RAL-TR-97-031

The Rutherford-Boeing Sparse Matrix Collection[1]

Iain S. Duff[2], Roger G. Grimes[3], and John G. Lewis[3]

ABSTRACT

The Rutherford-Boeing Sparse Matrix Collection is a large and actively growing archive of test problems for research in sparse linear algebra. We discuss the overall design of the Collection and specify the formats and file structures used to represent test problems. Mechanisms for obtaining test problems from the Collection and procedures for submitting new problems to the Collection are also presented.

The Rutherford-Boeing Sparse Matrix Collection greatly extends its predecessor, the Harwell-Boeing Sparse Matrix Collection, in scope, in the ways problems can be represented, and in the number of matrices and test problems that are included. We have also changed the way in which the Collection can be accessed by making extensive use of the World Wide Web.

Keywords: sparse matrices, test matrices, sparse linear equations, sparse eigenvalue problems.

AMS(MOS) subject classifications: 65F05, 65F50.

---

[1] Current reports available by anonymous ftp from matisa.cc.rl.ac.uk (internet 130.246.8.22) in the directory "pub/reports". This report is in file duglRAL97031.ps.gz. This report is also published as Boeing Report ISSTECH-97-017.

[2] email address: isd@rl.ac.uk

[3] Lewis and Grimes work at The Boeing Company, Mail Stop 7L-22, P.O. Box 3707, Seattle WA 98124-2207, USA. email addresses: roger.g.grimes@boeing.com and john.g.lewis@boeing.com.

Department for Computation and Information
Atlas Centre
Rutherford Appleton Laboratory
Oxon OX11 0QX

September 19, 1997.

Contents

Chapter 1

Overview

The Rutherford-Boeing Sparse Matrix Collection is a repository for sparse matrices for testing and validating algorithms. The Collection includes a large number of matrices, together with documentation on the source and use of the matrices, and other data that help to define the linear algebraic settings from which the matrices were taken. We define what we mean by the *Collection* in §1.1.

The most important feature of the Collection is that it provides a reproducible set of test data. Data published as official data in the Collection are never changed. Matrices, once in the Collection, are never altered. We provide software to help ensure that data has not been corrupted by transmission errors. We describe how we represent matrices and associated problem data in §1.2 and §1.3, respectively. We discuss our convention for naming files in §1.4 indicating the mechanism for associating more than one data file with the same problem. We consider the issue of "reproducibility" in more detail in §1.5. Filters that extract certain features from the data are often used, but they can cause problems with reproducibility. We discuss our policy on these in §1.6.

The Rutherford-Boeing Sparse Matrix Collection is a significantly updated version of the Harwell-Boeing Sparse Matrix Collection that has been distributed for many years by the authors of this report. We summarize the main changes between the Rutherford-Boeing Sparse Matrix Collection and the Harwell-Boeing Sparse Matrix Collection in §1.11 and briefly review the history of the Collection in §1.12. With the Rutherford-Boeing Sparse Matrix Collection we have greatly enhanced electronic distribution of the Collection over the earlier Harwell-Boeing Sparse Matrix Collection. In cooperation with the Matrix Market project at the National Institute of Standards and Technology (NIST) and with the netlib project at the University of Tennessee, the Collection is now easily accessible through anonymous ftp (to `ftp.netlib.org`) and the World Wide Web (WWW) at URL `http://math.nist.gov/MatrixMarket/`. We welcome suggestions on other ways to improve access to the Collection. Fuller details of the WWW interface are given in §1.9. This interface provides a good mechanism for providing further documentation on the Collection (as described in §1.7) and a number of support programs (see §1.8).

The continuing strength of the Collection depends on further contributions. The Collection is always open for contributions by email or through the WWW as discussed further in §1.10. This better facility for contributing data, coupled with the ease of

updating the documentation, will allow the Collection to grow easily and result in a more dynamic Collection.

1.1 Definition of the Collection

It is important that the Collection be defined precisely since matrices from it will often be used to compare or validate software. Any matrix, stored in the standard formats in Tennessee and documented in the User Guide to the Collection, is in the Collection, as are all exact copies. Nothing else is a matrix from the Collection.

1.2 Representation of Matrices

We represent matrices in the Collection in two ways. Most matrices are represented in a general scheme that is capable of representing any sparse matrix. The Collection also includes a second representation to store matrices from finite-element applications in their native elemental formulation. Both representations are realized by compact data structures that store the nonzero entries. The two representations are discussed in Chapter 2.

Matrices in either representation are stored in the Collection as ASCII text files. We use two different schemes for formatting these text files. One scheme extends the format used in the Harwell-Boeing Sparse Matrix Collection and the other extends the Matrix Market format. We refer to these two file formats as the RB and MM formats. The RB format is the more compact of the two and provides a natural growth path for users of the previous Harwell-Boeing Sparse Matrix Collection. The MM format is particularly simple and easy to use, and provides a natural mechanism for users of the Matrix Market format to access the new capabilities of the Rutherford-Boeing Sparse Matrix Collection. Each matrix in the Collection is available in both RB and Matrix Market formats. They are independent of one another, so it is likely that a user will choose one to the exclusion of the other. The details of the RB format for matrix files are given in Chapter 3. The corresponding details of the MM file format are given in Chapter 4.

Each matrix is identified in the Collection by a unique eight character identifier. This matrix identifier is included in the text file itself. It serves as the root name for the file containing the matrix. For a particular matrix whose name is, say, `SAMPLEMX`, the files `samplemx.mtx.rb` and `samplemx.mtx.mm` stored at the netlib Rutherford-Boeing archive contain the data for matrix `SAMPLEMX` in RB and MM format respectively.

Numerical values are represented in a (floating-point) decimal form that is readable with free-format input routines from either Fortran or C. Within the limitations of this representation, our standard is to represent data in IEEE 64 bit precision. We plan a future release that will include a portable IEEE binary representation of matrices.

1.3 Representation of Problems

The Rutherford-Boeing Sparse Matrix Collection contains extensive supplementary information which, for example, further describes the linear algebra problem in which

a particular matrix arose or provides verifiable results for computations with a particular matrix. The types of supplementary data for which we currently have established a naming[1] and format convention are given in Table 1. We anticipate adding other supplementary types as needed.

Table 1: Supplementary Files

keyword	*extension*		
	type	*position*	*organization*
orderings	ord	[l|r|s]	
right-hand-sides	rhs	[l|r]	[e|d|s]
solutions	sln	[l|r|s]	
estimates	est	[l|r|s]	
eigenvalues	evl		
singular-values	svl		
eigenvectors	evc	[l|r|s]	
singular-vectors	svc	[l|r|s]	
Schur-basis-vectors	sbv		
Schur-basis-matrix	sbm		
Schur-basis-parameters	sbp		
partition	ipt	[l|r|s]	
covering	icv	[l|r|s]	
Laplacian-values	lvl		
Laplacian-vectors	lvc		
geometry	geo		
auxiliary-values	avl		

Supplementary data are stored in ASCII text files separately from the matrices, but using similar formats. Most of the supplementary data are in the form of dense vectors or dense matrices, although some are represented as sparse vectors or matrices. We use a simple and obvious representation for dense data. For supplementary data in the form of sparse vectors or matrices, we use our existing general representation for sparse matrices. This allows us to use a common and small set of generic file formats that facilitate the task of reading the data. As with matrices, there are independent RB and MM file formats. The RB formats are described in §3.2; the MM formats are described in §4.2.

Often a complete description of a problem requires more than can be represented by a single vector or matrix. As examples, a partial eigensolution consists of a vector of eigenvalues and a set of eigenvectors; a general reordering of a rectangular matrix requires two permutations, which have different dimensions. To represent such composite information, we use multiple files. Each of these examples is represented in the Collection

[1] The *position* and *organization* fields in the filename extension depend on the type and contents of the file. Generally, l, r, and s, correspond to operations on the matrix on the left (rows), the right (columns), or symmetrically (rows and columns). The *organization* subfield is discussed in §3.2 and §4.2.

by a pair of supplementary files. We maintain the connections between such a multiplicity of files in part by a file naming convention that helps to group associated files together.

Using several files together allows us to specify general structural descriptions of matrices. These include arbitrary block structures, matrix partitionings and problem characteristics beyond those directly listed in Table 1. For example, consider the block matrix

$$\begin{pmatrix} A & B \\ C & 0 \end{pmatrix},$$

which arises in a mixed finite-element model in fluid flow problems. To adequately describe the setting in which this matrix arises, it may be useful to identify separately the submatrices involved. This can be done by holding the sparse matrices A, B, and C in separate files linked through the naming convention.

1.4 Naming Conventions

The current Collection contains a large number of matrices, each held in a separate data file. In addition, many matrices have supplementary data that we choose to hold in a simple format. We thus have many data files that have a natural grouping, corresponding to all files pertaining to a particular matrix. To address this interrelationship and to enable better management of the data files, we have established a standard file naming convention.

Matrices in the collection have names of no more than eight characters, using only upper case letters, digits, and underscores. The first character is never an underscore. Matrix data is stored and transmitted in files whose names are of the form

matrixid`.mtx.`*ff*

In this name, the root name *matrixid* is the eight character matrix name in lower case. The extension "`.mtx`" signifies that this file and data represent a matrix in the Collection. The further extension "*ff*" is either `rb` or `mm`, denoting whether the file is in RB or MM format. Thus, the data representing matrix `LNS_3937` is found in file `lns_3937.mtx.rb` and in file `lns_3937.mtx.mm`.

Supplementary files have a more complicated naming structure, of the form:

matrixid.ext.case.ff

The root name *matrixid* is the name of the matrix for which this file provides supplementary data. Thus, the set of all files related to a given matrix is given by the list of files with names *matrixid*`.*`. The second field, the extension *ext* given by the three to five character field listed in Table 1, specifies the characteristics of supplementary data. For example, files with names of the form `mahindas.ordl.*` would contain left (row) orderings for the matrix `MAHINDAS`.

The third field, *case*, is used to distinguish between different cases for the same type of data and to connect closely related files of different types. For example, multiple sets of right-hand sides for a given matrix will have different case identifiers, and

solution vectors for a particular set of right-hand sides for a particular matrix will have the same case identifier as the right-hand sides. Thus, files with names of the form *matrixid.*.case.** will together describe a specific instance of a problem. For example, file names `cegb2802.rhsrd.time0.*` and `cegb2802.slnr.time0.*` could be used to represent a particular set of right-hand sides and the associated solutions for matrix `CEGB2802`. A file named `cegb2802.rhsrd.time2.*` would represent a different set of right-hand sides for this matrix. Names for cases are subject to the same rules as matrix names.

This file naming convention is used to structure the public file archives. Users do not need to adopt the same convention, but the fields that make up the file names are themselves given in the text of the files, so an understanding of the convention will help the user understand the connections between files.

In some cases, problems are described by a set of matrices, not by a single matrix. In such cases, the supplementary data files use as their root name field a name derived from the matrices involved. Our convention is that closely related matrices have names that differ only in a single character; that character is changed, usually to an underscore, or omitted, for data using them in combination. For example, a set of eigenvectors for the generalized eigenproblem for matrices `BCSSTK38` and `BCSSTM38` would be stored in a file whose root name is `bcsst_38`.

There are cases, such as preconditioners, in which the supplementary data are themselves a matrix. In these cases the supplementary data are held as matrices in their own right, with the connection between the primary matrix and the supplementary matrix given in the documentation for each and by use of similar names; again our standard is that the names differ only in a single character.

1.5 Reproducibility

Reproducibility is a major and crucial aspect of the Collection. We must be able to ensure that tests are comparable and that results can be reproduced. Therefore, making any matrix or supplementary data available in the Collection requires "freezing" the data. Otherwise, complete verification of results is not possible.

The data can be corrupted during the distribution process. We use and provide a standard procedure for verifying correct transport of data by means of a variant of the "signature" technology developed for the mirrored distribution of netlib. We organize the signature so that it is the same for the same data, irrespective of whether it is held in RB or MM format.

Specifically we provide a utility program that converts a matrix in any format that meets the specifications of the Collection into a particular canonical ASCII representation of that matrix. A checksum for that ASCII representation is computed; this canonical file checksum is the "signature" of the matrix. We compute and publish the signature for each data file in the Collection; our documentation specifies that *the* Rutherford-Boeing matrix specified by a given name will provide a particular signature. Anything else is not a Rutherford-Boeing matrix. This checksum is not completely sufficient to guarantee that a matrix, obtained from another source or transmitted directly, exactly matches data in the Collection. However, it is unlikely that anything other than particularly clever and malevolent modifications would disguise changes to the data. By computing the

signature from a canonical form, we allow users to make inconsequential modifications to the data and to verify that the changes are indeed inconsequential. When the checksums do not match, the canonical form that is produced during the signature computation can be compared directly with data in Rutherford-Boeing format acquired directly from the Collection; any differences are due to significant modifications and not to changes to white space or case.

The signature computation itself is portable and integer only. Both the code for signature computation and the signatures for all of the data files in the Collection are provided as machine readable files in the Matrix Market and netlib archives. For ease of transmission, files are usually "gzipped", which results in file sizes not much larger than if we stored binary data.

Programs that generate matrices present a different challenge for reproducibility. We avoid all problems of changes of computer or environment by not including such programs as part of the Collection. However, we include specific generated matrices in the Collection if they are formally submitted as matrices. The matrix in the Collection becomes the specification of the particular problem, not the generator program.

1.6 Filters

We have received many requests for facilities for filtering data, for example, providing only patterns where the Collection includes numerical values, data that is easier to read from C, or zero-based indexing for subscripts. We provide no such filters because each filter option becomes, in effect, a new Rutherford-Boeing matrix format. It is essential that any filtered matrix be reproducible, that the filter be invertible, and that concatenated filters do not corrupt data. Rather than risking a loss of reproducibility, we have modified the Harwell-Boeing representation to make it readable from C and from other free-format input/output environments, and have added the Matrix Market sparse matrix format as a second supported format. We hope that together these two formats are sufficiently useful to remove the need for filters.

1.7 Documentation

Various types of documentation are available in the netlib archives:

- the User Guide for the Collection describing formats and conventions,
- documentation of specific matrices and of specific subsets,
- documentation of the entire Collection,
- a listing of all data files in the Collection and their signatures.

All are available electronically as postscript files. The last is also a data file that can be downloaded as ASCII text. A special request for printed copies can be made to the authors.

A BIBTEX bibliography of papers that reference the Collection or that are referred to in the documentation of the Collection is under construction by the Matrix Market project at `http://math.nist.gov/MatrixMarket/mm.bib`.

1.8 Support Programs

We have developed a number of programming support tools that facilitate use of the Rutherford-Boeing Sparse Matrix Collection. Among these are procedures to:

- read Rutherford-Boeing Sparse Matrix Collection files, in Fortran and in C,
- write Rutherford-Boeing Sparse Matrix Collection files, in Fortran and in C,
- check whether a file adheres to Rutherford-Boeing Sparse Matrix Collection standards,
- compute the signature of a file in a Rutherford-Boeing Sparse Matrix Collection format.

Source code is available through netlib.

1.9 Internet/WWW Interface

The primary repository for the Rutherford-Boeing Sparse Matrix Collection is now with the netlib Project; their repository at Tennessee, `netlib.ornl.gov`, contains *every* matrix in the Collection. The netlib archives also include documentation and support programs for the Collection.

Through an active collaboration between the Rutherford-Boeing Sparse Matrix Collection and the Matrix Market, the Matrix Market is the primary means for accessing the Rutherford-Boeing Sparse Matrix Collection on the WWW. The URL for the Matrix Market is `http://math.nist.gov/MatrixMarket/`. Every file in the Collection on the netlib repository can be accessed through the Matrix Market, which provides useful software tools, access to sources of generated matrices and to documentation for the Collection. The Matrix Market also provides database search and matrix selection services on-line. There are already several mirror sites for netlib in Europe, and it is planned to offer this for the Matrix Market in the near future.

The WWW interface to the Rutherford-Boeing Sparse Matrix Collection includes the capability of downloading to the user's machine:

- a matrix or matrices, optionally including related supplementary files;
- the documentation for a matrix;
- the complete User Guide;
- a set of bibliographic references;
- various visual representations of the matrix.

The Matrix Market provides other capabilities that supplement the use of the Rutherford-Boeing Sparse Matrix Collection. In particular the Matrix Market also supports matrix generation procedures by which the user can execute a more specialized (perhaps proprietary) program to generate a matrix according to a set of supplied parameters. Much effort have been spent in achieving reproducibility of matrices generated by these procedures, but this cannot be completely guaranteed. We expect that many matrices so generated will be found useful in testing or comparisons, but these matrices are *not* part of the Collection unless they have been submitted to the Collection and accepted as specific instances worthy of preservation. We encourage users of the generator programs, who find the results useful, to submit the test matrices that they used.

Matrices from the Harwell-Boeing Sparse Matrix Collection are held in a separate collection at netlib for archival purposes. Normally a user requesting a Harwell-Boeing Sparse Matrix Collection matrix via the WWW interface will be pointed to the equivalent Rutherford-Boeing Sparse Matrix Collection matrix. An explicit request for a Harwell-Boeing Sparse Matrix Collection matrix using its old name will result in a ftp transfer of an appropriate subcollection from the Harwell-Boeing Sparse Matrix Collection archive at netlib.

1.10 New Submissions

We encourage researchers to submit new data to the Collection that will add to its value to the research community. Access to the Collection over the Internet means that new contributions can be added at any time. To qualify for inclusion in the Rutherford-Boeing Sparse Matrix Collection, submitted matrices must

- be in the public domain,
- represent a significant application area,
- possess some inherent interest or novelty for developers of matrix algorithms, and
- not substantially duplicate other matrices in the Collection.

Citations for published references in which the application and algorithms are discussed often help to establish the value of a contribution. Final decisions as to whether to include a problem in the Collection rest with the three caretakers of the Collection.

Please let us know before you contribute to the Rutherford-Boeing Sparse Matrix Collection so we can assess the suitability of your submission and provide you with any recent changes in requirements. Our email addresses are found on the title page of this report and in the Matrix Market WWW pages. We will provide templates for your use in providing descriptive information for your data.

The matrices themselves can be submitted in either RB format or MM format. In either case, submissions to the Collection should be represented to precision that approximates IEEE 64 bit precision, with 17 significant decimal digits, unless the data can be represented exactly by lower precision or when the originating application is run entirely in lower precision than IEEE 64 bit format. Fortran format edit specifiers `ES24.16E3` or `1P, E24.16E3` are suitable descriptors for writing IEEE 64

precision data; the corresponding conversion specification for `fprintf` in C is `"%.16E"`. We provide a submission guide that contains a complete set of specifications for new submissions. This can be obtained by anonymous ftp from `ftp.netlib.org` as file `sparse-matrices/general-documentation/submission-instructions.ascii`, or downloaded from the netlib browser at `http://www.netlib.org`. We are also happy to send copies on request by email.

1.11 Differences between the Harwell-Boeing Sparse Matrix Collection and the Rutherford-Boeing Sparse Matrix Collection

For readers who have already accessed the Harwell-Boeing Sparse Matrix Collection, we discuss in this section the differences and enhancements introduced in the Rutherford-Boeing Sparse Matrix Collection. This section can be skipped for readers who are accessing the Rutherford-Boeing Sparse Matrix Collection for the first time.

There are eight major differences:

- Internet access to the Collection through the Matrix Market and netlib;
- more matrices;
- more supplementary data, such as orderings, estimates of solutions and eigensolutions, and geometric data.
- format changes to facilitate use of the Collection from languages other than Fortran, without compromising compatibility with existing procedures in Fortran;
- formats for matrices in elemental form extended to include rectangular elements and matrices;
- an alternative format available through the Matrix Market, using a different data structure, but containing the same data;
- on-line submission of data;
- access and documentation available by matrix and supplementary data, as well as by subcollection.

The representation of matrices follows that of the Harwell-Boeing Sparse Matrix Collection with only the five minor changes indicated below. A few matrices in the Harwell-Boeing Sparse Matrix Collection violate one of the first three requirements. In all cases where we have updated existing matrices to conform with these new rules, each such matrix has been given a new name and the original matrix, with its old name, will continue to be held in the Harwell-Boeing Sparse Matrix Collection archive.

- The index information for the nonzero entries is always lexicographically sorted.

- Supplementary data in the form of right-hand sides, estimates or solutions are treated as separate files.
- Matrix names are usable as file names; they do not include embedded blanks and the first character is alphanumeric. Where necessary, Harwell-Boeing Sparse Matrix Collection names are transformed by replacing blanks by underscores.
- We represent numerical data to as close to IEEE 64 bit precision as possible. For new submissions to the Collection, we specify that numerical data be given to seventeen significant digits. Lower precision data will be accepted only in cases where the data is only available or accurate to that lower precision. Matrices with integer entries are represented by exact integer data.
- Wherever possible and meaningful, numerical data is provided as well as matrix patterns.

1.12 History and Nomenclature

The effort to provide a systematic collection of sample test problems to the sparse matrix community began with the Harwell Sparse Matrix Collection, organized by John Reid and Iain Duff at the Harwell Laboratory in the 1970's. A long-standing collaboration between the numerical analysis groups at Harwell and Boeing resulted in extensive contributions to the Harwell collection by Roger Grimes and John Lewis. These led naturally to coordination of efforts and a large expansion of the Collection, which in 1982 was renamed the Harwell-Boeing Sparse Matrix Collection. The contents of the original Harwell Sparse Matrix Collection are identified as a subcollection of matrices in the Harwell-Boeing Sparse Matrix Collection. In turn, the contents of the Harwell-Boeing Sparse Matrix Collection are identifiable as a subset of the Rutherford-Boeing Sparse Matrix Collection.

The current Collection represents another major advance in the number and nature of problems provided in addition to fully embracing new methods of access and distribution. In view of the fact that the Numerical Analysis Group at Harwell moved to the Rutherford Appleton Laboratory in 1990, it seemed appropriate to take this opportunity to rename the Collection again, this time to the Rutherford-Boeing Sparse Matrix Collection. This serves to clearly identify this considerably changed Collection. It also removes a source of confusion with the Harwell Subroutine Library, the mathematical software library currently marketed by Harwell, but still supported by the Group at the Rutherford Appleton Laboratory.

Chapter 2

Representations for Sparse Matrices and Supplementary Data

The most important characteristic of an m by n sparse matrix is that it can be represented by far fewer than $m \times n$ numerical entries. This usually means that most of the numerical entries in the matrix are known to be exactly zero. We use this characteristic in our most general representations of sparse matrices, which are realized in files using the RB compressed column or MM coordinate formats.

The Rutherford-Boeing Sparse Matrix Collection also includes sparse matrices in a second representation, used to represent matrices arising in finite-element applications. Matrices from such applications are held in "unassembled" RB and MM elemental representations that better represent the original application.

Supplementary data in the Collection are represented by vectors or matrices. When the supplementary data is sparse, we use the general representation already in place for sparse matrices. Dense supplementary data is represented simply as dense rectangular arrays.

We discuss all of these representations in this chapter. Each of these representations is realized as ASCII files in two different formats. These two formats, the RB format and the Matrix Market format, are discussed in §3 and §4 respectively. The formats are independent of one another, so a typical user of the Collection will likely refer only to this chapter and to the one describing his or her chosen format.

2.1 General Conventions

Throughout, we use the term "numerical" in a generic sense; the values are finite precision decimal representations of real or complex numbers. Complex numbers are stored as a consecutive pair of values; with the real part first, followed by the imaginary part. Integer-valued matrices are represented with integer numerical fields.

A critical aspect of our notion of sparsity is that most entries are known to be zero. In some cases, other entries are also zero, but these zeros would not have been known to be zero *a priori* in the original application. A particular sparse matrix may be an instance of a class of sparse matrices for which the general sparsity pattern is known, but where

an instance of an explicit zero within the expected sparsity pattern would not have been used to advantage within a general application code. In such cases, the matrix in the Rutherford-Boeing Sparse Matrix Collection may include an explicit zero value in the list of "nonzero" entries. In the following text, we will use the term *entry* to define a value that is held in the sparse data structure irrespective of whether its actual value is zero or nonzero.

The only further structure we exploit is symmetry. In the representation of symmetric and Hermitian matrices, our convention is to store only the entries of the lower triangle. For skew symmetric matrices, we hold only the entries in the strict lower triangle, because the diagonal entries are known to be zero.

Certain operations on sparse matrices, for example reordering, are affected by the order in which the sparsity structure is presented. Within each of our representations we use a particular ordering of the entries. The details are given for each representation. The ordering conventions are part of the specification of a matrix being in the Collection and play an important role in ensuring reproducibility.

2.2 General Representations for Sparse Matrices

Example 1 *We illustrate the storage schemes with the* 5×5 *matrix*

$$A = \begin{pmatrix} 1. & -4. & & -8. & \\ & & -6. & & 10. \\ 2. & & & & \\ & 5. & & -9. & \\ 3. & & -7. & & 11. \end{pmatrix}.$$

One of the simplest sparse representations is to store the value and the row and column coordinates for each entry in the matrix. That is, we represent A by a list of triples $\{< i, j, a_{ij} >\}$. A complete list of all the entries of A is given by:

i	1	3	5	1	4	2	5	1	4	2	5
j	1	1	1	2	2	3	3	4	4	5	5
a_{ij}	1.	2.	3.	−4.	5.	−6.	−7.	−8.	−9.	10.	11.

All positions in the matrix not listed are known to be zero. This is the basis for the Matrix Market format for standard matrices, described in §4.1. If there are t entries in the list of triples, this representation requires storage of $2t$ integers and t numerical values.

When the entries for a given column are stored contiguously, as in the example above, the column indices appear as sequences of repeated values. When the columns themselves appear in consecutive order, the sequence of column indices is simply a subsequence of ones followed by a subsequence of twos and so on. The compressed column representation avoids storing this sequence by using a column-oriented representation in which each matrix is represented as a sequence of columns. Each column of the matrix is held as a sparse vector, represented by a list of pairs, each containing the row index and the corresponding numerical value of an entry. When the columns are given in order, the column indices are

unnecessary, and the matrix can be represented as in

i	1	3	5	1	4	2	5	1	4	2	5
a_{ij}	1.	2.	3.	−4.	5.	−6.	−7.	−8.	−9.	10.	11.

with some method of delineating the boundaries between columns. This representation requires only t integers and t numerical values. Additional storage is required to represent the demarcation between columns, suggested by the vertical lines in the list of pairs above. The RB representation uses an additional array that gives the position in the above arrays of the first pair in each column,

PTR	1	4	6	8	10	12

for the above example. For programming convenience, this list includes a final entry which is one greater than the total number of number of entries in the matrix. Thus `PTR(j)` gives the index of the first pair for column `J` and `PTR(J+1)-PTR(J)` gives the number of entries in column j. If n is the column dimension of the matrix, this requires an additional $n+1$ integers above the storage required for the matrix entries. Usually n is considerably smaller than the number of entries.

The two file formats discussed in the next two chapters differ in how they store sparse matrices. The RB format stores all integer information separately from the numerical values, while the MM format stores triples directly, intermixing integer and numerical values.

Neither of these representations nor the corresponding file formats completely specify an ordering of the entries. Formally, the matrix represented by triples is unchanged by changing the order in which the triples are listed. Similarly the compressed column representation is formally unchanged if the order of entries within a column changes. However, such changes can interfere with reproducibility of results if the operations applied to the matrix depend on the ordering of entries.[1]

Matrices from the Rutherford-Boeing Sparse Matrix Collection always present the entries in lexicographic ordering of the indices. For the compressed column representation, this means that entries within each column appear in ascending order by row index. For the triples representation, the entries appear in the same order as in the compressed column representation of the same matrix. That is, the triple $\{< i, j, a_{ij} >\}$ appears before the triple $\{< k, l, a_{kl} >\}$ whenever $j < l$ or when $j = l$ and $i < k$. In both representations, only a single entry will appear for a given pair of indices.

2.3 Finite-Element Matrices in Elemental Representation

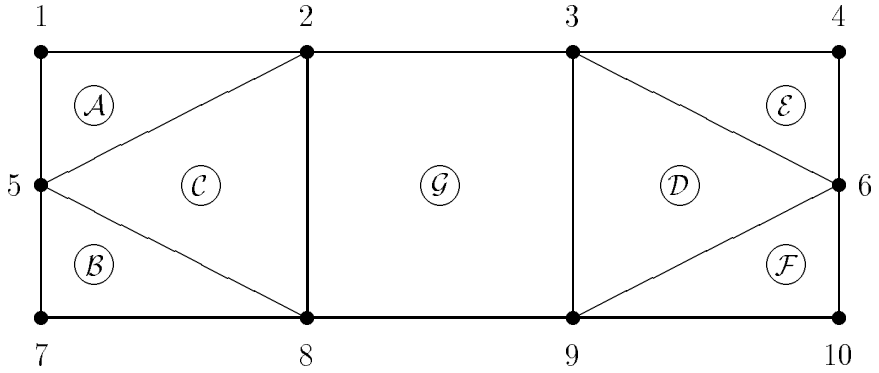
Matrices arising in finite-element applications are usually assembled from many small element matrices. Our Collection includes sparse matrices in original elemental form.[2]

[1] For example, some algorithms for reordering matrices break ties by relative position of indices. The orderings resulting from these algorithms may change if the order of entries is changed.

[2] The elemental representation in the Harwell-Boeing Sparse Matrix Collection applied only to square matrices generated from square elements with symmetric index patterns. Such matrices are either symmetric or structurally symmetric. The Rutherford-Boeing Sparse Matrix Collection also allows use of rectangular elements, resulting in an elemental form for general unsymmetric square matrices or rectangular matrices.

We illustrate the elemental representation using the finite-element problem shown in Example 2.

Example 2 *In the figure below, there are six triangular elements, labeled $\mathcal{A}$ through $\mathcal{F}$, and a quadrilateral element, labeled $\mathcal{G}$. The nodes are numbered from 1 through 10. We assume here that there is only one variable associated with each node, so the assembled matrix from this finite-element problem has order 10.*



A matrix can be associated with each element. Each has nonzeros only in the rows or columns associated with variables at the nodes of the given element. If we choose numerical entries for the nonzero entries in the element matrices to be 1.0 on the diagonal and -1.0 for the off-diagonals, then the element matrix corresponding to the element labeled $\mathcal{A}$ is the 10×10 matrix:

$$\begin{pmatrix}
1. & -1. & 0. & 0. & -1. & 0. & 0. & 0. & 0. & 0. \\
-1. & 1. & 0. & 0. & -1. & 0. & 0. & 0. & 0. & 0. \\
0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\
0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\
-1. & -1. & 0. & 0. & 1. & 0. & 0. & 0. & 0. & 0. \\
0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\
0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\
0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\
0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\
0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0.
\end{pmatrix} \tag{3.1}$$

It is much more convenient to represent the contribution from each element by a small dense matrix of order the number of variables in the element, together with a labeling to indicate the global index of the variables. Thus, the matrix associated with element $\mathcal{A}$ can be represented by the compact form or *element matrix* $E_{\mathcal{A}}$

$$E_{\mathcal{A}} = \begin{array}{c} \\ 1 \\ 2 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 5 \end{array} \\ \begin{pmatrix} 1. & -1. & -1. \\ -1. & 1. & -1. \\ -1. & -1. & 1. \end{pmatrix} \end{array}.$$

The remaining element matrices for elements $\mathcal{B}$ to $\mathcal{G}$ are then given by

$$E_{\mathcal{B}} = \begin{array}{c} \\ 5 \\ 7 \\ 8 \end{array} \begin{array}{c} \begin{array}{ccc} 5 & 7 & 8 \end{array} \\ \begin{pmatrix} 1. & -1. & -1. \\ -1. & 1. & -1. \\ -1. & -1. & 1. \end{pmatrix} \end{array}, \quad E_{\mathcal{C}} = \begin{array}{c} \\ 2 \\ 5 \\ 8 \end{array} \begin{array}{c} \begin{array}{ccc} 2 & 5 & 8 \end{array} \\ \begin{pmatrix} 1. & -1. & -1. \\ -1. & 1. & -1. \\ -1. & -1. & 1. \end{pmatrix} \end{array},$$

$$E_{\mathcal{D}} = \begin{array}{c} \\ 3 \\ 6 \\ 9 \end{array} \begin{array}{c} \begin{array}{ccc} 3 & 6 & 9 \end{array} \\ \begin{pmatrix} 1. & -1. & -1. \\ -1. & 1. & -1. \\ -1. & -1. & 1. \end{pmatrix} \end{array}, \quad E_{\mathcal{E}} = \begin{array}{c} \\ 3 \\ 4 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccc} 3 & 4 & 6 \end{array} \\ \begin{pmatrix} 1. & -1. & -1. \\ -1. & 1. & -1. \\ -1. & -1. & 1. \end{pmatrix} \end{array},$$

$$E_{\mathcal{F}} = \begin{array}{c} \\ 6 \\ 9 \\ 10 \end{array} \begin{array}{c} \begin{array}{ccc} 6 & 9 & 10 \end{array} \\ \begin{pmatrix} 1. & -1. & -1. \\ -1. & 1. & -1. \\ -1. & -1. & 1. \end{pmatrix} \end{array}, \quad E_{\mathcal{G}} = \begin{array}{c} \\ 2 \\ 3 \\ 8 \\ 9 \end{array} \begin{array}{c} \begin{array}{cccc} 2 & 3 & 8 & 9 \end{array} \\ \begin{pmatrix} 1. & -1. & -1. & -1. \\ -1. & 1. & -1. & -1. \\ -1. & -1. & 1. & -1. \\ -1. & -1. & -1. & 1. \end{pmatrix} \end{array},$$

The individual element matrices from the finite-element problem can be summed, using the expanded form of the element matrices as in (3.1), to produce an assembled matrix for the problem. The *assembled* coefficient matrix for Example 2 is

$$\begin{pmatrix} 1. & -1. & & & -1. & & & & & \\ -1. & 3. & -1. & & -2. & & & -2. & -1. & \\ & -1. & 3. & -1. & & -2. & & -1. & -2. & \\ & & -1. & 1. & & -1. & & & & \\ -1. & -2. & & & 3. & & -1. & -2. & & \\ & & -2. & -1. & & 3. & & & -2. & -1. \\ & & & & -1. & & 1. & -1. & & \\ & -2. & -1. & & -2. & & -1. & 3. & -1. & \\ & -1. & -2. & & & -2. & & -1. & 3. & -1. \\ & & & & & -1. & & & -1. & 1. \end{pmatrix}.$$

Note that our choice for the nonzero entries of the element matrices means that the magnitude of an entry in the assembled matrix is the number of element matrices that contributed to that entry.

While the linear algebra problem is completely defined by the assembled form of the problem, it is often possible to use the elemental representation to great advantage in devising or implementing algorithms to solve the finite-element problem. Therefore, we retain the original elemental form of such problems wherever it is available to us. In fact, the inverse process of disassembling an assembled matrix, not necessarily one from a finite-element problem, into an elemental form has been shown to be useful in some contexts, for example when exploiting partial separability in optimization applications.

We illustrate this disassembly process by Example 3 where we also generalize the element description to include matrices which are unsymmetric but whose nonzero pattern is symmetric, that is they are structurally symmetric.

Example 3 *We assume that we have the* 5×5 *structurally symmetric matrix*

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

that we choose to express as a sum of four unsymmetric element matrices, viz.

$$\begin{array}{c} \\ 1 \\ 4 \end{array}\begin{array}{c} \begin{array}{cc} 1 & 4 \end{array} \\ \begin{pmatrix} 2. & 3. \\ 1. & 7. \end{pmatrix} \end{array} \qquad \begin{array}{c} \\ 1 \\ 5 \end{array}\begin{array}{c} \begin{array}{cc} 1 & 5 \end{array} \\ \begin{pmatrix} 3. & 4. \\ 2. & 8. \end{pmatrix} \end{array} \qquad \begin{array}{c} \\ 2 \\ 3 \\ 5 \end{array}\begin{array}{c} \begin{array}{ccc} 2 & 3 & 5 \end{array} \\ \begin{pmatrix} 4. & 4. & 1. \\ 3. & 5. & 2. \\ 6. & 1. & 2. \end{pmatrix} \end{array} \qquad \begin{array}{c} \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{cc} 3 & 4 \end{array} \\ \begin{pmatrix} 2. & 6. \\ 8. & 2. \end{pmatrix}. \end{array}$$

We take advantage of the symmetry of the row and column index lists in our representation of such element matrices.

The use of an elemental expansion can be extended further through the use of rectangular element matrices. An example of a rectangular matrix and its elemental form is given as Example 4.

Example 4 *The matrix*

$$\begin{pmatrix} 1. & & 2. & & 3. \\ 4. & 5. & 6. & & 7. \\ & 8. & 9. & & 10. \\ 11. & & & 12. & \end{pmatrix}.$$

can be expressed as the sum of the element matrices

$$\begin{array}{c} \\ 1 \\ 2 \end{array}\begin{array}{c} \begin{array}{ccc} 1 & 3 & 5 \end{array} \\ \begin{pmatrix} 1. & 2. & 3. \\ 4. & 2. & 3. \end{pmatrix} \end{array} \qquad \begin{array}{c} \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{ccc} 2 & 3 & 5 \end{array} \\ \begin{pmatrix} 5. & 4. & 4. \\ 8. & 9. & 10. \end{pmatrix} \end{array} \qquad \begin{array}{c} \\ 4 \end{array}\begin{array}{c} \begin{array}{cc} 1 & 4 \end{array} \\ \begin{pmatrix} 11. & 12. \end{pmatrix}. \end{array}$$

Both the RB and MM formats store a matrix in elemental form as a list of element matrices. Each element matrix is itself represented by a list of the row/column indices (variables) associated with the nodes of the element and by a small dense matrix giving the numerical values. For symmetric and structurally symmetric matrices, a single list of indices suffices to give both the row and the column indices. Separate lists of row indices and column indices are required for rectangular element matrices.

In both RB and MM format the numerical values of an element matrix are given:

- in column major order for general matrices and structurally symmetric matrices;

- in column major order for only the lower triangular part for symmetric matrices.

That is, when C is a general $q \times r$ element matrix, its entries are held in the order

$$c_{11}, c_{21}, c_{31}, \ldots, c_{q1}, c_{12}, c_{22}, c_{32}, \ldots, c_{q2}, c_{13}, c_{23}, \ldots, c_{qr}.$$

Structurally symmetric elements are treated as general matrices, with $q = r$. When C is a symmetric $q \times q$ element matrix, its entries are held in the order

$$c_{11}, c_{21}, c_{31}, \ldots, c_{q1}, c_{22}, c_{32}, \ldots, c_{q2}, c_{33}, c_{43}, \ldots, c_{qq}.$$

The two file formats discussed in the next two chapters differ in how they store matrices in elemental form. The RB format stores all integer information separately from the numerical values, while the MM format intermixes integer and numerical data at an element matrix level.

The elemental representation and file formats have no intrinsic restrictions on the ordering of indices within element matrices or on the order of element matrices. Again to ensure that reproducibility is not compromised by inadvertent changes in order, matrices in elemental form from the Rutherford-Boeing Sparse Matrix Collection always present element indices in ascending order within each element matrix. The order in which the originating application supplied the element matrices may represent an important characteristic of the problem. We preserve this original order in the Rutherford-Boeing Sparse Matrix Collection. To make these problems more accessible for researchers unfamiliar with finite-element applications, we also hold an assembled form of the matrix for every elemental problem.

An important characteristic of element problems is that the largest integer used to index a variable can exceed the order of the matrix obtained if the elements are all assembled. A common example of this is found when the solution is required for only a substructure, but the numbering has been performed on the whole structure. A similar requirement occurs when using solution techniques based on domain decomposition. We use the finite-element problem of Example 2 to illustrate this. If we assume that the problem is divided into the three subproblems corresponding to elements $\mathcal{A}$ through $\mathcal{C}$, element $\mathcal{G}$, and elements $\mathcal{D}$ through $\mathcal{F}$ respectively, then the subproblem given by elements $\mathcal{D}$ through $\mathcal{F}$ could be represented in assembled form as a matrix of order 5 although the largest integer used as a variable index is 10. The canonical assembled form used in the Collection is generated by the index mapping that preserves relative index order. This would map the variables $\{3, 4, 6, 9, 10\}$ of this subproblem in order onto the set $\{1, 2, 3, 4, 5\}$; the resulting assembled matrix would be

$$\begin{pmatrix} 2. & -1. & -2. & -1. & \\ -1. & 1. & -1. & & \\ -2. & -1. & 3. & -2. & -1. \\ -1. & & -2. & 2. & -1. \\ & & -1. & -1. & 1. \end{pmatrix}.$$

2.4 Representations for Supplementary Data

The data supplementing the matrices as problem descriptions is exclusively in the form of dense or sparse vectors, or dense or sparse matrices. We treat a vector as a matrix with a single column. Thus we need only provide representations for sparse or dense matrices.

We already have general representations for sparse matrices, which we use for the special sparse matrices that represent sparse supplementary data. That is, sparse supplementary data is represented either in the compressed column representation for data stored in RB file format or in the coordinate representation for data stored in MM file format. This is discussed in detail in §3.2 and §4.2, respectively.

Dense data is represented by dense arrays. The representation in MM format uses the standard Matrix Market representation of dense matrices (§4.2). The RB format is described in §3.2. In both cases, the general rule is that the entries in a dense array are held in the file in column major order. That is, when B is a $q \times r$ dense matrix, its entries are held in the order

$$b_{11}, b_{21}, b_{31}, \ldots, b_{q1}, b_{12}, b_{22}, b_{32}, \ldots, b_{q2}, b_{13}, b_{23}, \ldots, b_{qr}.$$

Currently there is one exception to this rule, elemental right-hand sides in RB format. The ordering used for this data is described in §3.2.2.

Chapter 3

Rutherford-Boeing File Formats

In Chapter 2 we discussed two representations for sparse matrices, a general representation and a representation for finite-element applications. For each of these representations, we provide ASCII files for two different realizations of the matrices, the Rutherford-Boeing (RB) format and the Matrix Market (MM) format. In this chapter we discuss the RB format. The MM format is discussed separately in Chapter 4. The two formats are independent of one another, so this chapter and Chapter 4 are also independent of each other.

3.1 The Rutherford-Boeing Formats for Sparse Matrices

The RB format for the Rutherford-Boeing Sparse Matrix Collection is upward-compatible from the earlier format from the Harwell-Boeing Sparse Matrix Collection, which was designed originally for Fortran 66 codes. This means that a program that read HB matrices can be used to read data from the Rutherford-Boeing Sparse Matrix Collection held in RB format.[1] We add restrictions to the earlier format to facilitate the reading of matrices in the Collection from other languages. In such cases, some of the parameters in the file header need not be used. The RB format retains the restriction that all records or lines be no more than 80 characters long for portability.

If we define a data "block" as a contiguous set of 80 column records, delimited by record boundaries then each matrix file begins with a four-line header section 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. The header contains summary information on formatting and space requirements. From the header alone, the user can determine how much space will be required to store the matrix. Information on the number of lines for the representation is given to facilitate skipping past unwanted data.

The first line contains a 72-character descriptive title and also the 8-character identifier by which the matrix is referenced in our documentation. The 8-character identifier contains only upper case letters, digits and underscore characters. By convention, this identifier

[1] The current standard allows matrices of integer type, not included in the Harwell-Boeing Sparse Matrix Collection. This would require changing type declarations in the old code, probably through the same process as used to generate complex or double precision variants. Matrices generated from rectangular element matrices are an addition to the standard.

must also begin with an alphanumeric character. Within the file archives, the lower case translation of this identifier is used as the root name for all files related to this matrix. We prescribe no case rules for the title field, but we use normal English capitalization. We use only lower case characters in the remainder of the file, except in format descriptors.

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 section. The third line commences with a three character string denoting the matrix type. In the case of matrices in standard compressed column format, this is followed by the number of rows, columns, and entries. In the case of matrices in elemental form, the matrix type is followed by the largest integer used to index a variable, the number of element matrices, the total number of variable indices, and the total number of entries in the element matrices. The fourth line contains Fortran formats for the following data blocks. Both the second and fourth lines can be ignored by programs that use free format input. The exact format is given by the following, where the names of the variables used in the programs available from the Matrix Market are given in parentheses.

Line 1 (A72,A8)

Col. 1 - 72 Title (`TITLE`)

Col. 73 - 80 Matrix name / identifier (`MTRXID`)

Line 2 (I14, 3(1X,I13))

Col. 1 - 14 Total number of lines excluding header (`TOTCRD`)

Col. 16 - 28 Number of lines for pointers (`PTRCRD`)

Col. 30 - 42 Number of lines for row (or variable) indices (`INDCRD`)

Col. 44 - 56 Number of lines for numerical values (`VALCRD`)

Line 3 (A3, 11X, 4(1X,I13))

Col. 1 - 3 Matrix type (see below) (`MXTYPE`)

Col. 15 - 28 Compressed Column: Number of rows (`NROW`)
Elemental: Largest integer used to index variable (`MVAR`)

Col. 30 - 42 Compressed Column: Number of columns (`NCOL`)
Elemental: Number of element matrices (`NELT`)

Col. 44 - 56 Compressed Column: Number of entries (`NNZERO`)
Elemental: Number of variable indices (`NVARIX`)

Col. 58 - 70 Compressed Column: Unused, explicitly zero
Elemental: Number of element matrix entries (`NELTVL`)

Line 4 (2A16, A20)

Col. 1 - 16 Fortran format for pointers (PTRFMT)

Col. 17 - 32 Fortran format for row (or variable) indices (INDFMT)

Col. 33 - 52 Fortran format for numerical values of coefficient matrix (VALFMT)
(blank in the case of matrix patterns)

The first three characters on line 3 describe 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 in compressed column representation. The letter "a" appears as the descriptor for matrices in compressed column representation for historical reasons; it stands for assembled, used in contrast to the unassembled structure of matrices in elemental form.

First Character:

r real matrix

c complex matrix

i integer matrix

p pattern only (no numerical values supplied)

q pattern only (numerical values supplied in associated auxiliary value file)

Second Character:

s symmetric

u unsymmetric

h Hermitian

z skew symmetric

r rectangular

Third Character:

a compressed column form

e elemental form

The format data in Line 4 allows a Fortran program to use Fortran formatted

input/output for efficiency. This is compatible with the Harwell-Boeing Sparse Matrix Collection. The Rutherford-Boeing Sparse Matrix Collection allows easy access to the data from many languages by restricting the allowable Fortan formats to a very specific and simple set.

Data in each record of each data block is arranged in k fields, each of width w, where w is chosen large enough that there is at least one blank space between each datum and its neighbors. Each record requires no more than 80 characters of data. Floating-point decimal numerical entries always include an explicit decimal point; an exponent field, if present, has the form "E$\pm xy$" or "E$\pm xyz$", where x, y and z must each be a digit, not a blank.

Fortran programmers can use list-directed input, `READ (INPUT, *)` to read the data. C programmers can use `scanf`.

Fortran programs can also read, as character strings, the format data from Line 4, and use these as format specifications for subsequent reads of the matrix data. Only formats of the form `(kI`w`)` for integer data and `(kE`$w.d$`)` for floating-point data are allowed in the Collection. For example, the column start pointers for a matrix with $250,000,000$ nonzeros would be represented using `(8I10)` format; the standard representation for reals is `(3E25.16)`.

3.1.1 The Rutherford-Boeing Compressed Column Format

The structure of a file in RB compressed column format is

header
block of pointers to the beginning of columns
block of row indices
block of numerical values

In this structure, the matrix used in Example 1,

$$A = \begin{pmatrix} 1. & -4. & & -8. & \\ & & -6. & & 10. \\ 2. & & & & \\ & 5. & & -9. & \\ 3. & & -7. & & 11. \end{pmatrix},$$

would be represented by the vectors `COLPTR` (location of first entry), `ROWIND` (row indices), and `VALUES` (numerical values) given by:

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.	3.	-4.	5.	-6.	-7.	-8.	-9.	10.	11.

A complete file representation of this example is

```
Small general matrix used as Example 1                                  EXAMPLE1
             8             2             3             3
rua                        5             5            11
(5I5)           (5I5)           (5E10.1)
    1    4    6    8   10
   12
    1    3    5    1    4
    2    5    1    4    2
    5
       1.0       2.0       3.0      -4.0       5.0
      -6.0      -7.0      -8.0      -9.0   1.0E+01
   1.1E+01
```

In the Rutherford-Boeing Sparse Matrix Collection, we restrict the order in which entries of the matrices are presented. We now require that the row indices within each column be given in strict ascending order.

3.1.2 The Rutherford-Boeing Elemental Format

The structure of the file in RB elemental format is

header
block of pointers to the beginning of elements in the variable index list
block of variable indices
block of numerical values

In this structure, the structurally symmetric matrix used in Example 3, in §2.3, whose element matrices were:

$$\begin{array}{c} \\ 1 \\ 4 \end{array}\!\!\begin{array}{c} \begin{array}{cc} 1 & 4 \end{array} \\ \begin{pmatrix} 2. & 3. \\ 1. & 7. \end{pmatrix} \end{array} \qquad \begin{array}{c} \\ 1 \\ 5 \end{array}\!\!\begin{array}{c} \begin{array}{cc} 1 & 5 \end{array} \\ \begin{pmatrix} 3. & 4. \\ 2. & 8. \end{pmatrix} \end{array} \qquad \begin{array}{c} \\ 2 \\ 3 \\ 5 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} 2 & 3 & 5 \end{array} \\ \begin{pmatrix} 4. & 4. & 1. \\ 3. & 5. & 2. \\ 6. & 1. & 2. \end{pmatrix} \end{array} \qquad \begin{array}{c} \\ 3 \\ 4 \end{array}\!\!\begin{array}{c} \begin{array}{cc} 3 & 4 \end{array} \\ \begin{pmatrix} 2. & 6. \\ 8. & 2. \end{pmatrix} \end{array}.$$

would be stored in the arrays ELTPTR (location of first entry), VARIND (variable indices), and VALUES (numerical values) as:

Subscripts	1	2	3	4	5
ELTPTR	1	3	5	8	10

Subscripts	1	2	3	4	5	6	7	8	9
VARIND	1	4	1	5	2	3	5	3	4

Subscripts	1	2	3	4	5	6	7	8	9	10	11
VALUES	2.	1.	3.	7.	3.	2.	4.	8.	4.	3.	6.
Subscripts	12	13	14	15	16	17	18	19	20	21	
VALUES	4.	5.	1.	1.	2.	2.	2.	8.	6.	2.	

A complete Rutherford-Boeing file representation of this example is:

```
Small matrix in elemental form                                          EXAMPLE3
             8             1             2             5
rue                        5             4             9            21
(5I5)           (5I5)           (5E10.1)
    1    3    5    8   10
    1    4    1    5    2
    3    5    3    4
       2.0       1.0       3.0       7.0       3.0
       2.0       4.0       8.0       4.0       3.0
       6.0       4.0       5.0       1.0       1.0
       2.0       2.0       2.0       8.0       6.0
       2.0
```

This representation does not hold the pointers to the beginning of the numerical values for each element matrix. Because there is not a 1-1 correspondence between indices and numerical values, the pointers to the numerical values are not the same as the pointers to the indices given in `ELTPTR`. The pointers to the numerical values can, however, be created from the index start pointers (`ELTPTR`) by computing and summing the number of values in each of the elements. For example, an unsymmetric or structurally symmetric element matrix with ν variables has ν^2 numerical values [$(\nu \times (\nu+1))/2$ in the symmetric case]. In the example above, the elements have 4, 4, 9 and 4 values, respectively, and the pointers to the first numerical value in each element would be given by 1, 1+4 = 5, 1+4+4 =9, and 1+4+4+9 = 18, respectively. Note that, although the order in which elements are held is that given by the application, the indices within each element are always held in order.

Rectangular element matrices require two sets of indices, one for rows and another for columns. The RB file format for rectangular matrices in elemental form maintains compatibility with the symmetric and structurally symmetric representations by representing both sets of indices in a single element index vector. If the set of row variable indices for the ith of `NELT` elements is denoted by $\mathcal{R}_i$ and the set of column variable indices denoted similarly by $\mathcal{C}_i$, the structure of the elemental index vector is:

$$\mathcal{R}_1 \quad \mathcal{C}_1 \quad \mathcal{R}_2 \quad \mathcal{C}_2 \quad \ldots \quad \mathcal{R}_i \quad \mathcal{C}_i \quad \ldots \quad \mathcal{R}_{\texttt{NELT}} \quad \mathcal{C}_{\texttt{NELT}} \; .$$

The index start pointer vector `ELTPTR` for rectangular elemental matrices is of length $2 \cdot$ `NELT+1`. The beginning of the set $\mathcal{R}_i$ of row variable indices is given by the entry in position $2i-1$ of vector `ELTPTR`; the start of $\mathcal{C}_i$ is in position $2i$ of `ELTPTR`. The total number of indices is given by `NVARIX` in the header block. The last entry of `ELTPTR` contains `NVARIX`+1, so that the number of row variable indices for the ith element is given by `ELTPTR(2*i) - ELTPTR(2*i-1)` and the number of column variable indices by `ELTPTR(2*i+1) - ELTPTR(2*i)`.

The data stored in the header section of the rectangular elemental representation suffices for storage allocation of the matrix data. It does not include explicitly all important matrix characteristics. The number of columns represented in the matrix must be obtained by scanning the column variable indices in the index array. The address of the beginning of the ith element in `VALUES` must be calculated by summing the sizes of the preceding elements. Codes that perform these functions as part of the process of reading a matrix in elemental form are available with the Collection.

As an example of the rectangular elemental matrix representation, we use Example 4 from §2.3. The element matrices were:

$$\begin{array}{c} \\ 1 \\ 2 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} 1 & 3 & 5 \end{array} \\ \begin{pmatrix} 1. & 2. & 3. \\ 4. & 2. & 3. \end{pmatrix} \end{array} \qquad \begin{array}{c} \\ 2 \\ 3 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} 2 & 3 & 5 \end{array} \\ \begin{pmatrix} 5. & 4. & 4. \\ 8. & 9. & 10. \end{pmatrix} \end{array} \qquad \begin{array}{c} \\ 4 \end{array}\!\!\begin{array}{c} \begin{array}{cc} 1 & 4 \end{array} \\ \begin{pmatrix} 11. & 12. \end{pmatrix}. \end{array}$$

and would be represented by the RB elemental matrix file

```
Small rectangular matrix in elemental form                              EXAMPLE4
             8             2             3             3
rre                        5             3            13            14
(5I5)           (5I5)           (5E10.1)
    1    3    6    8   11
   12   14
    1    2    1    3    5
    2    3    2    3    5
    4    1    4
       1.0       4.0       2.0       2.0       3.0
       3.0       5.0       8.0       4.0       9.0
       4.0      10.0      11.0      12.0
```

For each matrix held in elemental form, we also hold the assembled matrix in which the row and column variable indices in the elemental form are mapped onto the sets $\{1, 2, 3, \ldots, u\}$ and $\{1, 2, 3, \ldots, v\}$, where u and v are the number of distinct row and column variable indices, respectively, that actually occur in the element matrices. Thus u and v are the row and column dimensions of the assembled matrix. In each case the index mapping preserves the order of indices, while eliminating any gaps in the sequence.

3.2 Rutherford-Boeing Formats for Supplementary Data

Usually the sparse matrix represents only part of the problem. When available, we include additional data that more completely describes the linear algebra problem. Supplementary data for which we have defined specific representations are given in Table 1, replicated from §1.3. These data are stored in files separate from the main matrix file. Each can be viewed as a rectangular matrix in dense, sparse or elemental form. We have intentionally kept these file structures simple. This means that complicated problem descriptions will be realized by multiple supplementary files.

The general outline of the supplementary RB data format is given below. Each file begins with a three line header block followed by the numerical data in one, two or three data blocks. The various header parameters have different, but related, interpretations for each type of data represented in the Collection.

Where the supplementary data is sparse, the data blocks that follow the header are structured identically to an RB compressed column sparse matrix. Usually there are three data blocks. In some cases, for example index partitionings, a sparse matrix pattern suffices to represent the supplementary data, in which case there will be only two data blocks.

Most supplementary cases are represented by dense data. When the numerical data represents an $m \times q$ dense matrix C, the numerical data form a single data block following

Table 1: Supplementary Files

keyword	*extension*		
	type	*position*	*organization*
orderings	ord	[l|r|s]	
right-hand-sides	rhs	[l|r]	[e|d|s]
solutions	sln	[l|r|s]	
estimates	est	[l|r|s]	
eigenvalues	evl		
singular-values	svl		
eigenvectors	evc	[l|r|s]	
singular-vectors	svc	[l|r|s]	
Schur-basis-vectors	sbv		
Schur-basis-matrix	sbm		
Schur-basis-parameters	sbp		
partition	ipt	[l|r|s]	
covering	icv	[l|r|s]	
Laplacian-values	lvl		
Laplacian-vectors	lvc		
geometry	geo		
auxiliary-values	avl		

the header and are simply the entries of the matrix listed in column major order, viz.

$$c_{11}, c_{21}, c_{31}, \ldots, c_{m1}, c_{12}, c_{22}, c_{32}, \ldots, c_{m2}, c_{13}, c_{23}, \ldots, c_{mq}.$$

Dense data representing a problem in elemental form have a more complicated format, at present used only for elemental form right-hand sides, described in 3.2.2.

The numerical entries are formatted so that they can be read using Fortran free-format or C `scanf`. Whether the data is real, complex or integer is specified by the parameter `NUMERF` in the header. This parameter plays the same role as the first character of `DATTYP` in a matrix header. Supplementary data in the form of sparse matrix patterns is indicated by `p`.

Some types of supplementary files do not require all of the identifiers in the header block. When the character data `POSITN, ORGNIZ, AUXFM2` and `AUXFM3` are not required, they are blank. When the integer parameter `NAUXD` is unused, it holds an explicit zero to facilitate reading in `C`.

Line 1 (A72, A8)

Col. 1 - 72 Description (`TITLE`)
(describes association of data with appropriate matrix)

Col. 73 - 80 Matrix name (`MTRXID`)
(matches the name/identifier of associated matrix or related matrices)

Line 2 (A3, 2A1, 1X, A8, 1X, A1, 3(1X, I13))

Col. 1 - 3 Data type (`DATTYP`)
(matches file extension, contains no embedded blanks)

Col. 4 - 4 Position qualifier (`POSITN`)
(`l, r, s`; blank when field is unused)

Col. 5 - 5 Data organization qualifier (`ORGNIZ`)
(`d, e, s`; blank when field is unused)

Col. 7 - 14 Case identifier (`CASEID`)
(used to distinguish multiple sets of supplementary data)

Col. 16 - 16 Numerical field (`NUMERF`)
(denotes the data type for the numerical data)

Col. 18 - 30 Row dimension of vector data (`M`)

Col. 32 - 44 Number of vectors (`NVEC`)

Col. 46 - 58 Supplementary integer parameter (`NAUXD`)
(interpreted specifically for each data type; zero when not used)

Line 3 (3A20)

Col. 1 - 20 First Fortran format for supplementary data (`AUXFM1`)

Col. 21 - 40 Second Fortran format for supplementary data (`AUXFM2`)

Col. 41 - 60 Third Fortran format for supplementary data (`AUXFM3`)

We use the same rules on case and Fortran formats for supplementary files as for matrix files. Alphabetic characters in the matrix name and case identifier (`MTRXID` and `CASEID`) are uppercase, the title has no restrictions on case, and all other alphabetic characters in the file are lower case, except the format descriptors. The Fortran data formats are of the form (`kI`w) for integer data and (`kE`$w.d$) for floating-point data.

Note that the first two lines provide, in an encoded fashion, the name of the file in the netlib Rutherford-Boeing archive; it is given by the lower case translation of `MTRXID.`*ext*`.CASEID.`*ff*, where *ext* is the concatenation of `DATTYP, POSITN` and `ORGNIZ`, as given in Table 1, and *ff* is *mm* or *rb*.

3.2.1 Orderings (DATTYP = ord)

These files provide permutations that represent interesting or unusual reorderings of sparse matrices. The permutations are given by integer mappings which map the original (old) labeling of the matrix as stored in the Collection to its new labeling. Thus, in the symmetric permutation case, the reordered matrix $\widehat{A}$ is represented mathematically by $\widehat{A} = PAP^T$. If the permutation vector is denoted by p, the entry-wise relationship is $\widehat{A}_{p(i),p(j)} = A_{ij}$. The unsymmetric and rectangular cases may require both a left and a right permutation P and Q, with the reordered matrix given by $\widehat{A} = PAQ$. If the stored permutation vectors are denoted by p and q, the entry-wise relationship is $\widehat{A}_{p(i),q(j)} = A_{ij}$. Such general orderings are represented in the Collection by a pair of files, one giving P and the other Q.[2]

The position qualifier `POSITN` is defined as:

`s`	if the single permutation P should be applied symmetrically
`l`	if the file represents the left permutation P
`r`	if the file represents the right permutation Q

Note that unsymmetric matrices can be permuted symmetrically and unsymmetric permutations of symmetric matrices are also valid. In the case of permutations for rectangular matrices, the left and right permutation vectors are of different lengths.

Permutations are held as integer vectors in dense format. Multiple permutations are stored as a dense matrix. The header parameters are:

Permutations	
`DATTYP = ord`	
`POSITN = s` or `l` or `r`	
`NUMERF = i`	
`M`	row dimension of permutation vector(s)
`NVEC`	number of permutations
`NAUXD`	unused
`AUXFM1`	format for integer values in permutation vector(s)
`AUXFM`i, $i = 2, 3$	unused

Example 5 *This file contains two symmetric permutations for the assembled matrix for Example 3.*

```
Symmetric orderings for a five by five matrix                    EXAMPLE3
ords  TWOPERMS i            5            2
(5I5)
    5    3    4    2    1
    4    3    5    1    2
```

[2] The two permutations are not encoded in a single file because the interpretation of the data as a vector or a matrix is awkward when the matrix is rectangular.

Each permutation is given as a column of a 5×2 integer matrix. The first permutation is the vector $p^{(1)} = [5, 3, 4, 2, 1]^T$ and the second permutation is $p^{(2)} = [4, 3, 5, 1, 2]^T$. The qualifier `S` indicates that each permutation is to be used symmetrically.

What permutations are available for a given matrix is specified in the documentation file (*matrixid*.doc) for the given matrix *matrixid*. All files for a given matrix can be accessed together automatically by the Matrix Market when the standard WWW interface is used to retrieve files.

Orderings for a matrix in elemental form are given as permutations of the elements, not of the variables. Thus the row dimension of the permutation(s) is the number of elements.

3.2.2 Right-Hand Side Vectors (DATTYP = rhs)

When the matrix originates in the solution of linear equations, and when the right-hand sides of the linear systems are available, we store the right-hand sides in a separate supplementary file. (Matrices with right-hand sides from the Harwell-Boeing Sparse Matrix Collection have been replicated, with the right-hand sides stored in separate supplementary files in the Rutherford-Boeing Sparse Matrix Collection. Data in the HB format is available from the Harwell-Boeing Sparse Matrix Collection Archive.)

We can include right-hand sides b for the systems $Ax = b$ and also for the transposed systems $A^T x = b$. The position qualifier `POSITN` distinguishes between the two, and is interpreted as:

`POSITN = r`	right-hand side b for $Ax = b$
`POSITN = l`	right-hand side b for $A^T x = b$.

Right-hand sides are also available for least-squares problems.

Usually right-hand sides are not sparse. We store a typical right-hand side as a dense vector whose order matches the appropriate order of the associated matrix. We permit two other representations for right-hand sides: an elemental representation of the right-hand sides for matrices in elemental form and a sparse representation if the right-hand sides are sparse. (At present, only the representation of right-hand sides use multiple representations. This section is therefore the most complicated of these for supplementary data.)

If the matrix A has order M by N, the three cases are distinguished by the organization qualifier `ORGNIZ`, defined as:

- `d` if the right-hand sides are ordinary **d**ense vectors of order M or N as `POSITN` is `r` or `l` respectively
- `s` if the right-hand sides are stored as **s**parse form
- `e` if the right-hand sides are dense vectors corresponding to the **e**lemental structure of the associated matrix.

Each case has its own interpretation of the header parameters. The general form of the header fields for right-hand sides is:

Right-hand sides	
DATTYP = rhs	
POSITN = r or l	
ORGNIZ = d or s or e	
NUMERF = r or c or i	
M	row dimension of right-hand side(s)
NVEC	number of right-hand sides
NAUXD	total number of numerical entries in all right-hand sides
when ORGNIZ = d or e	
AUXFM1	format for numerical values of right-hand side(s)
AUXFMi, $i = 2, 3$	unused
when ORGNIZ = s	
AUXFM1	format for column pointers for right-hand side(s)
AUXFM2	format for row indices for right-hand side(s)
AUXFM3	format for numerical values of right-hand side(s)

Dense Right-Hand Sides

The most common case is that of dense vectors, for which `ORGNIZ = d`. A single right-hand side is represented by `M` numerical entries following the header in the format described by `AUXFM1`. (As with matrices, the actual formats allow free-format input as well as formatted input.) Where the data represents multiple vectors, the vectors are stored as an `M` $\times$ `NVEC` dense array. In this case, the variable `NAUXD` holds the redundant value `M` $\times$ `NVEC` for compatibility with the other right-hand side representations. Note that the value `M` should match the row (or column in the case of `POSITN` = `l`) dimension of the matrix whose identifier is given by *matrixid*.

Example 6 *The following file represents two right-hand sides for a* 5×5 *matrix, where the first right-hand side is the vector* $[0, 0, 3, 4, 0]^T$ *and the second is* $[0, 0, 0, 0, 1]^T$ *:*

```
Two right-hand sides for a five by five matrix                    EXAMPLE3
rhsrd FULL_R2  r             5             2            10
(5E10.1)
      0.0       0.0       3.0       4.0       0.0
      0.0       0.0       0.0       0.0       1.0
```

Sparse Right-Hand sides

Applications with sparse right-hand sides are less common, but the sparsity can be exploited by direct solution techniques. We store each sparse right-hand side in the same format as a column of a general sparse matrix. Thus multiple right-hand sides are represented by a standard sparse matrix, with as many rows as the coefficient matrix and as many columns as there are right-hand sides. In this case, `ORGNIZ` is set to `s` and the header block will be followed by the three data blocks that represent a standard sparse

matrix: column pointers, row indices and numerical values. The number of row indices is equal to the number of numerical entries, NAUXD.

Example 7 *The following file represents the two sparse right-hand side vectors* $[0,0,3,4,0]^T$ *and* $[0,0,0,0,1]^T$*, which represent the same right-hand side as in Example 6:*

```
Two sparse right-hand sides for a five by five matrix                  EXAMPLE3
rhsrs SPARS_R2 r             5             2             3
(5I5)           (5I5)           (5E10.1)
    1    3    4
    3    4    5
       3.0       4.0       1.0
```

Note that our convention for ordering the indices in sparse matrices applies throughout the collection. Thus, within each column (right-hand side) the row indices appear in ascending order.

Elemental Right-Hand Sides

Normally applications in which the sparse matrix is in elemental form use the same elemental structure to represent the right-hand sides. Elemental right-hand sides, ORGNIZ = e, are only supplied when the associated matrix is in *elemental* form. The elemental structure of the right-hand sides must be interpreted through the elemental structure that describes the matrix.

A single elemental right-hand side is given by a list of q values $c_k, k = 1, q$, where $q =$ NVARIX is the sum of the number of variables in individual element matrices. Each entry c_k is a contribution to some entry in the right-hand side b of the assembled system $Ax = b$. Specifically, we list the entries of c in order corresponding to the order in which the (row) indices appear in the elemental representation of the matrix. Thus the list is a concatenation of contributions from individual elements with those from element i immediately preceding those from element $i+1$. For square matrices the array c exactly mirrors the array of indices from the elemental representation, IND say, so that the right-hand side b is defined by $b_j = \sum \{c_k | \texttt{IND}_k = j\}$.

For multiple right-hand sides, we store all right-hand side contributions from each element contiguously so that, for example, if we have two right-hand sides, the entries for the first element contribution are stored as $\{c_{11}, c_{21}, \ldots c_{n_1 1}, c_{12}, c_{22}, \ldots c_{n_1 2}\}$, with n_1 the number of indices in element 1. Note that this is different from the way in which other multiple sets of data are stored, where each member of the set is stored contiguously.

The variable NAUXD in the header block gives the total number of values that contribute to all the right-hand sides. This value must equal NVEC times the value of NVARIX of the corresponding matrix.

Example 8 *The following file gives two right-hand sides for the matrix in Example 3. Confirming that the right-hand sides are the same as the right-hand sides in Example 6 will test your understanding of the representation:*

```
Two elemental right-hand sides for Example 3                          EXAMPLE3
rhsre ELMNT_R2 r             5             2            18
(5E10.1)
      -1.0       2.0       2.0       1.0       1.0
       5.0      -2.0       0.0       0.0       0.0
      -5.0       0.0      -3.0       1.0       3.0
       2.0       3.0      -1.0
```

3.2.3 Solution Vectors (DATTYP = sln) and Estimates of Solution (DATTYP = est)

We always hold solution vectors and estimates of the solution in dense format. An estimate of the solution may be available in the originating application and may be used, for example, as a starting guess for an iterative solution scheme. As with right-hand sides, each vector is stored contiguously and vectors are stored consecutively. In both cases, the interpretation of the header parameters is:

Solution Vectors or Solution Estimates	
DATTYP = sln or est	
POSITN = l or r	
NUMERF = r or c or i	
M	row dimension of solution vector(s)
NVEC	number of solution vectors
NAUXD	unused
AUXFM1	format for numerical values
AUXFMi, $i = 2, 3$	unused

The position qualifier has the same meaning as with right-hand sides. In particular the position qualifier should match the position qualifier in the file of right-hand sides for which this data claims to be a solution or an estimate of the solution.

3.2.4 Eigenvalues (DATTYP = evl), Singular Values (DATTYP = svl), and Eigenvalues of Laplacian (DATTYP = lvl)

These files provide a list of some or all of the eigenvalues or singular values for a particular eigenproblem, or the eigenvalues of the Laplacian matrix[3] associated with a matrix in the Collection. In each case, only a single dense vector is needed, so the interpretation of the format is very simple.

[3]The Laplacian of a (structurally) symmetric sparse matrix A is another sparse matrix, L_A, with the same sparsity pattern as A. It has off-diagonal entries with the value -1 in exactly the same positions as the off-diagonal entries of A. Each diagonal of the Laplacian is set to the number of off-diagonal entries in the corresponding row of A; thus each diagonal entry of L_A is the modulus of the sum of the off-diagonals in its row. The eigenvalues and eigenvectors of L_A provide useful graph characterizations of the sparsity pattern of A.

Eigenvalues and Singular Values DATTYP = evl or svl or lvl NUMERF = r or c	
M	number of values
NVEC	1
NAUXD	unused
AUXFM1	format for numerical values
AUXFMi, $i = 2, 3$	unused

A given eigenproblem may be accompanied by several eigenvalue supplementary files. These may represent variations in accuracy, in choice of subset of eigenvalues or to match different subsets of eigenvectors in linked supplementary files. Different approximations to the same eigenpair may appear in more than one set of these files. An eigenvalue file may be associated with an eigenvector file. If this is the case, then the number of eigenvalues and the number of eigenvectors are equal.

3.2.5 Eigenvectors (DATTYP = evc), Singular Vectors (DATTYP = svc) and Laplacian Vectors (DATTYP = lvc)

These files hold some of the eigenvectors or singular vectors for a particular eigenproblem. In the Rutherford-Boeing Sparse Matrix Collection each eigenvector or singular vector file is always associated with a file containing the corresponding eigenvalues or singular values; values associated with Laplacian vectors are not always held. Only in trivially small cases will all the eigenvectors or singular vectors be available. Generally only subsets of the eigenvectors will be held. Since the eigenvectors of sparse matrices are not usually sparse, the vectors are stored as dense vectors or matrices. The eigenvectors of general real matrices are stored as complex vectors.

Which sets of vectors are present is indicated by POSITN, which is defined as:

- s if symmetry allows the vectors to represent both left and right eigensolutions
- l if the file contains the left vectors of a general eigenproblem or the left singular vectors
- r if the file contains the right vectors of a general eigenproblem or the right singular vectors

The data format is again very simple:

Eigenvectors, Singular Vectors and Laplacian Vectors DATTYP = evc or svc or lvc POSITN = s or l or r NUMERF = r or c	
M	dimension of eigenvectors or singular vector(s)
NVEC	number of vectors
NAUXD	unused
AUXFM1	format for numerical values
AUXFMi, $i = 2, 3$	unused

3.2.6 Schur Bases: Schur basis vectors (DATTYP = sbv), Schur basis matrix (DATTYP = sbm) and Schur basis parameters (DATTYP = sbp)

The matrix of eigenvectors of an unsymmetric matrix (or the solutions of other general eigenproblem) can be arbitrarily badly conditioned. An alternative representation with better numerical properties is the Schur decomposition (or generalizations thereof). The Schur decomposition is given by $AU = UT$, where U is unitary and T is complex upper triangular. When A is real, the real Schur decomposition is given by $AU = UT$, where U is real orthogonal and T is real quasi upper triangular, that is, a block triangular matrix with diagonal blocks of order one and two. The 1×1 and 2×2 blocks on the diagonal of T give the real eigenvalues and the complex eigenvalues (as a complex conjugate pair) respectively. Schur decompositions for generalized eigenproblems can be built from several orthogonal or unitary matrices and several triangular or quasi-triangular matrices.

A partial Schur decomposition is simply the leading columns of a (complete) Schur decomposition. Schur bases files hold partial Schur decompositions for ordinary eigenproblems and for some generalized problems. In the Rutherford-Boeing Sparse Matrix Collection each Schur basis is represented by two or three types of files. Two are always required: one holds the vectors U as a dense matrix and the second holds the (block) triangular matrix. The first is the "Schur-basis-vectors" file; the second is the "Schur-basis-matrix" file. The latter is stored as a square dense matrix which includes the zero entries in the lower triangular part.

The third file is used to help accurately represent Schur bases that were obtained using a transformation of the spectrum of the eigenproblem. Certain transformations are in common use as very powerful accelerators for sparse eigensolution algorithms, especially for generalized eigenproblems. We allow representations for cases where the transformation is a rational function of the matrix arguments of the problem, T and I. In these cases we hold the Schur basis matrix for the transformed problem because the transformation back to the original problem may be badly conditioned. In addition we hold the coefficients of the rational function. We do not attempt a general representation here; the third file, the "Schur-basis-parameters" file is simply a vector of coefficients. The form of the rational function is given in the documentation of the problem and so the coefficients can only be interpreted with the documentation.

The header format is straight-forward:

Schur Basis Vectors, Matrix and Parameters DATTYP = sbv or sbm or sbp NUMERF = r or c	
M	dimension of Schur basis (DATTYP = sbv) number of vectors (DATTYP = sbm) number of parameters (DATTYP = sbp)
NVEC	number of vectors (DATTYP = sbv or sbm) 1 (DATTYP = sbp)
NAUXD	unused
AUXFM1	format for numerical values
AUXFMi, $i = 2, 3$	unused

3.2.7 Index Partitions and Coverings (DATTYP = ipt or icv)

These files represent partitions or coverings of the row and/or column indices of sparse matrices. A partition is a set of S subsets $\{I_k, k = 1, 2, \ldots, S\}$ of the NROW row indices or the NCOL column indices, such that each index is assigned to one particular subset. Thus a partition can be represented as a map from the index number i to the to the index k of the single subset I_k that contains it. A covering of the row (or column) indices is a set of subsets of the indices, $\{I_k, k = 1, S\}$, where S is the number of subsets, such that every index appears in *at least one* subset of indices I_k. Coverings can be used to describe an underlying elemental structure for a matrix or to describe hierarchical or overlapping node partitionings as in domain decompositions. Clearly the difference between a partition and a covering is that an index in a partition is assigned to a unique subset, whereas it can be a member of several subsets in a covering. We use the same file format to represent both.

The index sets I_k are represented as a sparse matrix pattern with M rows and $S =$ NVEC columns. That is, each subset in the partition or covering is given by the list of indices that appear as a column of this sparse matrix. The structure of the pointer vector is the same as to that used for storing the columns of a sparse matrix. In this content the kth pointer gives the index of the first entry in subset k.

The field POSITN is defined as for ordering files:

- s if the single partition or covering should be applied symmetrically
- l if the file represents a partition or covering of the row indices
- r if the file represents a partition or covering of the column indices

The header parameters are:

Partitions
DATTYP = ipt or icv POSITN = s or l or r NUMERF = p

M	number of indices partitioned or covered
NVEC	number of subsets in partition or covering (S)
NAUXD	total number of entries in all partition/covering sets
AUXFM1	format for integer values in pointer vector
AUXFM2	format for integer values in index list vector
AUXFM3	unused

A file of this type holds only a single index partition or covering. General or unsymmetric partitionings or coverings will be represented by a pair of files. The representation described above does not prohibit empty subsets in a partition, which would be indicated by consecutive pointers having the same value. However, the convention used in the Collection is that partitions will not have empty subsets.

3.2.8 Geometric (Coordinate) Data (DATTYP = geo)

Often problems, and matrices, arise in a physical or geometric setting. In such cases there may be natural mappings between algebraic variables in the model and coordinates in a d-dimensional physical or Euclidean space. For example, the ith row of a matrix may describe the state equation that holds at a particular point $< x_i, y_i, z_i >$ in 3-space. The geometric coordinate data files capture this relationship.

Each geometry file represents a mapping between k points in a d-dimensional physical space and the row and/or column indices of the matrix. The geometric coordinate data give the locations of the k points in the d-dimensional physical space as a dense $k \times d$ array. Whether the geometric data describes the row vertices of the matrix, the column vertices, or both, is specified by the value of POSITN.

In the simplest case there is a 1-to-1 mapping between geometric points and algebraic variables; the number k of physical points is the same as the row and/or column dimensions, NROW or NCOL, of the associated matrix.

There are, however, more complicated cases. Often a model uses several variables to represent different characteristics at each of a set of physical points. For example, in fluid flow, there may be variables corresponding to velocity in each of three directions, pressure and temperature at each point in 3-dimensional physical space. If these five variables are modelled at each point, the dimensions of the matrix are five times the number of physical points. Representing geometric data directly for each variable would result in redundant data. We permit a more concise representation, which is sufficiently general to allow models that do not have the same number of variables at each physical point. Further, the representation identifies the set of variables corresponding to each particular common location. In the Rutherford-Boeing Sparse Matrix Collection we represent such situations with two files. One is a *.geo?.* geometry file, containing the geometric locations as a $k \times d$ array, where k is the true number of physical points. The second file contains

the nodal partition that labels all variables at a given physical point as a single subset; this will be an `*.ipt?.*` file, partitioning `NROW` or `NCOL` indices. The number of partition subsets in the index partition file, S, must equal k. The mapping between the two files is such that the locations in the ith row of the array are the locations of all variables in subset i in the partition.

The field `POSITN` is defined in the usual way:

- `s` if the geometry data applies symmetrically to row and column vertices
- `l` if the geometry represents the left/row vertices
- `r` if the geometry represents the right/column vertices

The data format is thus:

Geometric Data	
`DATTYP = geo`	
`POSITN = s` or `l` or `r`	
`NUMERF = r` or `c` or `i`	
`M`	number of geometric or physical points (k)
`NVEC`	geometric dimensionality (d)
`NAUXD`	unused
`AUXFM1`	format for coordinates of numerical values
`AUXFM`i, $i = 2,3$	unused

3.2.9 Auxiliary Matrix Values (`DATTYP = avl`)

Other kinds of information may be associated with a sparse matrix. Sparse matrices are often associated with graphs; for example, each row or column can represent a vertex in the graph and each entry in the matrix can correspond to an edge in the graph. In some contexts, a graph may have several values attached to each vertex or to each edge. For example, network flow problems may have costs and capacities for each edge. In such cases, rather than representing the single problem by multiple matrices sharing a common sparsity pattern, we can use supplementary files associated with a single matrix.

Auxiliary values may also be associated with the vertices. A geometric description of a problem may, for instance, associate some sort of nodal type with each vertex. For example, a modeler may distinguish between nodes on the the boundary of the physical graph and nodes in the interior. Such labeling can easily be accomplished by an integer vector of length the number of rows and/or columns.

We expect auxiliary value files to contain quite specialized, problem-dependent, data. To be flexible we only require that they be vectors or arrays in dense form, with the dimensions and structure described in text in the documentation for the associated matrix. For example, a Schur basis parameters file, described in §3.2.6, is a particular, now named, example of a quite general use of an auxiliary value file.

The interpretation of the header parameters is:

Auxiliary Matrix Values	
DATTYP = avl	
NUMERF = r or c or i	
M	row dimension of vector data
NVEC	number of auxiliary vectors
NAUXD	unused
AUXFM1	format for numerical values
AUXFMi, $i = 2, 3$	unused

Chapter 4

Matrix Market Formats for Rutherford-Boeing Files

In Chapter 2 we discussed two representations for sparse matrices, a general representation and a representation for finite-element applications. For each of these representations, we provide ASCII files for two different realizations of the matrices, the Rutherford-Boeing Sparse Matrix Collection (RB) format and the Matrix Market (MM) format. In this chapter we discuss the MM format. The RB format is discussed separately in Chapter 3. The two formats are independent of one another, so this chapter and Chapter 3 are also independent of each other.

A description of all of the matrix formats and representations available in the Matrix Market is available from the Matrix Market WWW web site. Rutherford-Boeing Sparse Matrix Collection files appear in only three of the options available: the coordinate format and the elemental format for sparse matrices and the array format for dense matrices. (The array format is only used for supplementary data, not for matrix files.)

4.1 The MM Formats for Sparse Matrices

Matrix files from the Rutherford-Boeing Sparse Matrix Collection in MM format consist of three sections: a standard one line header, a set of comments containing descriptive data relevant to the Rutherford-Boeing Sparse Matrix Collection, and a set of lines representing the numerical data. The first two sections are common to both the coordinate and elemental formats. The data sections are described separately in §4.1.1 and §4.1.2.

The header line identifies the file as a Matrix Market matrix file. It has the form

`%%MatrixMarket matrix` *[format] [field] [symmetry]*

The first keyword, `matrix`, distinguishes matrix files from other Matrix Market files. The second keyword, *[format]*, specifies how the matrix is represented. Only two of the possible Matrix Market formats apply to matrices in the Rutherford-Boeing Sparse Matrix Collection — matrices are either in `coordinate` or `elemental` format. The other two keywords state the numerical field from which the entries are drawn and the symmetry properties used in the matrix representation. The possible values for *[field]* are described

in Table 1.1. Table 1.2 summarizes the recognized symmetry properties and the allowed values for *[symmetry]*. Note that for matrices in elemental form, each element matrix satisfies the global symmetry property.

Table 1.1: **Recognized Values for** *[Field]*

Meaning	**Interpretation**
Real	Matrix entries are represented by a single floating-point number.
Complex	Matrix entries are represented by two floating-point numbers, the first giving the real part and the second the imaginary part.
Integer	Matrix entries are represented by a single integer.
Pattern	Only the matrix nonzero pattern is provided.

Table 1.2: **Recognized Values for** *[Symmetry]*

Meaning	**Interpretation**
General	The matrix has no symmetry properties, or symmetry is not used to reduce the number of matrix entries. (All non-square matrices are General.)
Symmetric	Square matrix with $A(i,j) = A(j,i)$. Only entries in the lower triangle are stored in the file.
Hermitian	Square complex matrix with $A(i,j) = \overline{A(j,i)}$. Only entries in the lower triangle are stored in the file.
Skew-symmetric	Square matrix with $A(i,j) = -A(j,i)$. Only entries below the main diagonal are stored in the file. (Note that $A(i,i) = 0$.)
Structurally-symmetric	Square matrix with $A(i,j) \neq 0 \iff A(j,i) \neq 0$. Symmetry of pattern is used to reduce indexing information for matrices in elemental form.

The meaningful combinations of keywords for MM matrices are:

%%MatrixMarket matrix coordinate	[real, complex, integer, pattern]	[general, symmetric, skew-symmetric]
%%MatrixMarket matrix coordinate	[complex]	[Hermitian]
%%MatrixMarket matrix RB-elemental	[real, complex, integer, pattern]	[general, symmetric, structurally-symmetric, skew-symmetric]
%%MatrixMarket matrix RB-elemental	[complex]	[Hermitian]

The MM formats are intended to be read by free-format or list-directed input routines. The character strings in the header line can be read in Fortran 90 as

```
read ( input, * ) MMHeaderString, MMtype, &
                  MMFormat, MMMxtype, MMSymmetry
```

It can be read in C as

```
fscanf ( input, "%s %s %s %s %s", MMHeaderString, MMtype,
                MMFormat, MMMxtype, MMSymmetry )
```

High quality routines are available in both Fortran and C from the Matrix Market.

The header line is followed by a set of lines interpreted in Matrix Market format as comments. These lines are not needed to read the matrix *per se*, but replicate data available in the Rutherford-Boeing format and serve as self-documentation for the file. They provide us with the ability to convert files in MM format back to Rutherford-Boeing form. Each of these lines begins with the text %%RB. The comment section always begins with three lines of the form

```
%%RBCode matrix
%%RBMatrixID  matrixid
%%RBTitle  title
```

These three lines may be followed by other comments commencing with %.

Character data in files in Matrix Market format may appear in either upper or lower case. That is, the data is case insensitive and programs to read these files must recognize either case. By convention within the Rutherford-Boeing Sparse Matrix Collection, all matrix identifiers of matrices in the Collection appear only in upper case and normal English capitalization is used in the descriptive title field.

Numerical entries of matrices from the Collection in MM format are subject to the same conditions used generally in the Matrix Market formats and in RB format, that is, explicit decimal points in numerical fields, and an exponent field, if present, of the form "E$\pm xy$" or "E$\pm xyz$", where x, y and z must each be a digit, not a blank.

4.1.1 The Matrix Market Coordinate Format

The coordinate format uses $< i, j, a_{ij} >$ triples as the basis for the representation. Files from the Rutherford-Boeing Sparse Matrix Collection in MM format include mandatory comments and meet restrictions on the order in which entries are listed, constraints which go beyond the basic specification of the Matrix Market formats.

The following template illustrates the numerical data section of MM files for matrices in coordinate format:

M	N	Nnonzeros
i_1	j_1	a(i_1,j_1)
i_2	j_2	a(i_2,j_2)
$\vdots$	$\vdots$	$\vdots$
i_k	j_k	a(i_k,j_k)
$\vdots$	$\vdots$	$\vdots$

The data section begins with three integers on a single line, separated by blank spaces, specifying the number of rows (M), columns (N), and entries (Nnonzeros) in the matrix. This is followed by Nnonzeros triples, listed each on its own line, with the values separated by blank spaces. "Triples" is used somewhat loosely here. In the case of matrix patterns, each data line contains only the two indices i and j since the numerical values are not present. In the case of complex matrices, each line contains the two indices i and j, followed by two floating-point values, representing the real part of a_{ij} and then the imaginary part, all separated by blank spaces.

Although the representation by triples does not require any particular ordering of the triples, the use of this Collection as a standard testbed causes us to impose a standard ordering. Matrices from the Rutherford-Boeing Sparse Matrix Collection in MM coordinate format appear with the triples in ascending reverse lexicographic ordering of the indices i and j. That is, $< i, j >$ appears before $< k, l >$ if $j < l$ or if both $j = l$ and $i < k$. In other words, the triples from the first column of the matrix are given first, in increasing order of row index, then those from the second column of the matrix, again with row indices strictly ascending, and so forth for all N columns. Coordinate form matrices from the Collection will not have any duplicate pairs of indices.

The data section is preceded by a header line and a comment section. A complete representation in MM format of the matrix

$$A = \begin{pmatrix} 1. & -4. & & -8. & \\ & & -6. & & 10. \\ 2. & & & & \\ & 5. & & -9. & \\ 3. & & -7. & & 11. \end{pmatrix}.$$

used as Example 1 would be:

```
%%MatrixMarket matrix coordinate real general
%%RBCode matrix
%%RBMatrixID EXAMPLE1
%%RBTitle Small general matrix used as Example 1
5   5   11
1 1  1.0
3 1  2.0
5 1  3.0
1 2 -4.0
4 2  5.0
2 3 -6.0
5 3 -7.0
1 4 -8.0
4 4 -9.0
2 5  1.0e+01
5 5  1.1e+01
```

4.1.2 The Matrix Market RB-Elemental Format

Matrices in elemental form are represented by a list of element matrices, "annotated" small dense matrices. The data section for the elemental representation is a concatenation of element matrices, where each element matrix is essentially an annotated Matrix Market array. The complete data section for a matrix in elemental form has the simple form:

$$\begin{array}{l} \texttt{M N Nelements} \\ \mathcal{E}_1 \\ \mathcal{E}_2 \\ \vdots \\ \mathcal{E}_k \\ \vdots \\ \mathcal{E}_{\texttt{Nelements}} \end{array}$$

Each $\mathcal{E}_i$ represents an element in a form we now describe.

The most general form represents an $r \times s$ rectangular element

$$\begin{array}{c} \\ i_1 \\ i_2 \\ \vdots \\ i_r \end{array} \begin{array}{c} \begin{array}{cccc} j_1 & j_2 & \dots & j_s \end{array} \\ \begin{pmatrix} c_{i_1,j_1} & c_{i_1,j_2} & \dots & c_{i_1,j_s} \\ c_{i_2,j_1} & c_{i_2,j_2} & \dots & c_{i_2,j_s} \\ \vdots & \vdots & \dots & \vdots \\ c_{i_r,j_1} & c_{i_r,j_2} & \dots & c_{i_r,j_s} \end{pmatrix} \end{array}$$

by the pattern

r s
i_1
i_2
$\vdots$
i_r
j_1
j_2
$\vdots$
j_s
c_{i_1,j_1}
c_{i_2,j_1}
$\vdots$
c_{i_r,j_1}
c_{i_1,j_2}
$\vdots$
c_{i_r,j_s}

In this pattern the numerical values appear in column major order, as they do in the Rutherford-Boeing formats and in the Matrix Market formats for dense matrices. The difference between the Matrix Market representation of an element matrix and a general dense matrix is the presence of the row and column indices between the dimensions and the numerical values. The index information is itself in the form of two Matrix Market integer vectors.

The Matrix Market representation for matrices in elemental form supports a special representation for square matrices that are generated from elements with symmetric index patterns, that is, where $r = s$ and $i_k = j_k$ for $k = 1, \ldots, r$. These are the most common cases of matrices from finite-element applications. Examples 2 and 3 are of this form. The second index set $(j_1, j_2, \ldots, j_r)$ is redundant, so it is omitted.

When all of the element matrices are symmetric (as in Example 2), so is the resulting elemental matrix. From symmetry it suffices to include only the entries in the lower triangle of each element matrix in the list. In contrast, if some or all element matrices are unsymmetric, we know only that the matrix assembled from these element matrices is structurally symmetric. That is, if entry (i, j) is an entry, (j, i) is also. However, the numerical values are not symmetric; that is, entry a_{ij} need not equal entry a_{ji}. In this case, each element matrix includes a full set of numerical entries, but only one set of indices is needed.

In the general case, the element matrices are rectangular. Both row and column variable indices must be held, and the resulting assembled matrix is unsymmetric or even rectangular.

We summarize the three major cases for element matrix patterns in the following table. (Skew-symmetric elements would be held like symmetric elements, excluding main diagonal entries.) Here, we indicate the order in which entries appear in a compressed form; the matrix files store each entry on a separate line.

Type of Matrix	Ordering of Data
general	r, s, $i_1, \ldots, i_r$, $j_1, \ldots, j_s$, $c_{i_1,j_1}, c_{i_2,j_1}, c_{i_3,j_1}, \ldots, c_{i_r,j_1}$, $c_{i_1,j_2}, c_{i_2,j_2}, c_{i_3,j_2}, \ldots, c_{i_r,j_2}$, $\vdots$, $c_{i_1,j_s}, c_{i_2,j_s}, c_{i_3,j_s}, \ldots, c_{i_r,j_s}$
structurally-symmetric	r, r, $i_1, \ldots, i_r$, $c_{i_1,i_1}, c_{i_2,i_1}, c_{i_3,i_1}, \ldots, c_{i_r,i_1}$, $c_{i_1,i_2}, c_{i_2,i_2}, c_{i_3,i_2}, \ldots, c_{i_r,i_2}$, $\vdots$, $c_{i_1,i_r}, c_{i_2,i_r}, c_{i_3,i_r}, \ldots, c_{i_r,i_r}$
symmetric	r, r, $i_1, \ldots, i_r$, $c_{i_1,i_1}, c_{i_2,i_1}, c_{i_3,i_1}, \ldots, c_{i_r,i_1}$, $c_{i_2,i_2}, c_{i_3,i_2}, \ldots, c_{i_r,i_2}$, $\vdots$, c_{i_r,i_r}

The matrix used as Example 3 consisted of the four element matrices:

$$\begin{array}{c} \\ 1 \\ 4 \end{array}\!\!\begin{array}{c} \begin{array}{cc} 1 & 4 \end{array} \\ \begin{pmatrix} 2. & 3. \\ 1. & 7. \end{pmatrix} \end{array} \qquad \begin{array}{c} \\ 1 \\ 5 \end{array}\!\!\begin{array}{c} \begin{array}{cc} 1 & 5 \end{array} \\ \begin{pmatrix} 3. & 4. \\ 2. & 8. \end{pmatrix} \end{array} \qquad \begin{array}{c} \\ 2 \\ 3 \\ 5 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} 2 & 3 & 5 \end{array} \\ \begin{pmatrix} 4. & 4. & 1. \\ 3. & 5. & 2. \\ 6. & 1. & 2. \end{pmatrix} \end{array} \qquad \begin{array}{c} \\ 3 \\ 4 \end{array}\!\!\begin{array}{c} \begin{array}{cc} 3 & 4 \end{array} \\ \begin{pmatrix} 2. & 6. \\ 8. & 2. \end{pmatrix} \end{array}.$$

This elemental matrix could be represented in MM format by:

```
%%MatrixMarket matrix RB-elemental real structurally-symmetric
%%RBCode matrix
%%RBMatrixID EXAMPLE3
%%RBTitle Matrix in elemental form used as Example 3
5 5 4
2 2
1
4
2.0
1.0
3.0
7.0
2 2
1
5
3.0
2.0
4.0
8.0
3 3
2
3
5
4.0
3.0
6.0
4.0
5.0
1.0
1.0
2.0
2.0
2 2
3
4
2.0
8.0
6.0
2.0
```

As in the case of coordinate form matrices, the ordering of indices is not intrinsic to the definition of a matrix, but is part of the Rutherford-Boeing Sparse Matrix Collection specification. Elemental form matrices from the Collection will satisfy the additional condition that indices appear in ascending order within elements.

The order in which the originating application supplied the element matrices may represent an important characteristic of the problem. We preserve this order in the Rutherford-Boeing Sparse Matrix Collection.

For each matrix held in elemental form, we also hold the assembled matrix in which the row and column variable indices in the elemental form are mapped onto the sets

$\{1, 2, 3, \ldots, u\}$ and $\{1, 2, 3, \ldots, v\}$, where u and v are the number of distinct row and column variable indices, respectively, that actually occur in the element matrices. Thus u and v are the row and column dimensions of the assembled matrix. In each case the index mapping preserves the order of indices, while eliminating any gaps in the sequence.

4.2 The MM Formats for Supplementary Data

Usually the sparse matrix represents only part of the problem. When available, we include additional data that more completely describes the linear algebra problem. Supplementary data for which we have defined specific representations are given in Table 1, replicated from §1.3. Each of these representations is a rectangular matrix, stored in a separate file. We have intentionally kept the supplementary file structures simple, similar to matrix files. Complicated problem descriptions are realized by multiple supplementary files.

Table 1: Supplementary Files

keyword	*extension*		
	type	*position*	*organization*
orderings	ord	[l|r|s]	
right-hand-sides	rhs	[l|r]	[e|d|s]
solutions	sln	[l|r|s]	
estimates	est	[l|r|s]	
eigenvalues	evl		
singular-values	svl		
eigenvectors	evc	[l|r|s]	
singular-vectors	svc	[l|r|s]	
Schur-basis-vectors	sbv		
Schur-basis-matrix	sbm		
Schur-basis-parameters	sbp		
partition	ipt	[l|r|s]	
covering	icv	[l|r|s]	
Laplacian-values	lvl		
Laplacian-vectors	lvc		
geometry	geo		
auxiliary-values	avl		

Most supplementary cases are represented by dense data, for which we use the Matrix Market `array` format for dense matrices. We do not require or use symmetry in supplementary files, so we use only the general version of the `array` format. Example 9 below gives a simple example of a MM array formatted file. The key MM characteristics are the initial header line, which has the form

`%%MatrixMarket matrix array` *[field]* `general`

and then, after optional comment lines, a line specifying the row and column dimensions of the array. The numerical entries of the array follow, one per line, in column major order. That is, when D is a $q \times r$ array, its entries are held in the order

$$D_{11}, D_{21}, D_{31}, \ldots, D_{q1}, D_{12}, D_{22}, D_{32}, \ldots, D_{q2}, D_{13}, D_{23}, \ldots, D_{qr}.$$

Sparse supplementary data is represented in the MM format for sparse matrices. Thus, Rutherford-Boeing Sparse Matrix Collection supplementary files in MM format are standard Matrix Market Exchange Format matrix files in either `matrix array` *[field]* `general` or `matrix coordinate` *[field]* `general` format. These files can be read directly with the standard Matrix Market routines for these two fundamental formats.

Each Rutherford-Boeing supplementary data file contains a series of comments to make the files self-documenting. In particular the second line of the file is a structured comment that describes how to interpret the role of the supplementary data contained in the file. Other comments describe the relationship between this file and a matrix or matrices in the Collection. These comments are not generally intended to be parsed by the user's program, although they are structured to make that possible.

Ultimately a user of the supplementary files will use these data files together with matrix files. The relationship between the data files will be determined by the user's program and probably file naming and file assignment procedures in the user's computer system. The comments permit the user to develop some mechanism for self-checking the file assignments. (We have found that simply copying the header comments into the output stream is a useful tool.)

Example 9 *A sample file, representing two right-hand sides,* $[0,0,3,4,0]^T$ *and* $[0,0,0,0,1]^T$*, for a* 5×5 *matrix, is given by:*

```
%%MatrixMarket matrix array real general
%%RBCode right-hand-sides right dense
%%RBMatrixID EXAMPLE3
%%RBCaseID FULL_R2
%%RBTitle set of right-hand sides for Example 3
%
%    This file illustrates the Rutherford-Boeing supplementary file
%       extensions to the Matrix Market format.
%    The first line identifies this file as a Matrix Market matrix
%       in array format.  Lines beginning with ''%'' are comments.
%    The four lines beginning with ''%%RB'' are mandatory
%       Rutherford-Boeing extensions to the Matrix Market standard.
%    The first numerical line gives the row and column dimensions
%       of the matrix.  Succeeding lines list the numerical entries
%       in column major order
%
5     2
0.000
0.000
3.000
4.000
0.000
0.000
0.000
0.000
0.000
1.000
```

The right-hand sides are represented by a Matrix Market array with five rows and two columns. The type code, `%%RBCode` given on the second line, specifies the data to be right-hand side data (`right-hand-sides`) for the system $Ax = b$ (`right`), stored as n vectors (`dense`).

The second line of the file, containing the type code, must have the form:

`%%RBCode` *datatype [position] [organization]*

The other three lines beginning with `%%RB` must appear in each supplementary file. Like other Matrix Market files, character data in supplementary files is case insensitive. Within the Collection we consistently use only uppercase letters within matrix identifiers and use normal English capitalization in descriptive fields.

The position and organization keywords appear only when the datatype permits several options. The position keyword can take the following values:

keyword	meaning
`left`	data appears on left side of matrix or matrices (e.g., y in $y^T A = y^T \lambda$)
`right`	data appears on right side of matrix or matrices (e.g., x in $Ax = b$)
`symmetric`	data is a single argument to be used symmetrically on both sides of a matrix or matrices (e.g., P in PAP^T)

The specific meanings of the position codes for a particular datatype are given in the separate discussions of the various datatypes. However the use is consistent across the types of data.

The organization keyword, which currently appears only to describe right-hand side data, must be one of:

keyword	meaning
`dense`	the data is in the form of dense vectors of order one of the dimensions of the associated matrix
`sparse`	the supplementary data is in the form of sparse vectors of order one of the dimensions of the associated matrix; indexing information is part of the data
`elemental`	the data is stored as dense vectors corresponding to the elemental indexing of the associated matrix in elemental form

We name supplementary data files in our archive using the convention described in §1.4, namely

matrixid.ext.case.ff

The *ext* field is built from a three letter acronym for the datatype, followed by the initial letter of the position keyword if present and the initial letter of the organization keyword if present. The acronyms appear in Table 1, which also indicates for which datatypes the position or organization keywords must appear.

4.2.1 Orderings
(`%%RBCode orderings` *position*)

These files provide permutations that represent interesting or unusual reorderings of sparse matrices. The permutations are given by integer mappings which map the original (old) labeling of the matrix as stored in the Collection to its new labeling. Thus, in the symmetric permutation case, the reordered matrix $\widehat{A}$ is represented mathematically by $\widehat{A} = PAP^T$. If the permutation vector is denoted by p, the entry-wise relationship is $\widehat{A}_{p(i),p(j)} = A_{ij}$. The general case may require both a left and a right permutation P and Q, with the reordered matrix given by $\widehat{A} = PAQ$. If the stored permutation vectors are denoted by p and q, the entry-wise relationship is $\widehat{A}_{p(i),q(j)} = A_{ij}$. Such general orderings are represented in the Collection by a pair of files, one giving P and the other Q.[1]

The position keyword is interpreted as

`symmetric`	if the single permutation P should be applied symmetrically
`left`	if the file represents the left permutation P
`right`	if the file represents the right permutation Q

Note that unsymmetric matrices can be permuted symmetrically and unsymmetric permutations of symmetric matrices are also valid. In the case of permutations for rectangular matrices, the left and right permutation vectors are of different lengths.

[1] The two permutations are not encoded in a single file because the interpretation of the data as a vector or a matrix is awkward when the matrix is rectangular.

Example 10 *This file contains two symmetric permutations for Example 1.*

```
%%MatrixMarket matrix array integer general
%%RBCode orderings symmetric
%%RBMatrixID EXAMPLE1
%%RBCaseID TWOPERMS
%%RBTitle two symmetric orderings for Example 1
%
% this file illustrates a possible ordering file for Example 1
%
5 2
5
3
4
2
1
4
3
5
1
2
```

This example presents two separate symmetric permutations of the 5×5 *example matrix. Each permutation is given as a column of a* 5×2 *integer matrix. The first permutation is the vector* $p^{(1)} = [5,3,4,2,1]^T$ *and the second permutation is* $p^{(2)} = [4,3,5,1,2]^T$. *The qualifier* `symmetric` *indicates that each permutation is to be used symmetrically.*

What permutations are available for a given matrix is specified in the documentation file (*matrixid*.doc) for the given matrix *matrixid*. All files for a given matrix can be accessed together automatically by the Matrix Market when the standard WWW interface is used to retrieve files. In the case of permutations for rectangular matrices, the left and right permutation vectors are of different lengths. When the Collection holds both a left and a right permutation, these are represented in Matrix Market format by two separate files. In such cases each of the permutation files includes a text description of the link to the other permutation in its header comments.

Orderings for a matrix in elemental form are given as permutations of the elements, not of the variables. Thus the row dimension of the permutation(s) is the number of elements.

4.2.2 Right-Hand Side Vectors (`%%RBCode right-hand-sides` *position organization*)

Example 9 introduced the most simple and most common format for right-hand sides. The type code includes two qualifier keywords. The position keyword is interpreted as:

`right`	right-hand side b for $Ax = b$
`left`	right-hand side b for $A^T x = b$.

Corresponding interpretations are used for right-hand sides for least-squares problems with rectangular matrices.

Usually right-hand sides are not sparse. We store a typical right-hand side as a dense vector whose order matches the appropriate order of the associated matrix. We permit two other representations for right-hand sides: an elemental representation of the right-hand sides for matrices in elemental form and a sparse representation for the special case in which the right-hand sides are sparse. (At present, only the representation of right-hand sides use multiple representations. This section is therefore the most complicated discussion of supplementary data.)

The three cases are distinguished by the organization keyword, , which must take one of the following options.

`dense`	if the right-hand sides are ordinary dense vectors of order M or N as the position keyword is `right` or `left` respectively
`sparse`	if the right-hand sides are stored as sparse vectors
`elemental`	if the right-hand sides are dense vectors corresponding to the elemental structure of the associated matrix.

Dense Right-Hand Sides

This first case is the most common. The right-hand sides are represented by an $N \times K$ or an $M \times K$ `array`, where M and N are the row and column dimensions of the associated matrix A, and K is the number of right-hand sides. Thus a file that begins

```
%%MatrixMarket matrix array real general
%%RBCode right-hand-sides right dense
```

will have a numerical data section of the form

$$
\begin{array}{ll}
m & k \\
b_{11} & \\
b_{21} & \\
\vdots & \\
b_{m1} & \\
b_{12} & \\
b_{22} & \\
\vdots & \\
b_{m2} & \\
\vdots & \\
b_{mk} &
\end{array}
$$

Example 9 is a concrete illustration of the file format.

Sparse Right-Hand Sides

Applications with sparse right-hand sides are less common, but the sparsity can be exploited by direct solution techniques. We store each sparse right-hand side in the same format as a column of a general sparse matrix. Thus the representation of multiple right-hand sides is stored as a standard sparse matrix, with as many rows as the coefficient matrix and as many columns as there are right-hand sides. That is, all entries are given explicitly by triples of the form $< i, k, b_{ik} >$, where b_{ik} is the value found in row i of the

kth right-hand side. Thus, a set of K sparse right-hand sides for an $M \times N$ matrix A will be a Matrix Market file beginning

```
%%MatrixMarket matrix coordinate real general
%%RBCode right-hand-sides left sparse
```

and having a numerical data section of the form

$$
\begin{array}{lll}
n & k & l \\
i_1 & 1 & b_{i_1 1} \\
i_2 & 1 & b_{i_2 1} \\
\vdots & & \\
i_p & 1 & b_{i_p 1} \\
i_{p+1} & 2 & b_{i_{p+1} 2} \\
i_{p+2} & 2 & b_{i_{p+2} 2} \\
\vdots & & \\
i_l & k & b_{i_l k}
\end{array}
$$

Note that our convention for ordering the indices in sparse matrices applies throughout the collection. We list all entries for the first right-hand side before any entries for the second right-hand side, and so forth. Within each column (right-hand side) the row indices appear in ascending order.

Elemental Right-Hand Sides

Normally applications in which the sparse matrix is in elemental form use the same elemental structure to represent the right-hand sides. Elemental right-hand sides are only supplied when the associated matrix is in `elemental` form. The elemental structure of the right-hand sides must be interpreted through the elemental structure that describes the matrix.

A single elemental right-hand side is given by a list of q values $c_k, k = 1, q$, where q is the sum of the number of variables in individual elements. Each entry c_k is a contribution to some entry in the right-hand side b of the assembled system $Ax = b$. Specifically, we list the entries of c in order corresponding to the order in which the (row) indices appear in the elemental representation of the matrix. Thus the list is a concatenation of contributions from individual elements with those from element i immediately preceding those from element $i + 1$. If the elemental indices in this order are $i_k, k = 1, q$, then the right-hand side b is defined by $b_j = \sum \{c_k | i_k = j\}$.

A set of K elemental right-hand sides will be represented by a $q \times K$ dense array, in standard Matrix Market array format. Thus, the first q entries represent the first right-hand side, the next q entries give the second right-hand side, and so forth.

Example 11 *The following file gives two right-hand sides for the matrix in Example 3. Confirming that the right-hand sides are the same as in Example 9 will test your understanding of the representation:*

```
%%MatrixMarket matrix array real general
%%RBCode right-hand-sides right elemental
%%RBMatrixID EXAMPLE3
%%RBCaseID ELMNT_R2
%%RBTitle elemental right-hand sides for Example 3
5     2
-1.0
 2.0
 1.0
 5.0
 0.0
 0.0
-5.0
 3.0
 2.0
 2.0
 1.0
-2.0
 0.0
 0.0
-3.0
 1.0
 3.0
-1.0
```

4.2.3 Solution Vectors and Estimates of Solution (%%RBCode solutions *position*) (%%RBCode estimates *position*)

We always hold solution vectors and estimates of the solution in dense format. An estimate of the solution may be available in the originating application and may be used, for example, as a starting guess for an iterative solution scheme. Solutions and estimates of solutions to sets of K right-hand sides are represented by arrays of dimensions N or M by K.

The position qualifier has the same meaning as with right-hand sides. In particular the position qualifier should match the position qualifier in the file of right-hand sides for which this data claims to be a solution or an estimate of the solution.

4.2.4 Eigenvalues, Singular Values, and Eigenvalues of Laplacian (%%RBCode singular-values) (%%RBCode eigenvalues) (%%RBCode Laplacian-values)

These files provide a list of some or all of the eigenvalues or singular values for a particular eigenproblem, or the eigenvalues of the Laplacian matrix[2] associated with a matrix in the

[2]The Laplacian of a (structurally) symmetric sparse matrix A is another sparse matrix, L_A with the same sparsity pattern as A. It has off-diagonal entries with the value -1 in exactly the same positions as the off-diagonal entries of A. Each diagonal of the Laplacian is set to the number of off-diagonal entries in the corresponding row of A; thus each diagonal entry of L_A is the modulus of the sum of the off-diagonals

Collection. In each case only a single vector is needed, which is stored as an Matrix Market array with a single column.

Example 12 *A file containing the lowest thirty eigenvalues of a symmetric matrix* `testmtx` *would begin with:*

```
%%MatrixMarket matrix array real general
%%RBCode eigenvalues
%%RBMatrixID TESTMTX
%%RBCaseID LOWEST30
%%RBTitle eigenvalues for ....
30 1
```

This would be followed by the 30 eigenvalues, listed one per line.

A given eigenproblem may be accompanied by several eigenvalue supplementary files. These may represent variations in accuracy, in choice of subset of eigenvalues or to match different subsets of eigenvectors in linked supplementary files. Different approximations to the same eigenpair may appear in more than one set of these files. An eigenvalue file may be associated with an eigenvector file. If this is the case, then the number of eigenvalues and the number of eigenvectors are equal.

4.2.5 Eigenvectors, Singular Vectors and Laplacian Vectors (%%RBCode `eigenvectors` *position*) (%%RBCode `singular-vectors` *position*) (%%RBCode `Laplacian-vectors` *position*)

These files hold some of the eigenvectors or singular vectors for a particular eigenproblem. In the Rutherford-Boeing Sparse Matrix Collection each eigenvector or singular vector file is always associated with a file containing the corresponding eigenvalues or singular values; values associated with Laplacian vectors are not always held. Only in trivially small cases will all the eigen- or singular vectors be available. Generally only subsets of the eigenvectors will be held. Since the eigenvectors of sparse matrices are usually dense, the vectors are stored as Matrix Market arrays. The eigenvectors of general real matrices are stored as complex vectors.

The position keyword has the meanings:

`symmetric`	if symmetry allows the vectors to represent both left and right eigensolutions
`left`	if the file contains the left vectors of a general eigenproblem or the left singular vectors
`right`	if the fi2le contains the right vectors of a general eigenproblem or the right singular vectors

The file contains an $M \times K$ array ($N \times K$ in the case of the left singular vectors). The comments always provide text descriptions of related files.

in its row. The eigenvalues and eigenvectors of L_A provide useful graph characterizations of the sparsity pattern of A.

4.2.6 Schur Bases

(%%RBCode `Schur-basis-vectors` *position*)
(%%RBCode `Schur-basis-matrix` *position*)
(%%RBCode `Schur-basis-parameters` *position*)

The matrix of eigenvectors of an unsymmetric matrix (or the solutions of other general eigenproblem) can be arbitrarily badly conditioned. An alternative representation with better numerical properties is the Schur decomposition that, for an ordinary eigenproblem, is given by $AU = UT$, where U is unitary and T is block upper triangular, with blocks of order one and two. The 1×1 and 2×2 blocks on the diagonal of T give the real eigenvalues and the complex eigenvalues (as a complex conjugate pair) respectively. A partial Schur decomposition is simply the leading columns of a (complete) Schur decomposition. Schur bases files hold partial Schur decompositions for ordinary eigenproblems and for some generalized problems. In the Rutherford-Boeing Sparse Matrix Collection each Schur basis is represented by two or three files. Two are always required: one holds the vectors U as a dense matrix and the second holds the block triangular matrix. The first is the "Schur-basis-vectors" file; the second is the "Schur-basis-matrix" file. The latter is stored as a square dense matrix which includes the zero entries in the lower triangular part.

The third file is used to help accurately represent Schur bases that were obtained using a transformation of the spectrum of the eigenproblem. Certain transformations are in common use as very powerful accelerators for sparse eigensolution algorithms, especially for generalized eigenproblems. We allow representations for cases where the transformation is a rational function of the matrix arguments of the problem, T and I. In these cases we hold the Schur basis matrix for the transformed problem because the transformation back to the original problem may be badly conditioned. In addition we hold the coefficients of the rational function. We do not attempt a general representation here; the third file, the "Schur-basis-parameters" file is simply a vector of coefficients. The form of the rational function is given in the documentation of the problem and so the coefficients can only be interpreted with the documentation.

The position keyword has the meanings:

`left`	if the files contain the left Schur basis
`right`	if the files contain the right Schur basis

The Schur-basis-vector file contains an $M \times K$ array ($N \times K$ in the case of the left singular vectors). The Schur-basis-matrix file contains a dense $K \times K$ matrix. The comments always provide text descriptions of related files.

4.2.7 Index Partitions and Coverings

(%%RBCode `partition` *position*)
(%%RBCode `covering` *position*)

These files represent partitions or coverings of the row or column indices of sparse matrices. A partition is a set of S subsets $\{I_k, k = 1, 2, \ldots, S\}$ of the M row indices or the N column indices, such that each index is assigned to one particular subset. Thus a partition can be represented as a map from the index number i to the index k of the single subset I_k that contains it. A covering of the row (or column) indices is also a set of subsets of the

indices, $\{I_k, k = 1, \ldots, S\}$, such that every index appears in *at least one* subset of indices I_k. Coverings can be used to describe an underlying elemental structure for a matrix or to describe hierarchical or overlapping node partitionings as in domain decompositions. Clearly the difference between a partition and a covering is that an index in a partition is assigned to a unique subset, whereas it can be a member of several subsets in a covering. We use the same file format to represent both.

The index sets I_k are represented as a sparse matrix pattern with M or N rows and S columns. The occurrence of the pair $< i, k >$ specifies that row index i occurs in subset k. By our usual convention, we list the pairs in lexicographic order. That is, the indices for the first subset are given first, with the row indices in order, then the indices for the second subset, and so on. A file of this type holds only a single index partition or covering.

The position is defined exactly as

`symmetric`	if the single partition or covering should be applied symmetrically
`left`	if the file represents a partition or covering of the row indices
`right`	if the file represents a partition or covering of the column indices

Example 13 *Thus, a row index covering file for a sparse matrix will begin*

```
%%MatrixMarket matrix coordinate pattern general
%%RBCode icv 1
%%RBMatrixID ....
%%RBCaseID ....
%%RBTitle row index covering for ....
M S L
```

which would be followed by the list of the L index pairs, one per line.

A file of this type holds only a single index partition or covering. General or unsymmetric partitionings or coverings will be represented by a pair of files. The representation described above does not prohibit empty subsets in a partition, which would be given by a pattern with an empty column. However, the convention used in the Collection is that partitions will not have empty subsets.

4.2.8 Geometric Coordinate Data (%%RBCode geometry *position*)

Often problems, and matrices, arise in a physical or geometric setting. In such cases there may be natural mappings between algebraic variables in the model and coordinates in a d-dimensional physical or Euclidean space. For example, the ith row of a matrix may describe the state equation that holds at a particular point $< x_i, y_i, z_i >$ in 3-space. The geometric coordinate data files capture this relationship.

Each geometry file represents a mapping between k points in a d-dimensional physical space and the row and/or column indices of the matrix. The geometric coordinate data give the locations of the k points in the d-dimensional physical space as a dense $k \times d$ array, stored in Matrix Market `array` format. Whether the geometric data describes the row vertices of the matrix, the column vertices, or both, is specified by the value of the position keyword. The position is described in the usual way as:

`symmetric`	if the geometry data applies symmetrically to row and column vertices
`left`	if the geometry represents the left/row vertices
`right`	if the geometry represents the right/column vertices

Example 14 *This leads to the following simple structure for data from problems in three physical space dimensions.*

```
%%MatrixMarket matrix array real general
%%RBCode geometry right
```

$k \quad 3$
x_1
x_2
$\vdots$
x_k
y_1
y_2
$\vdots$
y_k
z_1
z_2
$\vdots$
z_k

In the simplest case there is a 1-to-1 mapping between geometric points and algebraic variables; the number k of physical points is the same as M or N, the row and/or column dimensions of the associated matrix.

There are, however, more complicated cases. Often a model uses several variables to represent different characteristics at each of a set of physical points. For example, in fluid flow, there may be variables corresponding to velocity in each of three directions, pressure and temperature at each point in 3-dimensional physical space. If these five variables are modelled at each point, the dimensions of the matrix are five times the number of physical points. Representing geometric data directly for each variable would result in redundant data. We permit a more concise representation, which is sufficiently general to allow models that do not have the same number of variables at each physical point. Further, the representation identifies the set of variables corresponding to each particular common location. In the Rutherford-Boeing Sparse Matrix Collection we represent such situations with two files. One is a `*.geo?.*` geometry file, containing the geometric locations as a $k \times d$ array, where k is the true number of physical points. The second file contains the nodal partition that labels all variables at a given physical point as a single subset; this will be an `*.ipt?.*` file, partitioning M or N indices. The number of partition subsets in the index partition file, S, must equal k. The mapping between the two files is such that the locations in the ith row of the array are the locations of all variables in subset i in the partition.

4.2.9 Auxiliary Values
(%%RBCode `auxiliary-values` *position*)

Other kinds of information may be associated with a sparse matrix. Sparse matrices are often associated with graphs; for example, each row or column can represent a vertex in the graph and each entry in the matrix can correspond to an edge in the graph. In some contexts, a graph may have several values attached to each vertex or to each edge. For example, network flow problems may have costs and capacities for each edge. In such cases, rather than representing the single problem by multiple matrices sharing a common sparsity pattern, we can use supplementary files associated with a single matrix.

Auxiliary values may also be associated with the vertices. A geometric description of a problem may, for instance, associate some sort of nodal type with each vertex. For example, a modeler may distinguish between nodes on the the boundary of the physical graph and nodes in the interior. Such labeling can easily be accomplished by an integer vector of length the number of rows and/or columns.

We expect auxiliary value files to contain quite specialized, problem-dependent, data. To be flexible we only require that they be vectors or arrays, with the dimensions and structure described in text in the comment section and in documentation for the associated matrix. The file structure is simply that of a Matrix Market array, beginning with the usual header line and the four required %%RB lines, with additional self-documenting comments. For example, a Schur basis parameters file, described in §4.2.6, is a particular, now named, example of a quite general use of an auxiliary value file.