**Technical Report**
RAL-TR-2006-014

# Sparse system solution and the HSL Library

**Iain S. Duff**

September 1, 2006

Council for the Central Laboratory of the Research Councils

Library and Information Services
CCLRC Rutherford Appleton Laboratory
Chilton Didcot
Oxfordshire OX11 0QX
UK
Tel: +44 (0)1235 445384
Fax: +44(0)1235 446403
Email: library@rl.ac.uk

CCLRC reports are available online at:
http://www.clrc.ac.uk/Activity/ACTIVITY=Publications;SECTION=225;

**ISSN 1358-6254**

**RAL-TR-2006-014**

# Sparse system solution and the HSL Library[1]

Iain S. Duff[2]

**ABSTRACT**

We consider the solution of large sparse systems, sketch their ubiquity, and briefly describe some of the algorithms used to solve these systems.

The HSL mathematical software library started life in 1963 as the Harwell Subroutine Library making it one of the oldest such libraries. The main strengths of the Library lie in packages for large scale system solution. It is particularly strong in direct methods for sparse matrices and optimization. The Library has been used worldwide by a wide range of industries.

We briefly discuss the history of the library and its organization and contents. We discuss the evolution of some of our current packages and the efforts to ensure reliability, robustness, and efficiency.

We describe in some detail the functionality of one of our most popular sparse direct codes.

**Keywords:** sparse solvers, sparse matrices, sparse linear equations, mathematical software libraries, Harwell Subroutine Library, HSL, MA57.

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

---

[1]This paper was presented as an invited talk at the Shanghai Forum on Industrial and Applied Mathematics, 25-26 May 2006. Current reports available by anonymous ftp to `ftp.numerical.rl.ac.uk` in directory pub/reports. This report is in files duffRAL2006014.pdf and duffRAL2006014.ps.gz. The report also available through the URL `www.numerical.rl.ac.uk/reports/reports.html`.

[2]i.s.duff@rl.ac.uk. This work was supported by the EPSRC Grant GR/S42170.

Computational Science and Engineering Department
Atlas Centre
Rutherford Appleton Laboratory
Oxon OX11 0QX

September 1, 2006

# Contents

# 1 Introduction

Sparse systems of linear equations

$$\mathbf{Ax} = \mathbf{b} \tag{1.1}$$

are simply sets of linear equations where the coefficient matrix $\mathbf{A}$ has sufficient zero entries to cause it to be beneficial to exploit this fact. Thus we are not here concerned with any particular structure or any particular application area. In fact, we emphasize in Section 2 the ubiquity of sparse systems and illustrate the diversity of their structure. In Section 3, we briefly introduce the main elements of the direct solution of sparse equations indicating their complexity on a range of structures. We then discuss the origins, structure, and development of HSL in Section 4 before a brief summary of the paper in Section 5.

# 2 Sparse matrices

Sparse matrices have arisen naturally in numerical applications since the mid 1960s. Some of the earliest applications involving the solution of sparse systems with a general structure were in the solution of ordinary differential equations using backward difference formulae and in linear programming. The former area was the main driving force in the development of sparse matrix methods at AERE Harwell in the 1970s. We list some of the major numerical application areas stimulating and benefiting from sparse matrix research in Table 2.1.

| |
|---|
| Stiff ODEs ... BDF ... Sparse Jacobian |
| Linear Programming |
| ..... simplex |
| ..... interior point |
| Optimization/Nonlinear Equations |
| Elliptic Partial Differential equations |
| Eigensystem Solution |
| Two Point Boundary Value Problems |
| Least Squares Calculations |

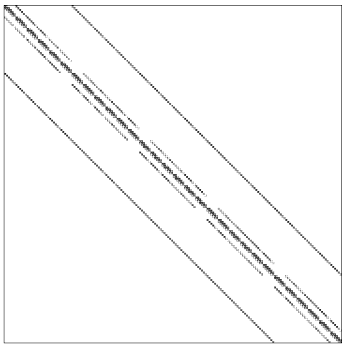
Table 2.1: Some numerical applications

In a more general context, we can look at application areas that often make extensive use of sparse matrices or sparse equation solvers. We show a range of these in Table 2.2. In this list, standard applications in the hard sciences are listed along with slightly more esoteric applications in the soft sciences.

We now show some pictures of sparse matrices from various applications in order to illustrate different structures for sparse matrices. The thermal simulation example exhibits a structure which is very structured and familiar to most of you. It is typical of a matrix arising in the finite-difference discretization of a three-dimensional elliptic PDE. In this case, the inclusion of thermal terms give this matrix, from oil reservoir modelling, interesting properties.

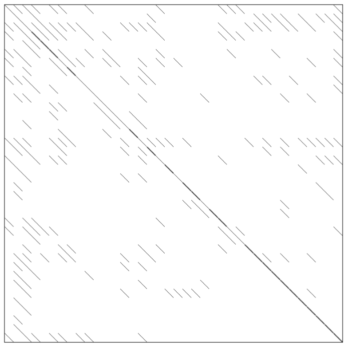
| | |
|---|---|
| Physics | CFD |
| | Lattice gauge |
| | Atomic spectra |
| Chemistry | Quantum chemistry |
| | Chemical engineering |
| Civil engineering | Structural analysis |
| Electrical engineering | Power systems |
| | Circuit simulation |
| | Device simulation |
| Geography | Geodesy |
| Demography | Migration |
| Economics | Economic modelling |
| Behavioural sciences | Industrial relations |
| Politics | Trading |
| Psychology | Social dominance |
| Business administration | Bureaucracy |
| Operations research | Linear Programming |

Table 2.2: Application areas



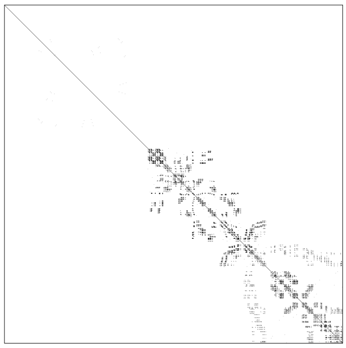
Thermal Simulation; SHERMAN2

Our so-called weather matrix is somewhat more interesting and models the combination of chemical kinetics and atmospheric transport. It is in fact a block matrix where each block is diagonal or tridiagonal and comes from studies in atmospheric pollution, a hot topic in environmental science.



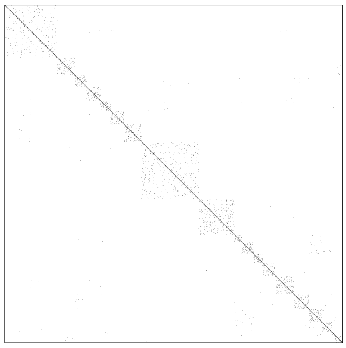
Weather Matrix; FS 760 3

The matrix from dynamic calculations is typical of a matrix arising from a finite-element discretization of a structures problem, in this case in a study of the effect of earthquake vibrations on a building in the western USA.



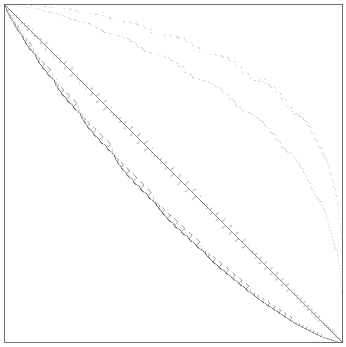
Dynamic Calculation in Structures; BCSSTM13

The power system matrix also comes from the western USA. Those with an eagle eye will see that the matrix is not quite block diagonal but there are only few entries outside the diagonal blocks that are themselves sparse. The blocks correspond to the power system network for a single utility and the off-diagonal entries to the much fewer links between the utilities, that often only carry loads when there is a problem in one utilities capacity.



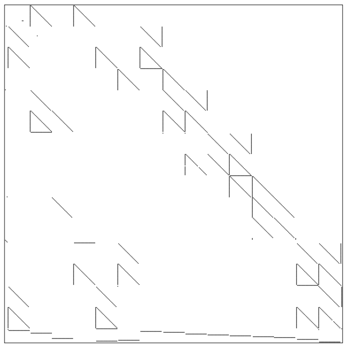
Power Systems; BCSPWR07

The matrix from the simulation of computing system comes from a Markov model of a computing system and has the remarkable anti-symmetric property that if there is an entry $a_{ij}$ then the entry $a_{ji}$ does not exist, for all $i \neq j$.



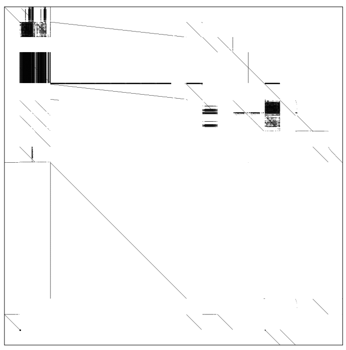
Simulation of Computing Systems; GRE 1107

The chemical engineering industry is a rich source of unsymmetric sparse matrices that are particularly challenging for solution by iterative methods. There is certainly a structure to the matrix but somewhat more irregular than the earlier examples. Notice that the diagonal is nearly all zero.



Chemical Engineering; WEST0381

The matrix from an econometric input/output model from South East Asia also has considerable structure but not one that can be exploited like that of the first matrix we displayed.



Economic Modelling; ORANI678

Duff and Reid distributed a set of sparse matrices from Harwell in the late 1970s but the main test collection for many years was the Harwell-Boeing Sparse Matrix Collection. This is available by anonymous ftp from `ftp.numerical.rl.ac.uk` in directory pub/harwell_boeing or from the Web page `http://www.cse.clrc.ac.uk/nag/hb/hb.shtml`

This set was later developed by Duff, Grimes, and Lewis (7 ) to include larger matrices in a wider range of application areas and to define more language-friendly formats and auxiliary files for other matrix properties (for example, eigenvalues) and associated information (for example, sparsity orderings). The Rutherford-Boeing Sparse Matrix Collection (8 ) will be supported by the GRID-TLSE Project `http://www.enseeiht.fr/lima/tlse` and is also available by anonymous FTP to `ftp.cerfacs.fr` in the directory pub/algo/matrices or `http://www.cerfacs.fr/algor/Softs/RB/index.html`

An extended set of test matrices available from Tim Davis at `http://www.cise.ufl.edu/research/sparse/matrices` and Matrix market at `http://math.nist.gov/MatrixMarket`.

# 3 Direct Methods

Although equation (1.1) nominally has the solution

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

it must be stressed that this should only be thought of notationally. It is really crucial that one must not use or even think of the inverse of $\mathbf{A}$ in this context.

For sparse $\mathbf{A}$, $\mathbf{A}^{-1}$ is usually dense. Indeed, if $\mathbf{A}$ is irreducible, one can prove (5 ) that $\mathbf{A}^{-1}$ will always be dense in a structural sense. That is, there exists a set of entries in the original sparsity pattern of $\mathbf{A}$ that make any position in the matrix $\mathbf{A}^{-1}$ nonzero.

Examples of sparse matrices that are very sparse but have dense inverses are tridiagonal and arrowhead matrices, where an arrowhead matrix has entries only in all positions on the diagonal and the last row and column. These examples are particularly interesting since, although their inverses are dense, linear systems involving these as coefficient matrices can be solved with no extra storage, as we shall shortly show.

If we thus dismiss the use of the inverse, we must propose other methods for solving the systems of the form (1.1). In some instances, iterative methods (11 , 12 ) can be used, often based on Krylov sequences, but these are not guaranteed to converge on general systems and usually require very sophisticated preconditioning so we do not consider them further here. Instead we look at direct methods for solution (6 ) that involve some matrix factorization representation of the inverse. The methods that we consider here are all based on Gaussian Elimination, that generates the factorization:

$$\mathbf{PAQ} \rightarrow \mathbf{LU} \tag{3.2}$$

where permutations $\mathbf{P}$ and $\mathbf{Q}$ are chosen to preserve sparsity and maintain stability, and $\mathbf{L}$ and $\mathbf{U}$ are lower and upper triangular matrices, respectively. When $\mathbf{A}$ is symmetric, the factorization is of the form

$$\mathbf{PAP^T} \rightarrow \mathbf{LDL^T}. \tag{3.3}$$

The solution to equation (1.1) is then easily obtained by solving the lower triangular system

$$\mathbf{Ly} = \mathbf{Pb}$$

followed by the upper triangular system

$$\mathbf{UQ^Tx} = \mathbf{y}.$$

Clearly, as in the case for dense systems, most of the work is usually in the factorization. The work in the forward and back substitution is proportional to the number of entries in the factors. This subdivision of work is reflected in software for sparse direct methods. Although the exact subdivision of tasks for sparse direct solution will depend on the algorithm and software being used, a common subdivision is given by:

**ANALYSE** An analysis phase where the matrix structure is analysed to produce a suitable ordering and data structures for efficient factorization.

**FACTORIZE** A factorization phase where the numerical factorization is performed.

**SOLVE** A solve phase where the factors are used to solve the system using forward and backward substitution.

We note the following:

- ANALYSE is sometimes preceded by a preordering phase to exploit structure.
- For general unsymmetric systems, the ANALYSE and FACTORIZE phases are sometimes combined to ensure the ordering does not compromise stability.
- The concept of separate ANALYSE and FACTORIZE phases is not present for dense systems.

It is crucially important to try to ensure sparsity in the factors **L** and **U**. This is done by choosing an ordering for the elimination. For example, if we pivot down the diagonal of the matrix in the left-hand side of Figure 3.1 then the resulting matrix of factors will be dense, as shown on the right-hand side of Figure 3.1.

$$\begin{array}{ccccc} \times & \times & \times & \times & \times \\ \times & \times & & & \\ \times & & \times & & \\ \times & & & \times & \\ \times & & & & \times \end{array} \quad \xrightarrow{L/U} \quad \begin{array}{ccccc} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{array}$$

Figure 3.1: Factorization of reverse arrowhead matrix

However, if we permute this matrix symmetrically to put the last row and column to the end, obtaining the arrowhead matrix shown on the left-hand side of Figure 3.2, then the factors require no more space than the original matrix as shown on the right-hand side of Figure 3.2.

The complexity of LU factorization on a dense matrix of order $n$ is:

$$\begin{array}{cl} \frac{2}{3}n^3 + \mathcal{O}\left(n^2\right) & \text{floating-point operations (flops)} \\ n^2 & \text{storage,} \end{array}$$

while, for a band matrix (order $n$, semi-bandwidth $k$), it is:

$$\begin{array}{ccccc} \times & & & & \times \\ & \times & & & \times \\ & & \times & & \times \\ & & & \times & \times \\ \times & \times & \times & \times & \times \end{array} \quad \xrightarrow{L/U} \quad \begin{array}{ccccc} \times & & & & \times \\ & \times & & & \times \\ & & \times & & \times \\ & & & \times & \times \\ \times & \times & \times & \times & \times \end{array}$$

Figure 3.2: Factorization of arrowhead matrix

$$2k^2n \text{ flops, } 2nk \text{ storage.}$$

For a five-diagonal matrix (on a $k \times k$ grid) as would arise in the finite-difference discretization of a two-dimensional Laplacian, the complexity is:

$$\mathcal{O}\ (k^3) \text{ flops}$$
$$\text{and}$$
$$\mathcal{O}\ (k^2 \log k) \text{ storage}$$

while, for a tridiagonal or arrowhead matrix, the complexity is:

$$\mathcal{O}\ (n) \text{ work and storage.}$$

Indeed our target complexity for sparse matrix computations is $\mathcal{O}(n) + \mathcal{O}(\tau)$ for a sparse matrix of order $n$ with $\tau$ entries.

# 4 HSL

## 4.1 Mathematical software libraries

The benefits and advantages of using high quality mathematical software libraries include:

- Shorten application development cycle, cutting time-to-market and gaining competitive advantage
- Reduce development costs
- Increase modularity
- More time to focus on specialist aspects of applications
- Improve application accuracy and robustness

## 4.2 HSL

HSL began life as the Harwell Subroutine Library in 1963 and was originally developed by Mike Powell and Mike Hopper as an internal library for users of the IBM mainframe at AERE Harwell. However, the reputation of the Harwell Subroutine Library spread so quickly that it was being sent out to external users on request as early as 1964. HSL packages are now used worldwide by academics and commercial organisations, and are incorporated into a large number of commercial products.

HSL is now a collection of portable, fully documented and tested packages in standard Fortran, primarily written and developed by the Numerical Analysis Group at the Rutherford Appleton Laboratory although some routines have been written by visitors, colleagues and collaborators, and students of staff at RAL. The particular strengths are currently:

- sparse matrix computations
- optimization
- large-scale system solution.

There are two libraries: HSL 2004 and HSL Archive. HSL Archive consists of superseded routines and public domain software and is free for non-commercial use. All codes need a licence although academic and commercial are differentiated.

The most recent version of HSL is called HSL 2004 and was released in January 2004. HSL is marketed by Hyprotech UK, which was acquired by Aspen Technology in May 2002. For further details see: `www.cse.clrc.ac.uk/nag/hsl`

## 4.3 Organization of HSL

The HSL Library is organized into chapters, each identified by two letters

For example,

- MA: matrix routines (solvers)
- MC: matrix routines (manipulation)
- EB: unsymmetric eigensystems

Within each chapter, each package has a 2-digit identifier, generally allocated chronologically, for example:

- `MA48`: package for solving unsymmetric sparse equations
- `MA49`: package for sparse QR factorization and for solving sparse least-squares.

Following the Fortran 77 convention limiting the length of character strings, each subroutine has a six character identifier, for example:

- `MA48AD`: double precision analysis subroutine of `MA48` package
- `MA57BD`: double precision factorize subroutine of `MA57` package.

More recently, the prefix `HSL_` (for example, `HSL_MA48`) has been used to identify Fortran 90 or 95 packages.

The number of routines in the main chapters in HSL is shown in Figure 4.3.

Each package has a specification sheet, a short "demo" test program, and an exhaustive test deck.

| | | |
|---|---|---|
| MA | Matrix solution | 26 |
| MC | Matrix manipulation | 33 |
| ME | Complex matrices | 8 |
| MI | Iterative solvers and preconditioners | 9 |
| MP | MPI packages .. all solvers | 4 |
| E | Eigensystems | 8 |
| V | Optimization | 13 |

Table 4.3: Number of routines in some major chapters of HSL.

Fortran source code is always provided. Version numbers in the form a.b.c have been recently introduced to HSL. Changes to c are very minor, perhaps involving changes to comments in the code. The level b represents minor bug fixes, while at level a we expect more major fixes and perhaps new entries or facilities.

## 4.4 Development of HSL

HSL is both **revolutionary** and **evolutionary**.

By **revolutionary**, we mean that codes have been introduced that are radically different in technique and algorithm design than anything that has preceded them. Examples of this are:

| | | |
|---|---|---|
| `MA18` | First sparse direct code | 1971 |
| `MA27` | First multifrontal code | 1982 |

By **evolutionary**, we mean that some of our codes evolve, sometimes as a result of major changes in programming paradigm and sometimes because of added functionalities. Examples of this morphing are:

$$\begin{gathered}
\texttt{MA18} \longrightarrow \texttt{MA28} \longrightarrow \texttt{MA48} \\
\texttt{MA32} \longrightarrow \texttt{MA42} \longrightarrow \\
\texttt{MA37} \longrightarrow \texttt{MA41} \longrightarrow \\
\texttt{MA17} \longrightarrow \texttt{MA27} \longrightarrow \texttt{MA57}
\end{gathered}$$

We look in more detail at an example of evolution by giving more details for the last example above, namely our flagship code for symmetric sparse systems. The history of our HSL codes for solving the symmetric system with $\mathbf{A} = \mathbf{A^T}$ is shown in Figure 4.3.

- `MA17` ... 1971 (Curtis and Reid)
  - Sparse symmetric
  - $LDL^T$ .. $1 \times 1$ pivots only
  - linked lists
- `MA27` ... 1982 (Duff and Reid)
  - Sparse symmetric indefinite
  - $LDL^T$ .. $1 \times 1$ and $2 \times 2$ pivots
  - Multifrontal
- `MA47` ... 1995 (Duff and Reid)
  - Sparse symmetric indefinite structured
  - $LDL^T$ .. $1 \times 1$ and $2 \times 2$ pivots, with $\begin{matrix} \times & \times \\ \times & 0 \end{matrix}$ and $\begin{matrix} \times & \times \\ \times & 0 \end{matrix}$ pivots
  - Multifrontal
- `MA57` ... 2000 (Duff)
  - See Section 4.5

Figure 4.3: Example of evolution

### 4.5 HSL code `MA57`

To show the sophistication of our recent codes, we show the features of our current flagship code for solving sparse symmetric equations `MA57` in Figure 4.4.

- Analysis (ordering and symbolic factorization)
  - AMD (Approximate Minimum Degree) ordering
- Factorization ($PAP^T \longrightarrow LDL^T$)
  - Factorizes singular matrices
  - Pivoting options (including Schnabel-Eskow)
  - Stop and restart (or discard factors)
  - Option to return or alter pivots
- Solve (Fwd/Bwd substitution)
  - Several entries for error analysis and iterative refinement
  - Multiple rhs (using level 3 BLAS)
  - Partial solve (using $L$, $D$, or $L^T$)

Figure 4.4: `MA57` features.

Additions made to Version 3.x.y of `MA57` are:

- MeTiS nested dissection ordering available as an option
- Automatic choice of ordering ... decision made from matrix characteristics
- Built-in option for scaling matrix (transparent to user)
- Static pivoting option

We show the first and second pages of the specification sheet for `MA57` in Figures 4.5 and 4.6. The first page shows that the structure of the code follows the subdivision of direct solution methods that we discussed earlier. On the second page, we see details of the call for the analysis entry where the number of parameters are reduced by combing control and information parameters into arrays. In a Fortran 90 code, of course, the work arrays can be made internal and dynamic and derived data types can be used to create more structure and further reduce the parameter list.

aspentech CCLRC

# HSL MA57

PACKAGE SPECIFICATION HSL 2004

## 1 SUMMARY

To **solve a sparse symmetric system of linear equations.** Given a sparse symmetric matrix $\mathbf{A}=\{a_{ij}\}_{n\times n}$ and an $n$-vector **b** (or an $n\times s$ matrix **B**), this subroutine solves the system **Ax=b** (**AX=B**). The matrix **A** need not be definite.

The multifrontal method is used. It is a direct method based on a sparse variant of Gaussian elimination and is discussed further by Duff and Reid, ACM Trans. Math. Software **9** (1983), 302-325. A detailed discussion on the MA57 strategy and performance is given by Duff, ACM Trans. Math. Software **30** (2004), 118-144. More recent work on pivoting and scaling strategies is given in the Technical Report RAL-TR-2004-020 by Duff and Pralet. This will be published in the SIAM Journal on Matrix Analysis and Applications and can be obtained from the web site

`http://www.numerical.rl.ac.uk/reports/reports.html`

The MA57 package has a range of options including several sparsity orderings, multiple right-hand sides, partial solutions, error analysis, scaling, a matrix modification facility, and a stop and restart facility. Although the default settings should work well in general, there are several parameters available to enable the user to tune the code for his or her problem class or computer architecture.

**ATTRIBUTES — Version:** 3.0.1. **Types:** Real (single, double) **Calls:** FD15, MC64, MC71, MC47, BLAS routines _GEMM, _TPSV, and _GEMV, and (optionally) METIS_NODEND from the MeTiS package. **Remark:** Supersedes MA27. **Original date:** Original code: September 2000, Version 3.0.0: March 2005. **Origin:** I. S. Duff, Rutherford Appleton Laboratory.

## 2 HOW TO USE THE PACKAGE

### 2.1 Argument lists and calling sequences

There are five entries:

(a) MA57I/ID sets default values for the components of the arrays that hold control parameters. Normally the user will call MA57I/ID prior to any call to MA57A/AD. If non-default values for any of the control parameters are required, they should be set immediately after the call to MA57I/ID.

(b) MA57A/AD accepts the pattern of **A** and chooses pivots for Gaussian elimination using a selection criterion to preserve sparsity. It subsequently constructs subsidiary information for the actual factorization by MA57B/BD. The user may provide the pivot sequence; in which case only the necessary information for MA57B/BD will be generated.

(c) MA57B/BD factorizes a matrix **A** using the information from a previous call to MA57A/AD. The actual pivot sequence used may differ from that of MA57A/AD if **A** is not definite.

(c) MA57C/CD uses the factors generated by MA57B/BD to solve a system of equations **Ax=b** (**AX=B**).

(e) MA57D/DD uses the factors generated by MA57B/BD to solve a system of equations **Ax=b** (**AX=B**) using iterative refinement and (optionally) returning estimates of the error.

(d) MA57E/ED maps the data into new larger size arrays following a failure of MA57B/BD through insufficient storage so that the numerical factorization can be continued using these larger arrays.

A call to MA57C/CD or MA57D/DD must be preceded by a call to MA57B/BD which in turn must be preceded by a call to MA57A/AD. Since the information passed from one subroutine to the next is not corrupted by the second, several calls to MA57B/BD for matrices with the same sparsity pattern but different values may follow a single call to

**All use is subject to licence, see** `http://hsl.rl.ac.uk/acuk/cou.html`
**For any commercial application, a separate licence must be signed.**

HSL 2004 MA57 Version 3.0.1 Documentation date: 21st May 2006 1

Figure 4.5: First page of MA57 specification sheet

**MA57** **HSL 2004**

MA57A/AD, and similarly MA57C/CD or MA57D/DD can be used repeatedly to solve for different right-hand sides **b** (**B**). Note that it would be possible to use MA57A/AD on several matrices before calling MA57B/BD. When we state that parameters "must be unchanged since the call to *subroutine*", we mean a successful call of the routine on the same matrix.

**2.1.1 To set default values of controlling parameters**

*The single precision version*

```
CALL MA57I(CNTL,ICNTL)
```

*The double precision version*

```
CALL MA57ID(CNTL,ICNTL)
```

CNTL is a REAL (DOUBLE PRECISION in the D version) array of length 5 that need not be set by the user. On return it contains default values. For further information see Section 2.2.

ICNTL is an INTEGER array of length 20 that need not be set by the user. On return it contains default values. For further information see Section 2.2.

**2.1.2 To perform symbolic manipulations**

*The single precision version*

```
CALL MA57A(N,NE,IRN,JCN,LKEEP,KEEP,IWORK,ICNTL,INFO,RINFO)
```

*The double precision version*

```
CALL MA57AD(N,NE,IRN,JCN,LKEEP,KEEP,IWORK,ICNTL,INFO,RINFO)
```

N is an INTEGER variable that must be set by the user to the order $n$ of the matrix **A**. It is not altered by the subroutine. **Restriction:** N $\geq$ 1.

NE is an INTEGER variable that must be set by the user to the number of entries input in IRN and JCN. It is not altered by the subroutine. **Restriction:** NE $\geq$ 0.

IRN and JCN are INTEGER arrays of length NE. The user must set them so that each diagonal entry $a_{ii}$ is represented by IRN(k)=$i$ and JCN(k)=$i$ and each pair of off-diagonal entries $a_{ij}$ and $a_{ji}$ is represented by IRN(k)=$i$ and JCN(k)=$j$ or by IRN(k)=$j$ and JCN(k)=$i$. Entries (on or off the diagonal) that are known to be zero can be excluded. Multiple entries are permitted. If IRN(k) or JCN(k) are less than 1 or greater than N the entry is ignored. These arrays are not altered by any of the calls to the MA57 package. They must be preserved by the user between this call and a call to MA57D/DD for the same matrix.

LKEEP is an INTEGER variable that must be set by the user to the length of array KEEP. It might be more efficient to allocate more than the minimum required, say about N to 2*N more space. **Restriction:** LKEEP $\geq$ 5*N+NE+MAX(N,NE)+42.

KEEP is an INTEGER array of length LKEEP. It need not be set by the user and must be preserved between a call to MA57A/AD and subsequent calls to MA57B/BD. If the user wishes to input the pivot sequence, the position of variable $i$ in the pivot order should be placed in KEEP(I), I = 1, 2,..., N and ICNTL(6) should be set to 1. The subroutine may replace the given order by another that gives the same fill-in pattern and virtually identical numerical results.

IWORK is an INTEGER array of length 5*N that is used as workspace.

ICNTL is an INTEGER array of length 20 that contains control parameters and must be set by the user. Default values for the components may be set by a call to MA57I/ID. Details of the control parameters are given in Section 2.2.

INFO is an INTEGER array of length 40 that need not be set by the user. On return from MA57A/AD, a non-negative value for INFO(1) indicates that the subroutine has performed successfully. For nonzero values, see Section



2 MA57 Version 3.0.1 Documentation date: 21st May 2006 HSL 2004

Figure 4.6: Second page of MA57 specification sheet

### 4.6 Parallel codes in HSL

In recent years, we have introduced some parallel codes to HSL. The earliest parallel code was an OpenMP version of `MA41`.

Work on this code (1 ) was later developed by teams originally at RAL and CERFACS and now also at Lyon, ENSEEIHT-IRIT, and Bordeaux to produce the much downloaded `MUMPS` package (2 ). Note that this package is freely available by request to `mumps@cerfacs.fr` but is not in HSL.

MPI-based routines that are available in HSL are in the MP chapter:

| | |
|---|---|
| HSL_MP42 | Multiple front method .. equation entry |
| HSL_MP43 | Multiple front method .. element entry |
| HSL_MP62 | Symmetric element entry multiple front |
| HSL_MP48 | General unsymmetric using singly bordered block diagonal form |

### 4.7 HSL summary

It is impossible to discuss in detail the many sparse codes in HSL in an article of this kind but we present a list of HSL sparse codes in Table 4.4.

| Package | System solved | Algorithm |
|---|---|---|
| `MA38` | Unsymmetric assembled | Multifrontal |
| `MA41` | Unsymmetric assembled | Multifrontal |
| `MA42` | Unsymmetric assembled and unassembled | Frontal |
| `MA43` | Unsymmetric assembled | Frontal |
| `MA45` | Weighted least squares | Normal equations |
| `MA46` | Unsymmetric unassembled | Multifrontal |
| `MA48` | Unsymmetric assembled | Markowitz-Threshold |
| `MA49` | Rectangular assembled | Multifrontal $QR$ |
| `MA55` | Symmetric positive definite | Variable band |
| `MA57` | Symmetric indefinite assembled | Multifrontal |
| `MA62` | Symmetric definite unassembled | Frontal |
| `MA67` | Symmetric indefinite structured | Zero-tracking |

Table 4.4: Some of the sparse matrix codes in HSL that use direct methods. In many cases there is also a version for complex matrices. There are parallel versions of `MA41`, using OpenMP, and of `MA42`, `MA43`, `MA62`, and `MA48` using MPI. An out-of-core multifrontal code will soon be available.

## 5 Summary

The twin aims of our talk are to emphasize the ubiquity of sparse matrices and the availability of high quality codes for solving sparse systems with HSL. We should stress that there are several packages available elsewhere, for example the already mentioned `MUMPS`, although we do not know of a greater concentration of codes than in HSL.

To sum up:

- Sparse matrices occur in very many application areas.

- Sparse direct methods can be used to robustly solve large sparse problems.
- There are many packages available that implement direct methods.
- There are several packages implementing direct methods in HSL (`www.cse.clrc.ac.uk/nag/hsl`).

## References


[1] P. R. Amestoy and I. S. Duff. Vectorization of a multiprocessor multifrontal code. *Int. J. of Supercomputer Applics.*, 3:41–59, 1989.

[2] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM J. Matrix Analysis and Applications*, 23(1):15–41, 2001.

[3] A. R. Curtis and J. K. Reid. Fortran subroutines for the solution of sparse sets of linear equations. Technical Report AERE R 6844, Her Majesty's Stationery Office, London, 1971.

[4] I. S. Duff. MA57 – A new code for the solution of sparse symmetric indefinite systems. Technical Report RAL-TR-2002-024, Rutherford Appleton Laboratory, Oxfordshire, England, 2002. To appear in *ACM Trans. Math. Softw.*

[5] I. S. Duff, A. M. Erisman, C. W. Gear, and J. K. Reid. Sparsity structure and Gaussian elimination. *SIGNUM Newsletter*, 23(2):2–8, April 1988.

[6] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices.* Oxford University Press, Oxford, England, 1986.

[7] I. S. Duff, R. G. Grimes, and J. G. Lewis. Sparse matrix test problems. *ACM Trans. Math. Softw.*, 15(1):1–14, March 1989.

[8] I. S. Duff, R. G. Grimes, and J. G. Lewis. The Rutherford-Boeing Sparse Matrix Collection. Technical Report RAL-TR-97-031, Rutherford Appleton Laboratory, Oxfordshire, England, 1997. Also Technical Report ISSTECH-97-017 from Boeing Information & Support Services, Seattle and Report TR/PA/97/36 from CERFACS, Toulouse.

[9] I. S. Duff and J. K. Reid. MA27 – A set of Fortran subroutines for solving sparse symmetric sets of linear equations. Technical Report AERE R10533, Her Majesty's Stationery Office, London, London, 1982.

[10] I. S. Duff and J. K. Reid. MA47, a Fortran code for direct solution of indefinite sparse symmetric linear systems. Technical Report RAL 95-001, Rutherford Appleton Laboratory, Oxfordshire, England, 1995.

[11] Y. Saad. *Iterative Methods for Sparse Linear Systems Second Edition.* Society for Industrial and Applied Mathematics, 2003.

[12] H. A. van der Vorst. *Iterative Krylov Methods for Large Linear systems.* Cambridge University Press, 2003.