwell–Boeing Sparse Matrix Collection [1], which gives the finer details of the contents and formats of the collection. The Users' Guide is published as a technical report at each of our respective institutions and will be updated periodically to reflect changes to the collection. The Users' Guide can be obtained by writing to any of the authors.

## 2. SUMMARY OF CURRENT CONTENTS

The Harwell–Boeing Sparse Matrix Collection currently comprises matrices from more than 20 disciplines. These disciplines include structural analysis (static and dynamic), partial differential equations, circuit analysis, power systems, atomic spectra, oil reservoir modeling, linear programming, atmospheric pollution, finite-element analysis, simulation, chemical kinetics, solution of stiff ordinary differential equations, chemical engineering, demography, and econometrics.

The problems range in order from 9 to 44,609. Many of these matrices stem from actual applications and exhibit numerical pathologies that arise in practice. Other matrices also come from practical problems, but we maintain only the sparsity structure of these matrices for reasons of space or because the value of the problems is strictly for evaluating sparsity-preserving techniques. At present there are 292 matrices represented by over 110 Megabytes of data. In Table I, we

Table I. Summary of Collection

| Discipline | Number of matrices in set | Largest matrix in set | |
|---|---|---|---|
| | | Order | Number of entries |
| Counter-examples—small matrices | 3 | 11 | 76 |
| Original Harwell test set | 36 | 822 | 4,841 |
| Air traffic control | 1 | 2,873 | 15,032 |
| Astrophysics | 2 | 765 | 24,382 |
| Chemical engineering | 16 | 2,021 | 7,353 |
| Circuit simulation | 1 | 991 | 6,027 |
| Demography | 3 | 3,140 | 543,162 |
| Economic modeling | 11 | 2,529 | 90,158 |
| Nuclear reactor core modeling | 3 | 1,374 | 8,606 |
| Optimal power flow problems | 3 | 4,929 | 47,369 |
| Stochastic modeling | 7 | 1,107 | 5,664 |
| Acoustic scattering | 4 | 841 | 4,089 |
| Oil reservoir modeling | 19 | 5,005 | 20,033 |
| Stiff ODE problems | 10 | 760 | 5,976 |
| George and Liu test set—mesh problems | 21 | 3,466 | 13,681 |
| Model PDE problems | 3 | 900 | 4,322 |
| Navier-Stokes problems | 7 | 3,937 | 25,407 |
| Unassembled finite-element matrices | 10 | 5,976 | 15,680 |
| Oceanography | 4 | 1,919 | 17,159 |
| Power network matrices | 14 | 5,300 | 13,571 |
| Everstine test set—ship structures | 30 | 2,680 | 13,853 |
| Structures—eigenproblems | 22 | 15,439 | 133,840 |
| Structures—linear equations | 36 | 44,609 | 1,029,655 |
| Least-squares problems | 4 | 1,850 | 10,608 |

Table II. Summary of Generator Programs

| Source | Description of generator |
|---|---|
| Duff and Grimes | Generator for 5- or 9-point discretization of the Laplacian operator on a rectangular grid |
| Grimes | Generator for 3-dimensional oil reservoir simulation |

list these matrices in summary form by class. The Users' Guide lists all of the matrices in the collection with relevant information on their properties, history, and use in the literature. In addition, we also provide parameterized families of sparse matrices of arbitrary size through generator programs. Our present generator programs are summarized in Table II.

## 3. SYSTEM FOR DISTRIBUTION

Requests for data from the collection should be made to one of the authors. At present there is no charge for this service other than the cost of a tape and postage. Requesters can offset this cost (and simplify our work) by supplying their own tape. The data will be provided in card-image format in either ASCII or EBCDIC blocked format, as requested.

The sheer size of the collection is an obstacle to its distribution and use. As it stands today the total collection requires at least three 2,400-ft. reels of 1,600

BPI tape (a single reel at 6,250 BPI). For most researchers the total collection is not relevant. Unsymmetric matrices are not of interest to researchers studying orderings for symmetric positive definite problems, for example. Workstation users may find problems with millions of entries to be a challenge. Although we can provide the complete collection, we have also developed a mechanism for extracting subcollections. We discuss this mechanism in some detail in the Users' Guide. In this paper, we describe only the salient features to give a flavor of the kind of subsets that can be extracted.

The basis for extracting subcollections is a small database. For each matrix, the information held in the database consists of the unique matrix name or identifier, its type, its source and discipline, its order and number of entries, and key words. From this information we can select any subset specified according to any of the attributes held in the database. A typical request might be of the form "all symmetric matrices from structures problems with order greater than 1,000 but with less than 100,000 entries." One could also request all matrices supplied by Boeing Computer Services, the original Harwell sparse matrix collection, or matrices from other sources, perhaps with additional qualifications. Any of the characteristics held in the database may be used in the selection process.

We use this extraction mechanism ourselves to generate subcollections that we consider to be appropriate benchmarks for some standard requirements. On request we will provide our benchmark subcollections for:

linear equation solvers (symmetric)
linear equation solvers (unsymmetric)
generalized symmetric eigenproblems
linear least-squares problems

The specific contents of each are described in the Users' Guide. These collections all contain systems of medium to large size. We also provide corresponding sets of small matrices to assist in debugging.

## 4. REPRESENTATION OF MATRICES

We use three different modes for storing the sparse matrices. For most matrices we use an explicit sparse matrix representation. A small number of our problems were obtained from finite-element problems in original elemental matrix form. We also include generator routines that can be used to provide families of sparse matrices with certain regular properties. We give only an overview of the representation in this section. Full details are presented in the Appendix.

All matrices held in explicit form in the sparse matrix test collection are stored in a compact format where only the entries corresponding to nonzero values are stored. The standard format for matrices uses a column-oriented form so that only two vectors of length the number of nonzeros are required. We store only the entries of the lower triangle (by columns including the diagonal) of symmetric and Hermitian matrices. Our representation is a simple, general, compact scheme that is widely used in sparse matrix research. Other schemes exist, but they are generally less compact or specialized for specific applications. No provision

has been made at this time for the special requirements of an explicit hypermatrix partitioning or for data compression in areas where certain data appear repeatedly.

Unassembled finite-element matrices are stored in a condensed format that retains the elemental structure. The entries of each element are stored as a small full matrix, where any zero entries are stored explicitly. Our elemental representation allows only structurally symmetric matrices, although the matrix and its element matrices need not be symmetric. Symmetric storage is used for the element matrices when all element matrices (and thus the assembled matrix) are symmetric.

We also allow the storage of right-hand sides. The right-hand-side vectors are stored either as full vectors or in a form similar to that for the matrix itself. To facilitate comparison of iterative methods, our format permits the inclusion not only of right-hand sides, but also of starting vectors and solution vectors.

For portability, each explicitly held matrix is represented on tape by a sequence of card images. The card images contain header records that provide size and formatting information and then the actual indices, pointers, and numerical data. The specific formats used are described in the Appendix. A prospective user may never need these details since we also provide a collection of utility subroutines for using the collection. These subroutines include routines that can be called from a Fortran program to read a matrix from its database form into the array representation described above. Similar routines exist for writing matrices in our database format for use by prospective contributors.

The third representation we have adopted is for easily generated families of sparse matrices. The generator subroutines have a common interface that gives the representation of a sparse matrix described above and the parameters necessary to describe a specific member of the generated family. The generated matrices are produced in exactly the same format as described for the explicitly held matrices. The input parameters are passed to the subroutine through two arrays—one of integer, one of numerical values. Thus all families can be generated through a common interface.

## 5. GUIDELINES FOR CONTRIBUTIONS

It has always been our intention to generate a test collection that reflected the features of many different application areas, and indeed a principal criterion when deciding to augment the test set has been whether the application area is already adequately represented.

In the early days of the collection, we were delighted to accept almost any matrix and were prepared to accept it in any format, although it often involved significant work to reorganize it into a standard format. Thus we were able to build and establish our test collection in a reasonably short time. Now our collection stands at nearly 300 matrices and over 110 Mbytes of data covering a wide range of applications. We are therefore more particular about adding further data to the collection. At the same time we realize that the characteristics of interesting scientific problems evolve with time; this collection will maintain its value only if new types of problems are added when they are encountered.

Therefore, we are still keen to augment the collection but would like any prospective contributions to satisfy one (or more) of the following criteria:

(i) the matrices have unusual numerical or structural properties not currently represented;
(ii) they represent some important or some unusual class of problems;
(iii) they demonstrate the effectiveness or ineffectiveness of particular solution schemes;
(iv) they are samples of matrices available in parametric form, representing a family of matrices, and can be used to demonstrate parametric effects on solution techniques;
(v) they have been widely used as test problems or otherwise referenced in existing literature; or
(vi) they have some other claim to fame!!

All submissions should, in any case, conform to the standard data representation described in Section 4 and the Appendix. Submissions, like requests for the collection, may be addressed to any of the authors. Prospective contributors should contact any of the authors to discuss the suitability of their material before sending any data.

## 6. HISTORICAL COMMENTS AND CONCLUDING REMARKS

Historically, sample test problems have been collected and maintained by individuals in various disciplines, with widely differing representations, formats, and availability. Curtis and Reid used some test examples generated from the solution of ordinary differential equations, supplemented by others from colleagues and conferences, in the development of the Harwell MA18 code in 1971. Duff and Reid [3] then organized the collection and augmented it. This was known as the Harwell collection of sparse matrices and represented the only collection in a uniform format covering several disciplines. Despite its lack of coverage in some important areas, it has been widely distributed. Researchers at Boeing Computer Services were also generating test examples, partly through their development of software for finite-element packages and partly through work with the Electric Power Research Institute. It was natural to coordinate these efforts, and Duff et al. [2] presented the combined Harwell–Boeing collection at the Sparse Matrix Meeting at Fairfield Glade, Tennessee. Since then there have been many requests for data from that collection. The work reported on here is intended to increase the value and coverage of the collection and make the handling of such requests easier and more routine.

We do not know of any comparable set of test matrices, although there are collections of large systems from particular application areas. For example, Gay, Reid, and Saunders all have sets of linear programming problems held in the now standard MPS format. There is also a great need for and interest in the creation of a set of problems for large-scale nonlinear programming. Although sets for small-scale problems exist (for example, [5]), we know of no satisfactory database for the large-scale case. We hope that this organization and collection of a linear set will encourage efforts for the nonlinear case, although it is not yet clear in what format nonlinear problems should be stored.

Very recently Toint [6] and Lenard [4] have presented some initial suggestions in this area.

## APPENDIX. SPARSE MATRIX TEST COLLECTION—MATRIX FORMAT

Each matrix held in explicit form in the sparse matrix test collection is stored in one of two compact formats. The first is for arbitrary matrices in standard sparse matrix formulation. The second format is used to represent unassembled finite-element matrices in an elemental formulation.

Each matrix in the collection is held as a sequence of formatted records that can be read conveniently into FORTRAN arrays. In Sections A1 th.. ·gh A3, we describe the resulting matrix representation. In Section A4, we describe the format itself and give two example programs that read it.

### A1. Standard Sparse Matrix Format

The standard sparse matrix format is column-oriented. That is, the matrix is represented by a sequence of columns. Each column is held as a sparse vector, represented by a list of the row indices of the entries in an integer array and a list of the corresponding values in a separate real array. A single integer array and a single real array are used to store the row indices and the values, respectively, for all of the columns. (Throughout, we use the term "real" in a generic sense so that it should be read as a FORTRAN real, double precision, complex, or double-precision complex as appropriate.) Data for each column are stored in consecutive locations, the columns are stored in order, and there is no space between the columns. A separate integer array holds the location of the first entry of each column and the first free location. For symmetric and Hermitian matrices, we store only the entries of the lower triangle (including the diagonal). For skew symmetric matrices, we hold only the strict lower triangle.

We illustrate the storage scheme with the following example. The $5 \times 5$ matrix

$$\begin{bmatrix} 1. & -3. & 0 & -1. & 0 \\ 0 & 0 & -2. & 0 & 3. \\ 2. & 0 & 0 & 0 & 0 \\ 0 & 4. & 0 & -4. & 0 \\ 5. & 0 & -5. & 0 & 6. \end{bmatrix}$$

would be stored in the arrays COLPTR (location of first entry), ROWIND (row indices), and VALUES (numerical values) according to the following prescription:

| Subscripts | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| COLPTR | 1 | 4 | 6 | 8 | 10 | 12 | | | | | |
| ROWIND | 1 | 3 | 5 | 1 | 4 | 2 | 5 | 1 | 4 | 2 | 5 |
| VALUES | 1. | 2. | 5. | −3. | 4. | −2. | −5. | −1. | −4. | 3. | 6. |

We can generate column 5, say, by observing that its first entry is in position COLPTR(5) = 10 of arrays ROWIND and VALUES. This entry is in row ROWIND(10) = 2 and has value VALUES(10) = 3. Other entries in column 5 are found by scanning ROWIND and VALUES to position

COLPTR(6) −1, that is position 11. Thus, the only other entry in column 5 is in row ROWIND(11) = 5 with value VALUES(11) = 6.

## A2. Finite-Element Matrices in Unassembled Format

Matrices arising in finite-element applications are usually assembled from numerous small elemental matrices. Our collection includes a few sparse matrices in original unassembled form. The storage of the unassembled matrices is analogous to the explicit form above, which stores each matrix as a list of matrix columns. The elemental representation stores the matrix as a list of elemental matrices. Each elemental matrix is represented by a list of the row/column indices (variables) associated with the element and by a small, dense matrix giving the numerical values by columns (in the symmetric case only the lower triangular part). The lists of indices are held contiguously, just as for the lists of row indices in the standard format. The dense matrices are held contiguously in a separate array, with each matrix held by columns. Although there is not a 1:1 correspondence between the arrays of integer and numerical values, our representation does not hold the pointers to the beginning of the real values for each element. These pointers can be created from the index start pointers (ELTPTR) after noting that an element with $v$ variables has $v^2$ real values ($v \times (v + 1)/2$ in the symmetric case).

We illustrate the elemental storage scheme with a small example. Consider a $5 \times 5$ symmetric matrix

$$\begin{pmatrix} 5. & 0. & 0. & 1. & 2. \\ 0. & 4. & 3. & 0. & 6. \\ 0. & 3. & 7. & 8. & 1. \\ 1. & 0. & 8. & 9. & 0. \\ 2. & 6. & 1. & 0. & 10. \end{pmatrix},$$

generated from four elemental matrices,

$$1\begin{pmatrix} 2. & 1. \\ 1. & 7. \end{pmatrix}\quad 1\begin{pmatrix} 3. & 2. \\ 2. & 8. \end{pmatrix}\quad \begin{pmatrix} 2 & 4. & 3. & 6. \\ 3 & 3. & 5. & 1. \\ 5 & 6. & 1. & 2. \end{pmatrix}\quad \begin{pmatrix} 3 & 2. & 8. \\ 4 & 8. & 2. \end{pmatrix},$$

where the variable indices are indicated by the integers before each matrix (columns are identified symmetrically to rows). This matrix would be stored in arrays ELTPTR (location of first entry), VARIND (variable indices), and VALUES (numerical values) according to the following prescription:

| Subscripts | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ELTPTR | 1 | 3 | 5 | 8 | 10 | | | | | | | | | | |
| VARIND | 1 | 4 | 1 | 5 | 2 | 3 | 5 | 3 | 4 | | | | | | |
| VALUES | 2. | 1. | 7. | 3. | 2. | 8. | 4. | 3. | 6. | 5. | 1. | 2. | 2. | 8. | 2. |

## A3. Right-Hand Sides

Where the matrices originate in the solution of linear equations and the right-hand sides are available, the right-hand-side vectors are stored with the matrices.

Usually the right-hand-side vectors are dense, in which case they are stored contiguously (as in ordinary FORTRAN array storage). Multiple right-hand sides are stored as consecutive vectors, so that the right-hand sides are accessible as the columns of a FORTRAN array.

An alternative form is available in which right-hand sides are represented in the same format as the matrix. For unassembled matrices the associated right-hand sides can be represented by elemental contributions. Right-hand sides in elemental form are stored as a sequence of small dense matrices, each small matrix having as many columns as the number of right-hand sides, and with as many rows as the corresponding element in the matrix representation. Within each elemental right-hand side, the rows correspond to the entries in the variable index vector for that element.

The format for assembled sparse matrices is used to store sparse right-hand sides. Applications with sparse right-hand sides are less common, but the sparsity can be used to advantage in direct solution techniques. We only allow sparse right-hand sides for assembled matrices, in which case we store the right-hand sides exactly as a standard sparse matrix, with the same number of rows as the coefficient matrix and the same number of columns as right-hand sides.

We allow the specification of a starting guess for the solution of the problem and a vector that purports to be the exact solution. These can only be supplied as full arrays and only when right-hand side(s) are present. Either or both of these arrays can be present. The starting vector(s) precede the solution vector(s) if both are given and the number of such vectors must be equal to the number of right-hand sides.

## A4. Detailed Formats

Our collection is held in an 80-column, fixed-length format for portability. Each matrix begins with a multiple-line header block, which is followed by two, three, or four data blocks. The header block contains summary information on the storage formats and space requirements. From the header block alone, the user can determine how much space will be required to store the matrix. Information on the size of the representation in lines is given for ease in skipping past unwanted data.

If there are no right-hand side vectors, the matrix has a four-line header block followed by two or three data blocks containing, in order, the column (or element) start pointers, the row (or variable) indices, and the numerical values. If right-hand sides are present, there is a fifth line in the header block and a fourth data block containing the right-hand side(s). The blocks containing the numerical values and right-hand side(s) are optional. The right-hand side(s) can be present only when the numerical values are present. If right-hand sides are present, then vectors for starting guesses and the solution can also be present; if so, they appear as separate full arrays in the right-hand-side block following the right-hand-side vector(s).

The first line contains the 72-character title and the 8-character identifier by which the matrix is referenced in our documentation. The second line contains the number of lines for each of the following data blocks as well as the total number of lines, excluding the header block. The third line contains a three-character string denoting the matrix type, as well as the number of rows, columns

(or elements), entries, and, in the case of unassembled matrices, the total number of entries in elemental matrices. The fourth line contains the variable FORTRAN formats for the following data blocks. The fifth line is present only if there are right-hand sides. It contains a one-character string denoting the storage format for the right-hand sides, as well as the number of right-hand sides, and the number of row index entries (for the assembled case). The exact format is given by the following, where the names of the FORTRAN variables in the subsequent programs are given in parentheses:

Line 1 (A72, A8)

| | |
|---|---|
| Col. 1–72 | Title (TITLE) |
| Col. 73–80 | Key (KEY) |

Line 2 (5I14)

| | |
|---|---|
| Col. 1–14 | Total number of lines excluding header (TOTCRD) |
| Col. 15–28 | Number of lines for pointers (PTRCRD) |
| Col. 29–42 | Number of lines for row (or variable) indices (INDCRD) |
| Col. 43–56 | Number of lines for numerical values (VALCRD) |
| Col. 57–70 | Number of lines for right-hand sides (RHSCRD)<br>(including starting guesses and solution vectors if present)<br>(zero indicates no right-hand-side data are present) |

Line 3 (A3, 11X, 4I14)

| | |
|---|---|
| Col. 1– 3 | Matrix type (see below) (MXTYPE) |
| Col. 15–28 | Number of rows (or variables) (NROW) |
| Col. 29–42 | Number of columns (or elements) (NCOL) |
| Col. 43–56 | Number of row (or variable) indices (NNZERO)<br>(equal to number of entries for assembled matrices) |
| Col. 57–70 | Number of elemental matrix entries (NELTVL)<br>(zero in the case of assembled matrices) |

Line 4 (2A16, 2A20)

| | |
|---|---|
| Col. 1–16 | Format for pointers (PTRFMT) |
| Col. 17–32 | Format for row (or variable) indices (INDFMT) |
| Col. 33–52 | Format for numerical values of coefficient matrix (VALFMT) |
| Col. 53–72 | Format for numerical values of right-hand sides (RHSFMT) |

Line 5 (A3, 11X, 2I14)
Only present if there are right-hand sides present.

| | | |
|---|---|---|
| Col. 1 | Right-hand side type:<br>F for full storage or<br>M for same format as matrix | (RHSTYP) |
| Col. 2 | G if a starting vector(s) (Guess) is supplied. | |
| Col. 3 | X if an eXact solution vector(s) is supplied. | |
| Col. 15–28 | Number of right-hand sides (NRHS) | |
| Col. 29–42 | Number of row indices (NRHSIX)<br>(ignored in case of unassembled matrices) | |

The three character type field on line 3 describes the matrix type. The following table lists the permitted values for each of the three characters. As an example of the type field, RSA denotes that the matrix is real, symmetric, and assembled.

First Character:

- R Real matrix
- C Complex matrix
- P Pattern only (no real values supplied)

Second Character:

- S Symmetric
- U Unsymmetric
- H Hermitian
- Z Skew symmetric
- R Rectangular

Third Character:

- A Assembled
- E Elemental matrices (unassembled)

To formalize the logical block structure of the data, we have included two pieces of sample FORTRAN code for reading a matrix in the format of the sparse matrix test collection. Both codes assume the data comes from input unit LUNIT. Neither is a complete code. Real code should include error checking to ensure that the target arrays into which the data are read are large enough. The design allows the arrays to be read by a separate subroutine that can avoid the use of possibly inefficient implicit DO-loops. A complete FORTRAN subroutine for reading matrices from the tape is supplied with the collection.

The first sample code is for the standard case, a sparse matrix in standard format with no right-hand sides.

```
C     ------------------------------------------------------
C     ... SAMPLE CODE FOR READING A SPARSE MATRIX IN STANDARD
C     FORMAT
C     ------------------------------------------------------

      CHARACTER   TITLE*72,   KEY*8       MXTYPE*3,
     1            PTRFMT*16, INDFMT*16, VALFMT*20, RHSFMT*20
      INTEGER     TOTCRD, PTRCRD, INDCRD, VALCRD, RHSCRD,
     1            NROW,   NCOL,   NNZERO, NELTVL
      INTEGER     COLPTR (*), ROWIND (*)
      REAL        VALUES (*)

C     ------------------------
C     ... READ IN HEADER BLOCK
C     ------------------------

      READ (LUNIT, 1000)  TITLE,   KEY,
     1                    TOTCRD, PTRCRD, INDCRD, VALCRD, RHSCRD,
     2                    MXTYPE, NROW,   NCOL,   NNZERO, NELTVL,
     3                    PTRFMT, INDFMT, VALFMT, RHSFMT
 1000 FORMAT (A72, A8 / 5I14 / A3, 11X, 4I14 / 2A16, 2A20)
```

```
C     ------------------------------
C     ... READ MATRIX STRUCTURE
C     ------------------------------

      READ (LUNIT, PTRFMT) (COLPTR (I), I = 1, NCOL + 1)
      READ (LUNIT, INDFMT) (ROWIND (I), I = 1, NNZERO)
      IF      (VALCRD .GT 0) THEN

C       ------------------------
C       ... READ MATRIX VALUES
C       ------------------------

        READ (LUNIT, VALFMT) (VALUES (I), I = 1, NNZERO)
      ENDIF
```

The second sample code illustrates the full generality of the representation.

```
C     ------------------------------------------------------------------
C     ... SAMPLE CODE FOR READING A GENERAL SPARSE MATRIX, POS-
C         SIBLY WITH RIGHT-HAND-SIDE VECTORS
C     ------------------------------------------------------------------

      CHARACTER  TITLE*72,    KEY*8,       MXTYPE*3, RHSTYP*3,
     1           PTRFMT*16, INDFMT*16, VALFMT*20, RHSFMT*20
      INTEGER    TOTCRD, PTRCRD, INDCRD, VALCRD, RHSCRD,
     1           NROW,   NCOL,   NNZERO, NELTVL,
     2           NRHS,   NRHSIX, NRHSVL, NGUESS, NEXACT
      INTEGER  POINTR (*), ROWIND (*), RHSPTR (*), RHSIND (*)
      REAL     VALUES (*), RHSVAL (*), XEXACT (*), SGUESS (*)

C     ------------------------
C     ... READ IN HEADER BLOCK
C     ------------------------

      READ (LUNIT, 1000)  TITLE,    KEY,
     1                    TOTCRD, PTRCRD, INDCRD, VALCRD, RHSCRD,
     2                    MXTYPE, NROW,   NCOL,   NNZERO, NELTVL,
     3                    PTRFMT, INDFMT, VALFMT, RHSFMT
      IF (RHSCRD .GT. 0)
     1   READ (LUNIT, 1001) RHSTYP, NRHS, NRHSIX
 1000 FORMAT (A72, A8 / 5I14 / A3, 11X, 4I14 / 2A16, 2A20)
 1001 FORMAT (A3, 11X, 2I14)

C     ------------------------------
C     ... READ MATRIX STRUCTURE
C     ------------------------------

      READ (LUNIT, PTRFMT) (POINTR (I), I = 1, NCOL + 1)
      READ (LUNIT, INDFMT) (ROWIND, (I), I = 1, NNZERO)
      IF (VALCRD .GT. 0) THEN

C       ------------------------
C       ... READ MATRIX VALUES
C       ------------------------

        IF (MXTYPE (3:3) .EQ. 'A') THEN
          READ (LUNIT, VALFMT) (VALUES (I), I = 1, NNZERO)
        ELSE
          READ (LUNIT, VALFMT) (VALUES (I), I = 1, NELTVL)
        ENDIF
```

```
C     -----------------------------
C     ... READ RIGHT-HAND SIDES
C     -----------------------------
      IF (NRHS .GT. 0) THEN
        IF (RHSTYP (1:1) .EQ. 'F') THEN
C       -------------------------------------
C       ... READ DENSE RIGHT-HAND SIDES
C       -------------------------------------
        NRHSVL = NROW * NRHS
        READ (LUNIT, RHSFMT) (RHSVAL (I), I = 1, NRHSVL)
      ELSE
C       -----------------------------------------------------
C       ... READ SPARSE OR ELEMENTAL RIGHT-HAND SIDES
C       -----------------------------------------------------
        IF (MXTYPE (3:3) .EQ. 'A') THEN
C         -------------------------------------------------------------
C         ... SPARSE RIGHT-HAND SIDES--READ POINTER ARRAY
C         -------------------------------------------------------------
          READ (LUNIT, PTRFMT) (RHSPTR (I), I = 1, NRHS + 1)
C         -------------------------------------------------------------
C         ... READ SPARSITY PATTERN FOR RIGHT-HAND SIDES
C         -------------------------------------------------------------
          READ (LUNIT, INDFMT) (RHSIND (I), I = 1, NRHSIX)
C         -----------------------------------------
C         ... READ SPARSE RIGHT-HAND SIDE VALUES
C         -----------------------------------------
          READ (LUNIT, RHSFMT) (RHSVAL (I), I = 1, NRHSIX)
        ELSE
C         -----------------------------------------
C         ... READ ELEMENTAL RIGHT-HAND SIDES
C         -----------------------------------------
          NRHSVL = NNZERO * NRHS
          READ (LUNIT, RHSFMT) (RHSVAL (I), I = 1, NRHSVL)
        ENDIF
      END IF
      IF (RHSTYP (2:2) .EQ. 'G') THEN
C       -----------------------------
C       ... READ STARTING GUESSES
C       -----------------------------
        NGUESS = NROW * NRHS
        READ (LUNIT, RHSFMT) (SGUESS (I), I = 1, NGUESS)
      END IF
      IF (RHSTYP (3:3) .EQ. 'X') THEN
C       ----------------------------
C       ... READ SOLUTION VECTORS
C       ----------------------------
        NEXACT = NROW * NRHS
        READ (LUNIT, RHSFMT) (XEXACT (I), I = 1, NEXACT)
      END IF
    END IF
  END IF
```

The code above outlines the structure of the data. The interpretation of the row (or variable) index arrays will require knowledge of the matrix and right-hand-side types, as read in this code.

ACKNOWLEDGMENTS

We would like to thank John Reid for his comments when we were setting up this current collection, and for his remarks on a draft of this paper, Gail Pieper for stylistic and grammatical comment, and the editor, Michael Saunders, and the anonymous referees for their helpful comments.

REFERENCES

1. DUFF, I. S., GRIMES, R. G., AND LEWIS, J. G. *Users' Guide for the Harwell–Boeing sparse matrix collection.* To appear as Harwell Report and Boeing Report.
2. DUFF, I. S., GRIMES, R. G., LEWIS, J. G., AND POOLE, W. G. JR. Sparse matrix test problems. *SIGNUM Newsl. 17,* 2, (1982), p. 22.
3. DUFF, I. S., AND REID, J. K. Performance evaluation of codes for sparse matrix problems. In *Performance Evaluation of Numerical Software.* Fosdick, L. D., ed. Elsevier, North-Holland, New York, 1979, pp. 121–135.
4. LENARD, M. L. Standardizing the interface with nonlinear optimizers. Presented at TIMS/ ORSA Joint National Meeting (Washington, D.C., Apr. 25–27, 1989). Paper WC 36.1, *TIMS/ORSA Bull. 25.*
5. MORÉ, J. J., GARBOW, B. S., AND HILLSTROM, K. E. Testing unconstrained optimization software. *ACM Trans. Math. Softw. 7* (1981), 17–41.
6. TOINT, PH. L. Call for test problems in large scale nonlinear optimization. Report 87/9, Department of Mathematics, Facultés Universitaires ND de la Paix, Namur, Belgium, 1987.

Received December 1987; revised July 1988; accepted July 1988