

HSL 2011

A CATALOGUE OF SUBROUTINES

HSL 2011 is a collection of Fortran packages for large scale scientific computation written and developed primarily by the Numerical Analysis Group at the Rutherford Appleton Laboratory. This catalogue contains a complete list of all the packages in HSL 2011 and gives for each one a brief outline of its purpose, method, origin, language, and other attributes. An extensive index allows the appropriate package for a particular task to be identified. Full details of how to obtain a licence for HSL 2011 are provided. This edition supersedes all previous editions.

First edition December 1978
Second edition September 1979
Third edition September 1980
Fourth edition October 1981
Fifth edition January 1984
Sixth edition October 1985
Seventh edition April 1987
Eighth edition February 1988
Ninth edition April 1989
Tenth edition August 1990
Revised tenth edition March 1992
Eleventh edition June 1993
Twelfth edition December 1995
Thirteenth edition October 2000
Fourteenth edition January 2002
Fifteenth edition September 2004
Sixteenth edition September 2007
Revised 6 August 2008
Seventeenth edition January 2011

Contents

1	INTRODUCTION	7
1.1	HSL Mathematical Software Library	7
1.2	Licencing	8
1.3	New in HSL 2011	9
1.4	The language of HSL	11
1.5	The use by HSL of external software libraries	12
1.6	Using HSL facilities	12
1.7	The catalogue	13
1.8	Acknowledgement	14
E	EIGENVALUES AND EIGENVECTORS	15
EA	Eigenvalues and eigenvectors of real symmetric matrices	15
EA16	Compute selected eigenpairs using rational Lanczos method	15
HSL_EA19	Sparse symmetric or Hermitian: leftmost eigenpairs	15
HSL_EA20	fractional power of a sparse self-adjoint positive-definite pencil	16
EA22	Sparse symmetric: simultaneous iteration	16
EA25	Sparse symmetric: Lanczos for the spectrum	16
EB	Eigenvalues and eigenvectors of real general matrices	16
EB13	Sparse unsymmetric: Arnoldi's method	16
EB22	Sparse unsymmetric: subspace iteration	17
EP	Multiprocessor specific packages	17
EP25	Sparse symmetric: Lanczos for the spectrum	17
F	MATHEMATICAL FUNCTIONS	19
FA	Random numbers	19
FA14	Uniform distribution	19
HSL_FA14	Uniform distribution	19
FD	Simple Functions	19
FD15	Real-valued machine constants	19
K	SORTING	21
KB	Sorting numbers	21
KB05	Sort numbers into ascending order using Quicksort	21
KB06	Sort numbers into descending order using Quicksort	21
KB07	Sort numbers into ascending order with indexing using Quicksort	21
KB08	Sort numbers into descending order with indexing using Quicksort	21
HSL_KB22	Sorting reals using the Heapsort method	21
LA	LINEAR PROGRAMMING	23
LA	Linear programming, i.e. minimization of a linear function subject to linear constraints	23
LA04	Sparse linear programming: steepest-edge simplex method	23
LA15	Sparse mathematical programming bases: factorize and update	23

M - LINEAR ALGEBRA	25
MA Solution of linear systems	25
MA38 Sparse unsymmetric system: unsymmetric multifrontal method	25
MA41 Sparse unsymmetric system: unsymmetric multifrontal method	25
MA42 Sparse unsymmetric system: out-of-core frontal method	25
HSL_MA42 Sparse unsymmetric system: out-of-core frontal method	26
HSL_MA42_ELEMENT Unsymmetric finite-element system: out-of-core frontal method (real and complex)	26
MA43 Sparse unsymmetric system: row-by-row frontal method	27
MA44 Over-determined linear system: least-squares solution	27
MA46 Sparse unsymmetric finite-element system: multifrontal	27
MA48 Sparse unsymmetric system: driver for conventional direct method	28
HSL_MA48 Sparse unsymmetric system: driver for conventional direct method	28
MA49 Sparse over-determined system: least squares by QR	28
MA50 Sparse unsymmetric system: conventional direct method	29
MA51 Auxiliary for MA48 and MA50: identify ignored rows or columns in the rectangular or rank-deficient case, compute determinant	29
MA52 Sparse unsymmetric finite-element system: out-of-core multiple front method	29
HSL_MA54 Definite symmetric full matrix: partial or complete factorization and solution	30
HSL_MA55 Band symmetric positive-definite system	31
MA57 Sparse symmetric system: multifrontal method	31
HSL_MA57 Sparse symmetric system: multifrontal method	31
MA60 Iterative refinement and error estimation	31
MA61 Sparse symmetric positive-definite system: incomplete factorization	32
MA62 Sparse symmetric finite-element system: out-of-core frontal method	32
HSL_MA64 Indefinite symmetric full matrix: partial or complete factorization and solution	33
MA65 Unsymmetric banded system of linear equations	34
MA67 Sparse symmetric system, zeros on diagonal: blocked conventional	34
MA69 Unsymmetric system whose leading subsystem is easy to solve	34
HSL_MA69 Unsymmetric system whose leading subsystem is easy to solve	35
MA72 Sparse symmetric finite-element system: out-of-core multiple front method	35
HSL_MA74 Unsymmetric full matrix: partial or complete factorization and solution	35
MA75 Sparse over-determined system: weighted least squares	36
HSL_MA77 Sparse symmetric system: multifrontal out of core	37
HSL_MA78 Sparse unsymmetric finite-element system: multifrontal out of core	37
HSL_MA79 Sparse symmetric system: mixed precision	38
HSL_MA86 Sparse symmetric indefinite system using OpenMP	38
HSL_MA87 Sparse symmetric positive-definite system using OpenMP	39
MC Computations with matrices and vectors	39
MC13 Permute a sparse matrix to block triangular form	39
MC21 Permute a sparse matrix to put entries on the diagonal	39
MC22 Permute a sparse matrix given row and column permutations	40
MC25 Permute a sparse matrix to block triangular form	40
MC26 Sparse rectangular matrix: compute normal matrix	40
MC29 Sparse unsymmetric matrix: calculate scaling factors	40
MC30 Sparse symmetric matrix: calculate scaling factors	40
MC33 Sparse unsymmetric matrix: permute to bordered block triangular form	41
MC34 Sparse symmetric structure: expand from lower triangle	41
HSL_MC34 Sparse symmetric structure: expand from lower triangle	41
MC37 Sparse symmetric matrix: represent as a sum of element matrices	41
MC38 Sparse rectangular matrix held by columns: transpose	41

MC44	Unassembled finite-element matrix: generate the element or supervariable connectivity graph	41
MC46	Sparse rectangular matrix held by rows: transpose	42
MC47	Sparse symmetric pattern: approximate minimum-degree ordering allowing dense rows	42
MC53	Generate an ordering for finite-element matrices within a subdomain	42
MC54	Write a sparse matrix in Rutherford-Boeing format	43
MC55	Write a supplementary file in Rutherford-Boeing format	43
MC56	Read a file or a supplementary file held in Rutherford-Boeing format	43
HSL_MC56	Read a file containing a sparse matrix held in format	43
MC57	Assemble a set of finite-element matrices	43
MC58	Estimate rank and find independent rows/columns of a sparse unsymmetric or rectangular matrix	44
MC59	Sort a sparse matrix to an ordering by columns	44
MC60	Sparse symmetric pattern: reduce the profile and wavefront	44
MC61	Straightforward interface to MC60	44
MC62	Generate a row ordering for a row-by-row frontal solver	45
MC63	Generate an element assembly ordering for a frontal solver	45
MC64	Permute and scale a sparse unsymmetric matrix to put large entries on the diagonal	45
HSL_MC64	Permute and scale a sparse unsymmetric or rectangular matrix to put large entries on the diagonal	46
HSL_MC65	Construct and manipulate matrices in compressed sparse row format	46
HSL_MC66	Permute an unsymmetric sparse matrix to singly bordered blocked diagonal form	46
MC67	Refine a profile-reducing permutation of a symmetric matrix	47
HSL_MC68	Symmetric sparse matrix: compute elimination orderings	47
HSL_MC69	Matrix format converter	47
MC71	Unsymmetric matrix: estimate 1-norm	48
MC72	Full unsymmetric matrix: calculate scaling factors	48
HSL_MC73	Sparse symmetric matrix: compute Fiedler vector and permute to reduce the profile and wavefront	48
MC75	Sparse unsymmetric matrix: estimate condition number	49
MC77	Sparse unsymmetric matrix: calculate scaling factors	49
HSL_MC78	Analysis phase in Cholesky algorithm	49
HSL_MC79	Sparse matrix: maximum matching and Dulmage-Mendelsohn decomposition	49
ME	Solution of complex linear systems and other calculations for complex matrices	50
ME22	Permute a sparse matrix given row and column permutations	50
ME38	Sparse unsymmetric system: unsymmetric multifrontal method	50
ME42	Sparse unsymmetric system: out-of-core frontal method	51
ME43	Sparse unsymmetric system: row-by-row frontal method	51
ME48	Sparse unsymmetric system: driver for conventional direct method	51
ME50	Sparse unsymmetric system: conventional direct method	51
ME57	Sparse Hermitian or complex symmetric: multifrontal method	52
HSL_ME57	Sparse solver for complex symmetric or Hermitian linear systems	52
ME62	Sparse Hermitian or complex symmetric finite-element system: out-of-core frontal method	52
MF	Computations with complex matrices and vectors	53
MF29	Sparse unsymmetric matrix: calculate scaling factors	53
MF30	Sparse symmetric matrix: calculate scaling factors	54
MF64	Permute and scale a sparse complex unsymmetric matrix to put large entries on the diagonal	54
MF71	Unsymmetric matrix: estimate 1-norm	54
MI	Iterative methods for sparse linear systems	54
HSL_MI02	Symmetric possibly-indefinite system: SYMMBK method	54
MI11	Unsymmetric system: incomplete LU factorization	55

MI12 Unsymmetric system: approximate-inverse preconditioner	55
HSL_MI13 Preconditioners for saddle-point systems	55
MI15 Unsymmetric system: flexible GMRES	56
HSL_MI20 Unsymmetric system: algebraic multigrid preconditioner	56
MI21 Symmetric positive-definite system: conjugate gradient method	57
MI23 Unsymmetric system: CGS (conjugate gradient squared) method	57
MI24 Unsymmetric system: GMRES (generalized minimal residual) method	57
MI25 Unsymmetric system: BiCG (BiConjugate Gradient) method	58
MI26 Unsymmetric system: BiCGStab (BiConjugate Gradient Stabilized) method	58
HSL_MI27 Projected preconditioned conjugate gradient method for saddle-point systems	58
HSL_MI31 Symmetric positive-definite system: conjugate gradient method, stopping according to the A-norm of the error	59
MP Multiprocessor specific packages	59
HSL_MP01 MPI constants	59
HSL_MP42 Unsymmetric finite-element system: multiple-front method, element entry	59
HSL_MP43 Sparse unsymmetric system: multiple-front method, equation entry	60
HSL_MP48 Sparse unsymmetric system: parallel direct method	61
HSL_MP54 Parallel Cholesky solver	61
HSL_MP62 Symmetric finite-element system: multiple-front method	61
O - INPUT/OUTPUT	63
OF File management	63
HSL_OF01 Fortran virtual memory	63
P - POLYNOMIAL AND RATIONAL FUNCTIONS	65
PA Zeros of polynomials	65
PA16 Complex coefficients: all roots by the method of Madsen and Reid	65
PA17 Real coefficients: all roots by the method of Madsen and Reid	65
Y - TEST PROGRAM GENERATORS	67
YM Generate test programs for chapter M of the library	67
YM11 Generate a random sparse matrix	67
Z - FORTRAN SYSTEM FACILITIES	69
ZB Array allocation	69
HSL_ZB01 Reallocate an array	69
ZD Derived types	69
HSL_ZD11 Derived type for sparse matrix storage schemes	69
Index	70

Chapter 1

INTRODUCTION

1.1 HSL Mathematical Software Library

HSL (formerly the Harwell Subroutine Library) is a collection of ISO Fortran packages for large-scale scientific computation written and developed by the Numerical Analysis Group at the STFC Rutherford Appleton Laboratory and by other experts and collaborators. The Library was started in 1963 and was originally used at the Harwell Laboratory on IBM mainframes running under OS and MVS. Over the years, the Library has evolved and has been extensively used on a wide range of computers, from CRAY supercomputers to modern PCs. HSL now offers users a high standard of reliability and has an international reputation as a source of robust and efficient numerical software. Among its best known codes are those for the solution of sparse linear systems of equations and sparse eigenvalue problems.

HSL is arranged in two parts: the main library, HSL 2011 and an archive, HSL Archive. The HSL Archive comprises older packages that were part of previous releases of HSL, many of which have been superseded by more modern codes.

This catalogue is for HSL 2011; an online catalogue for HSL Archive is available at <http://www.hsl.rl.ac.uk/archive/index.html>

Enquiries

All inquiries regarding HSL and its usage should be addressed to the HSL Manager:

Dr Jonathan Hogg,

STFC Rutherford Appleton Laboratory,

Atlas Centre,

Harwell Oxford,

Didcot,

Oxfordshire,

OX11 0QX, UK.

Tel: 01235 44 6642 (UK) +44 1235 44 6642 (International)

E-mail: hsl@stfc.ac.uk

1.2 Licencing

A licence is required prior to making any use of any HSL package.

We currently offer three types of licence for HSL 2011 packages:

Academic HSL packages are available without charge for personal academic research and teaching. Full details, including how to access HSL packages as an academic user, are available at

<http://www.hsl.rl.ac.uk/>

Commercial (per seat) For use of HSL packages for work within a company, a per seat licence is required. The HSL Manager (see Section 1.1) should be contacted for current prices.

Commercial (incorporation) For incorporation of any HSL package into software that is distributed to a third party (whether or not a fee is involved), an incorporation licence is required. Pricing is done on a case-by-case basis. The HSL Manager (see Section 1.1) should be contacted for details.

Free use of HSL Archive

The HSL Archive packages are available without charge to anyone, so long as they are not then supplied to a third party (whether or not a fee is involved). Access to the HSL Archive is by way of the HSL web page

<http://www.hsl.rl.ac.uk/archive>

Do you have a valid licence?

If you have any doubt whether you have a valid HSL licence, or whether your licence covers your intended use, please contact the HSL Manager (see Section 1.1) promptly for confirmation or advice.

Acknowledging the use of HSL

It is a condition of use that all publications written by those using HSL free of charge that include results obtained with the help of one or more HSL 2011 or HSL Archive packages shall acknowledge the use of the packages. Users should reference HSL 2011 as follows:

HSL (2011). *A collection of Fortran codes for large scale scientific computation.*

<http://www.hsl.rl.ac.uk/hsl2011>

1.3 New in HSL 2011

The new packages in HSL 2011 are:

HSL_EA20 computes the product of the fractional power of a self-adjoint sparse positive-definite matrix and a vector. The generalized Lanczos method for positive-definite matrix pencils is used.

HSL_MA64 is a kernel code for the symmetric indefinite multifrontal method. It performs full or partial factorizations and solutions of dense sets of indefinite linear equations. The matrix may be real or complex symmetric or Hermitian.

HSL_MA79 solves one or more sets of sparse real symmetric linear systems using mixed precision. The matrix may be either positive definite or indefinite.

HSL_MA86 solves one or more sets of sparse symmetric indefinite linear systems. This package is designed for use on a **multicore machine**. The matrix may be real or complex symmetric or Hermitian.

HSL_MA87 solves one or more sets of sparse symmetric positive-definite linear systems. This package is designed for use on a **multicore machine**. The matrix may be real symmetric or Hermitian.

HSL_MC34 generates the expanded structure for a matrix with a symmetric sparsity pattern given the structure for the lower triangular part. The matrix may be real or complex symmetric, real or complex skew-symmetric, or Hermitian.

HSL_MC56 reads a matrix from a Rutherford-Boeing formatted file. This package is also available as part of HSL Archive.

HSL_MC69 converts matrices from a variety of sparse matrix formats to HSL standard sparse matrix format. This format is used by a number of new HSL 2001 packages.

HSL_MC78 performs tasks required in the analyse phase of a symmetric sparse direct solver.

HSL_MC79 computes the Dulmage-Mendelsohn decomposition of a rectangular sparse matrix.

HSL_ME57 solves one or more sets of sparse complex symmetric or Hermitian linear systems. The matrix may be either positive definite or indefinite.

HSL_MI27 uses the projected preconditioned conjugate gradient method to solve real saddle-point systems.

HSL_MP54 is a kernel code for the symmetric positive-definite multifrontal method. It performs full or partial factorizations and solutions of dense sets of positive-definite linear equations. OpenMP is used.

Significant improvements have been made to the following packages:

HSL_EA19 computes the left-most eigenvalues and corresponding eigenvectors of a sparse real symmetric (or Hermitian) generalized eigenvalue problem. In HSL 2011, algorithmic changes have been made to improve the reliability.

HSL_MA54 is a kernel code for the symmetric positive-definite multifrontal method. It performs full or partial factorizations and solutions of dense sets of positive-definite linear equations. In HSL 2011, some OpenMP support has been added and long integer addressing.

HSL_MA77 solves one or more sets of sparse symmetric equations using an out-of-core multifrontal method. In HSL 2011, the matrix may be indefinite. Some OpenMP support added.

HSL_MC64 permutes and scales a sparse real matrix that may be rectangular to put large entries on the diagonal. The main change in HSL 2011 is that both a row and column permutation are provided in all cases (symmetric, unsymmetric, and rectangular).

HSL_MC68 computes elimination orderings that are suitable for use with a sparse direct solver. Changes in HSL 2011 include an improved AMD ordering that handles dense rows and the user interface has changed to use the HSL standard sparse matrix format.

ME57 solves one or more sets of sparse complex symmetric or Hermitian linear systems. The matrix may be either positive definite or indefinite. In HSL 2011, the code has been modified to bring it more closely in line with the real code **MA57**.

MI11 forms an incomplete factorization of a real sparse unsymmetric matrix. The performance of the code has been substantially improved in HSL 2011.

These packages have been moved from the main library to HSL Archive:

HSL_AD02

HSL_DC05, DD14

EA23, EC23

NS12, NS23

TD22

VA08, VA34, VA35, VE08, VE10, HSL_VE12, HSL_VE13, HSL_VE19, VE24

HSL_VF05, HSL_VF06, HSL_VH01,

HSL_ZD13

Optimization packages

Most of the optimization packages that were previously part of the main HSL library have been moved to the HSL Archive. The Group's most recent optimization packages are available through the GALAHAD library; details are available at

<http://www.cse.scitech.ac.uk/nag/galahad>

1.4 The language of HSL

HSL 2011 is arranged as collections of threadsafe ISO Fortran 77 and Fortran 95 packages, each of which consists of either a single program unit or a set of program units. Almost all the Fortran 77 program units are subroutines, but there are also some functions. The Fortran 95 program units are mostly modules, but there are some external procedures. Each package performs a basic numerical task and has been designed to be incorporated into programs. Each has its own specification document, which gives full details about how to use the package, and is available as a PDF file.

All the HSL Fortran 77 packages adhere to the Fortran 77 Standard, with the following exceptions:

- a few use `COMPLEX*16` declarations and corresponding arithmetic operations and intrinsic procedure calls,
- there are long names for calls to MeTiS in a number of packages (see Section 1.5),
- FD15 calls the Fortran 95 inquiry functions `epsilon`, `huge`, `tiny`, and `radix`.

These extensions are widely available in Fortran 77 compilers, and may be regarded as *de facto* extensions of the Standard.

Most of the Fortran 95 packages adhere to the Fortran 95 Standard. Some use allocatable arrays as components of structure and arguments of procedures. Since 1998, this has been a standardized extension (ISO/IEC TR 15581) of Fortran 95 and is now universally available. It is included in Fortran 2003.

Use of MPI

The packages of the EP and MP sections of HSL 2011 contain packages for parallel programming in the presence of MPI. These packages are for use only where MPI is installed.

Use of OpenMP

A number of recent packages use OpenMP to provide parallelism for shared memory environments. To run in parallel, OpenMP must be enabled at compilation time by using the correct compiler flag (usually some variant of `-openmp`). The number of threads may be controlled at runtime by setting the environment variable `OMP_NUM_THREADS`.

Use of reverse communication

Several HSL packages are structured so that the user has to call a procedure repeatedly and perform specified actions between the calls. The procedure uses the value of an argument to indicate exactly what action is needed. The process is known as *reverse communication*. It is used, for example, when the procedure needs the value of the user's function at a certain point or when it needs to have a vector multiplied by the user's matrix. The advantage is that the user has access to the whole data environment at the point of call and can choose how to perform the required task. It also gives flexibility for the user to include tests on the progress and take alternative action if it is too slow.

Matlab interfaces for selected HSL packages

We have facilities for building Matlab (MEX) interfaces to several HSL 2011 packages. The software has been tested under Linux, and the installation assumes that the gcc and g95 compilers are available. The files have also been successfully installed under Solaris UNIX. Currently, we do not have working versions for Microsoft or Apple operating systems.

More information about the selected packages and the installation can be found at

<http://www.hsl.rl.ac.uk/matlab.html>

1.5 The use by HSL of external software libraries

Use of BLAS

Many HSL routines make use of the BLAS (Basic Linear Algebra Subprograms). Fortran code for the BLAS is included with the distribution of HSL 2011 but users are advised that, for HSL packages to be efficient, they should **always** use BLAS tuned for their machine. Where vendor-supplied BLAS are not available, we recommend ATLAS (Automatically Tuned Linear Algebra Software), see <http://math-atlas.sourceforge.net/> or GotoBLAS2, see <http://www.tacc.utexas.edu/tacc-projects/gotoblas2/>

Use of LINPACK and LAPACK

Several HSL packages make use of codes from the public domain libraries LAPACK and LINPACK. Fortran code for these is included with the distribution of HSL 2011 and are suitable, but users are strongly encouraged to use optimized versions from the vendor or the public domain.

Use of MeTiS

A small number of HSL packages offer the option of using the MeTiS ordering library (see <http://glaros.dtc.umn.edu/gkhome/views/metis>) and may give the best results by doing so. For these packages, included with the distribution of HSL 2011 is a stub subroutine that simply returns an error condition if the option is chosen. If the option of using MeTiS is to be invoked, the stub must be replaced by the MeTiS code, which must be acquired separately.

1.6 Using HSL facilities

Classifications

HSL packages are classified into groups, each associated with a major area of numerical mathematics, such as the solution of sets of linear of equations, calculation of eigenvalues and eigenvectors, or minimization of functions of several variables. Within each group, there is a further classification into subgroups. A complete list can be found in the contents pages.

Naming convention

Each package is associated with a sequence of four characters. The first two characters are always alphabetic and identify the group and subgroup to which the package belongs; for example, the MA series solves systems of linear equations. The next two characters are numeric.

Each Fortran 77 package has the sequence of four characters as its name. Each program unit within the package has its name starting with the four characters. The fifth character, which is always alphabetic, is used to distinguish between the program units. The sixth character of the name (or its absence) identifies the version. The possibilities are:

Absent	Single precision.
D	Double precision.
I	Integer.
L	Long integer.
C	Single precision complex.
Z	Double precision complex.

The name of each Fortran 95 package starts HSL_ and is followed by the sequence of four characters. In most cases, this is its whole name, but occasionally this is followed by a qualifier such as `_ELEMENT`. The package usually consists of a set of modules, but may also contain some external procedures. Each module name starts with the package name and is followed by a qualifier to identify the version. The possibilities are:

<code>single</code>	Single precision.
<code>double</code>	Double precision.
<code>integer</code>	Integer.
<code>long_integer</code>	Long integer.
<code>complex</code>	Single precision complex.
<code>double_complex</code>	Double precision complex.

With the appearance of hardware for which single precision working is significantly faster than double precision, the main use of the single precision versions is likely to be in mixed precision computations where the single precision result is computed and then refined to double precision accuracy.

Marking of versions

Since HSL 2004, we have marked each version of a package with a version label of the form *l.m.n* where *l* is a digit that labels major changes which may affect the interface, *m* is a technical change, usually a bug fix, that does not affect the interface, and *n* is an editorial change such as the correction of a spelling error in an output format.

We have not attempted to mark all changes prior to HSL 2004 with version labels, but a few packages with significant changes were labelled 2.0.0 at the start of HSL 2004.

This catalogue shows the version labels that were current at the time it was constructed (30 January 2011).

Specification documents

More detailed instructions on how to use an individual package is given in its specification document. Each is available as a PDF file. The specification document contains a package summary similar to that in this catalogue, a detailed description of how to set up the arguments, a short description of the method and an example of using the package on a simple problem. It is assumed that the user has a basic knowledge of Fortran.

1.7 The catalogue

This catalogue provides an overview of HSL 2011. It lists the names and purposes of each package. The packages are listed in alphabetical order using the HSL naming convention (see Section 1.6), ignoring HSL_ in the case of a Fortran 95 package. At the end of each entry, certain attributes are listed; these have been presented in a formal manner for consistency:

Version The label for the current version.

Types The Fortran types supported for the principal arguments.

Remark Information not otherwise covered.

Precision This entry is used only for those packages for which we recommend at least 8-byte arithmetic.

Calls This attribute lists the HSL packages that are used or called directly, names of procedures that must be provided by the user, and names of subroutines from LAPACK, LINPACK, or the Basic Linear Algebra Subroutine (BLAS) library (public domain libraries). For LAPACK, LINPACK, and BLAS, we replace the letter that specifies the precision by an underscore; for example, we use `_DOT` as a generic name for `SDOT` and `DDOT`. Note that only first level references are listed so that deeper level references must be traced by looking at the entries for the packages that are referenced directly.

Language This shows the language in which the package is written.

Date This gives the approximate date that the package was introduced into HSL or underwent its last major revision.

Origin The author's name and the place of origin of the package is given. Most of the packages coming from outside have undergone some modification for use in HSL and therefore the responsibility for maintaining their good working rests with the HSL team. Queries concerning HSL packages should not be directed to the authors personally but should be made to the HSL Manager (see Section 1.1).

Licence If a third-party licence for the package is available without charge, this is indicated.

1.8 Acknowledgement

Since October 2007, much of the research that lies behind the development of new HSL packages has been funded by the EPSRC research grants EP/F006535/1 and EP/E053351/1.

E - EIGENVALUES AND EIGENVECTORS

EA Eigenvalues and eigenvectors of real symmetric matrices

EA16 Compute selected eigenpairs using rational Lanczos method

EA16 uses an implicitly restarted block Lanczos method or rational Lanczos method to compute selected eigenpairs of large sparse real eigenproblems. EA16 may be used to compute approximate eigenpairs for the following problems:

1. Standard eigenvalue problem : $\mathbf{Ax} = \lambda\mathbf{x}$, \mathbf{A} sparse symmetric.
2. Generalised eigenvalue problem : $\mathbf{Ax} = \lambda\mathbf{Mx}$, \mathbf{A} sparse symmetric, \mathbf{M} sparse symmetric positive (semi) definite.
3. Buckling problem : $\mathbf{Ax} = \lambda\mathbf{Mx}$, \mathbf{A} sparse symmetric positive (semi) definite, \mathbf{M} sparse symmetric.

ATTRIBUTES — **Version:** 1.2.1. **Types:** Real (single, double). **Precision:** At least 8-byte arithmetic is recommended. **Calls:** `_LAMCH`, `_LASET`, `_LARNV`, `_SYEV`, `_GESVD`, `_SBEV`, `_LACPY`, `_LARTG`, `_LARFG`, `_RSCL` (LAPACK), `_NRM2`, `_DOT`, `_I_AMAX`, `_ASUM`, `_SCAL`, `_COPY`, `_GEMM`, `_AXPY`, `_GEMV`, `_GER`, `_TBSV` (BLAS), KB07, KB08 (HSL). **Language:** Fortran 77. **Date:** March 2000 . **Origin:** K. Meerbergen and J.A. Scott, Rutherford Appleton Laboratory.

HSL_EA19 Sparse symmetric or Hermitian: leftmost eigenpairs

HSL_EA19 uses a subspace iteration method to compute the leftmost eigenvalues and corresponding eigenvectors of a real symmetric (or Hermitian) operator \mathbf{A} acting in the n -dimensional real (or complex) Euclidean space R^n , or, more generally, of the problem

$$\mathbf{Ax} = \lambda\mathbf{Bx},$$

where \mathbf{B} a real symmetric (or Hermitian) positive-definite operator. By applying HSL_EA19 to $-\mathbf{A}$, the user can compute the rightmost eigenvalues of \mathbf{A} and the corresponding eigenvectors. HSL_EA19 does not perform factorizations of \mathbf{A} or \mathbf{B} and thus is suitable for solving large-scale problems for which a sparse direct solver for factorizing \mathbf{A} or \mathbf{B} is either not available or is too expensive.

The convergence may be accelerated by the provision of a symmetric positive-definite operator \mathbf{T} that approximates the inverse of $(\mathbf{A} - \sigma\mathbf{B})$ for a value of σ that does not exceed the leftmost eigenvalue. Computation time may also be reduced by supplying vectors that are good approximations to some of the eigenvectors.

ATTRIBUTES — **Version:** 1.3.2. **Types:** Real (single, double), Complex (single, double). **Language:** Fortran 95 + TR 15581 (allocatable components). **Calls:** Real: `_COPY`, `_AXPY`, `_DOT`, `_NRM2`, `_SYRK`, `_GEMM`, `_SYGV`, `ILAENV`, KB07. Complex: `_COPY`, `_AXPY`, `_DOTC`, `SCNRM2/DZNRM2`, `_HERK`, `_GEMM`, `_HEGV`, `ILAENV`, KB07. **Date:** July 2007. **Origin:** E. Ovtchinnikov, Harrow School of Computer Science, University of Westminster, London, UK; and J. K. Reid.

HSL_EA20 fractional power of a sparse self-adjoint positive-definite pencil

HSL_EA20 is a suite of Fortran 95 procedures for computing the product of the s -root of a sparse self-adjoint positive-definite matrix by a vector using a scalar product derived by a second symmetric positive-definite matrix. Given two $n \times n$ symmetric positive-definite matrices \mathbf{A} and \mathbf{M} , and a vector \mathbf{u} , the package uses the Lanczos method, applied to the matrix pencil (\mathbf{M}, \mathbf{A}) , to approximate

$$(\mathbf{M}^{-1}\mathbf{A})^s \mathbf{u}, \quad s \in (-1, 1).$$

Reverse communication is used. Control is returned to the user for the products of \mathbf{A} with a vector \mathbf{z} , of \mathbf{M} with a vector \mathbf{x} , or of \mathbf{M}^{-1} with a vector \mathbf{w} .

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real, Double. **Calls:** _DOT, _PTEQR, _GEMV. **Date:** July 2010. **Origin:** M. Arioli, Rutherford Appleton Laboratory. **Language:** Fortran 95.

EA22 Sparse symmetric: simultaneous iteration

Given a real symmetric matrix $\mathbf{A} = \{a_{ij}\}_{n \times n}$ of order n , calculates the k largest eigenvalues, $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_k$ and their associated eigenvectors $\mathbf{x}_i, i = 1, 2, \dots, k$, where $\mathbf{A}\mathbf{x}_i = \lambda_i\mathbf{x}_i$. The subroutine uses the method of simultaneous iteration and also allows the user to take advantage of sparsity. Eigensolutions from other parts of the spectrum can be obtained by using shifts of the form $\mathbf{A} - \beta\mathbf{I}$ optionally combined with inversion.

ATTRIBUTES — **Version:** 1.1.1. **Types:** Real (single, double). **Calls:** FA14. **Language:** Fortran 77. **Date:** 1976, revised May 2001. **Remark:** EA22 is a threadsafe version of EA12. **Origin:** I.S. Duff, Harwell.

EA25 Sparse symmetric: Lanczos for the spectrum

This subroutine uses the Lanczos algorithm to compute the part of the spectrum of a large symmetric matrix \mathbf{A} that lies in a specified interval, that is, it computes eigenvalues without regard to multiplicities. The user need only provide \mathbf{A} in the form of code which computes $\mathbf{u} + \mathbf{A}\mathbf{v}$ for any given vectors \mathbf{u} and \mathbf{v} . Auxiliary calls allow corresponding eigenvectors to be found.

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double). **Remark:** Supersedes EA14. **Precision:** At least 8-byte arithmetic is recommended. **Calls:** FA14. **Language:** Fortran 77. **Date:** 1982, revised May 2001. **Remark:** EA25 is a threadsafe version of EA15. **Origin:** J.K. Reid, Harwell.

EB Eigenvalues and eigenvectors of real general matrices

EB13 Sparse unsymmetric: Arnoldi's method

Given a real unsymmetric $n \times n$ matrix $\mathbf{A} = \{a_{ij}\}$, this routine uses Arnoldi based methods to calculate the r eigenvalues $\lambda_i, i = 1, \dots, r$, that are of largest absolute value, or are right-most, or are of largest imaginary parts. The right-most eigenvalues are those with the most positive real part. There is an option to compute the associated eigenvectors $\mathbf{y}_i, i = 1, \dots, r$, where $\mathbf{A}\mathbf{y}_i = \lambda_i\mathbf{y}_i$. The routine may be used to compute the left-most eigenvalues of \mathbf{A} by using $-\mathbf{A}$ in place of \mathbf{A} .

The Arnoldi methods offered by EB13 are:

- (1) The basic (iterative) Arnoldi method.
- (2) Arnoldi's method with Chebyshev acceleration of the starting vectors.
- (3) Arnoldi's method applied to the preconditioned matrix $p_l(\mathbf{A})$, where p_l is a Chebyshev polynomial.

Each method is available in blocked and unblocked form.

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double). **Calls:** FA14, KB06, _DOT, _NRM2, _AXPY, _COPY, _SCAL, _GER, _GEMV, and _GEMM. **Language:** Fortran 77. **Date:** December 1993. **Origin:** J.A. Scott, Rutherford Appleton Laboratory.

EB22 Sparse unsymmetric: subspace iteration

Given a real unsymmetric $n \times n$ matrix $\mathbf{A} = \{a_{ij}\}$, this routine uses subspace iteration to calculate the r eigenvalues λ_i , $i = 1, 2, \dots, r$, that are right-most, left-most, or are of largest modulus. The right-most (respectively, left-most) eigenvalues are the eigenvalues with the most positive (respectively, negative) real part. A second entry will return the associated eigenvectors \mathbf{y}_i , $i = 1, 2, \dots, r$, where $\mathbf{A}\mathbf{y}_i = \lambda_i\mathbf{y}_i$. The routine may also be used to calculate a group of eigensolutions elsewhere in the spectrum.

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double). **Calls:** FA14, KB06, _DOT, _NRM2, _COPY, _SCAL, _GER, _GEMV, and _GEMM. **Language:** Fortran 77. **Date:** 1991, revised April 2001. **Remark:** EB22 is a threadsafe version of EB12. **Origin:** I.S. Duff and J.A. Scott, Rutherford Appleton Laboratory.

EP Multiprocessor specific packages

EP25 Sparse symmetric: Lanczos for the spectrum

This subroutine uses the Lanczos algorithm to compute in parallel the part of the spectrum of a large symmetric matrix \mathbf{A} that lies in a specified interval, that is, it computes eigenvalues without regard to multiplicities. The user is required to partition the vectors into contiguous sections of similar sizes, each residing on a separate process. He or she must provide parallel code that computes $\mathbf{u} + \mathbf{A}\mathbf{v}$ for any given vectors \mathbf{u} and \mathbf{v} . The partitions should be chosen to make this computation straightforward and rapid.

Auxiliary calls allow corresponding eigenvectors to be found. In this case, the user is responsible for storing each vector \mathbf{v} and restoring it during the eigenvector calculation.

MPI is used for message passing. The system must be homogeneous, that is, all the processes must be identical.

ATTRIBUTES — **Version:** 1.2.0. **Types:** Real (single, double). **Calls:** FA14. **Date:** April 2004. **Remark:** EP25 is a parallel version of EA25. **Origin:** J.K.Reid, Rutherford Appleton Laboratory.

F - MATHEMATICAL FUNCTIONS

FA Random numbers

FA14 Uniform distribution

This function generates uniformly distributed pseudo-random numbers. Random numbers are generated in the ranges $0 < \xi < 1$, $-1 < \eta < 1$ and random integers in $1 \leq k \leq N$ where N is specified by the user.

A multiplicative congruent method is used where a 31 bit generator word g is maintained. On each call to the subroutine $g_n + 1$ is updated to $7^5 g_n \bmod (2^{31} - 1)$; the initial value of g is 1. Depending upon the type of random number required the following are computed $\xi = g_{n+1}/(2^{31} - 1)$; $\eta = 2\xi - 1$ or $k = \text{int.part}\{\xi N\} + 1$.

The subroutine also provides the facility for saving the current value of the generator word and for re-starting with any specified value.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** None. **Language:** Fortran 77. **Date:** 1979, revised March 2001. **Remark:** FA14 is a threadsafe version of FA04. **Origin:** C.R. Kirby and C.L. Winskill, Harwell.

HSL_FA14 Uniform distribution

This package generates uniformly distributed pseudo-random numbers. Random reals are generated in the range $0 < \xi < 1$ or the range $-1 < \eta < 1$ and random integers in the range $1 \leq k \leq N$ where N is specified by the user.

A multiplicative congruent method is used where a 31 bit generator word g is maintained. On each call to a procedure of the package, $g_n + 1$ is updated to $7^5 g_n \bmod (2^{31} - 1)$; the initial value of g is $2^{16} - 1$. Depending upon the type of random number required the following are computed $\xi = g_{n+1}/(2^{31} - 1)$; $\eta = 2\xi - 1$ or $k = \text{int.part}\{\xi N\} + 1$.

The package also provides the facility for saving the current value of the generator word and for restarting with any specified value.

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double). **Remark:** HSL_FA14 is a Fortran 95 version of FA14. **Calls:** None. **Language:** Fortran 95. **Date:** 1995, revised March 2001. **Remark:** HSL_FA14 is a threadsafe version of HSL_FA04. **Origin:** N.I.M. Gould and J.K. Reid, Rutherford Appleton Laboratory.

FD Simple Functions

FD15 Real-valued machine constants

This function supplies real-valued machine constants relating to the floating-point storage and arithmetic of the machine in use.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** None. **Language:** Fortran 95. **Date:** February 2005.

K - SORTING

KB Sorting numbers

KB05 Sort numbers into ascending order using Quicksort

To sort an array of numbers into ascending order. The quicksort algorithm is used.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double), Integer. **Calls:** None. **Language:** Fortran 77. **Date:** April 1980. **Origin:** C. Birch, Harwell.

KB06 Sort numbers into descending order using Quicksort

To sort an array of numbers into descending order. The quicksort algorithm is used.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double), Integer. **Calls:** None. **Language:** Fortran 77. **Date:** April 1980. **Origin:** C. Birch, Harwell.

KB07 Sort numbers into ascending order with indexing using Quicksort

To sort an array of numbers into ascending order maintaining an index array to preserve a record of the original order. The quicksort algorithm is used.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double), Integer. **Calls:** None. **Language:** Fortran 77. **Date:** April 1980. **Origin:** C. Birch, Harwell.

KB08 Sort numbers into descending order with indexing using Quicksort

To sort an array of numbers into descending order maintaining an index array to preserve a record of the original order. The quicksort algorithm is used.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double), Integer. **Calls:** None. **Language:** Fortran 77. **Date:** April 1980. **Origin:** C. Birch, Harwell.

HSL_KB22 Sorting reals using the Heapsort method

HSL_KB22 is a suite of Fortran 95 procedures for successively arranging a set of real numbers, a_1, a_2, \dots, a_n , in order of increasing size using the Heapsort method of J. W. J. Williams. At the k -th stage of the method the k -th smallest member of the set is found. The method is particularly appropriate if it is not known in advance how many smallest members of the set will be required as the Heapsort method is able to calculate the $k+1$ st smallest member of the set efficiently once it has determined the first k smallest members. The method is guaranteed to sort all n numbers in $O(n \log n)$ operations. If a complete sort is required, the Quicksort algorithm, KB05, may be preferred.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double), Integer (default, long). **Calls:** None. **Date:** September 2007. **Remark:** Supersedes HSL_KB12. **Origin:** N.I.M. Gould, Rutherford Appleton Laboratory, and Ph. L. Toint, University of Namur, Belgium. **Language:** Fortran 95.

LA - LINEAR PROGRAMMING

LA Linear programming, i.e. minimization of a linear function subject to linear constraints

LA04 Sparse linear programming: steepest-edge simplex method

This package uses the simplex method to solve the linear programming problem

$$\text{minimize } \mathbf{c}^T \mathbf{x} = \sum_{j=1}^n c_j x_j$$

subject to the constraints

$$\mathbf{Ax} = \mathbf{b}, \text{ that is, } \sum_{j=1}^n a_{ij} x_j = b_i, \quad i = 1, 2, \dots, m$$

$$l_j \leq x_j \leq u_j, \quad 1 \leq j \leq k, \quad (1.1)$$

$$x_j \geq 0, \quad l \leq j \leq n.$$

The variables x_j , $k + 1 \leq j \leq l - 1$, if any, are free (have no bounds). Full advantage is taken of any zero coefficients a_{ij} . The inequalities $0 \leq k < l \leq n + 1$ must hold. Special values $l_i = -\sigma$ and $u_i = \sigma$ may be used in (1.1) to remove one or both bounds.

To accommodate roundoff, all variables are permitted to lie slightly outside their bounds.

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double). **Precision:** At least 8-byte arithmetic is recommended. **Calls:** FA14, LA15, MC29. **Helpful:** MC59 (sorting). **Language:** Fortran 77. **Date:** March 1999. **Origin:** J.K. Reid, Rutherford Appleton Laboratory.

LA15 Sparse mathematical programming bases: factorize and update

To factorize a matrix, solve corresponding systems of linear equations and update the factorization when a column of the matrix is altered, exploiting sparsity in all cases. Its primary application is likely to be for handling **linear programming** bases.

ATTRIBUTES — **Version:** 1.2.0. **Types:** Real (single, double). **Remark:** Supersedes LA03A. **Calls:** MC59. **Language:** Fortran 77. **Date:** 1975, revised May 2001. **Remark:** LA15 is a threadsafe version of LA05. **Origin:** J.K. Reid, Harwell.

M - LINEAR ALGEBRA

MA Solution of linear systems

MA38 Sparse unsymmetric system: unsymmetric multifrontal method

This package solves a sparse unsymmetric system of n linear equations in n unknowns using an unsymmetric multifrontal variant of Gaussian elimination. There are facilities for choosing a good pivot order, factorizing another matrix with a nonzero pattern identical to that of a previously factorized matrix, and solving a system of equations using the factorized matrix. An option exists for solving triangular systems using the factors from the Gaussian elimination.

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double). **Calls:** MC13, MC21, _SWAP, _GEMV, I_AMAX, _GEMM, _TRSV, _TRSM. **Language:** Fortran 77. **Date:** March 1995. **Origin:** T.A. Davis, University of Florida, and I.S. Duff, Rutherford Appleton Laboratory.

MA41 Sparse unsymmetric system: unsymmetric multifrontal method

To solve a sparse unsymmetric system of linear equations. Given an unsymmetric square sparse matrix \mathbf{A} of order n and an n -vector \mathbf{B} , this subroutine solves the system $\mathbf{Ax} = \mathbf{b}$ or $\mathbf{A}^T\mathbf{x} = \mathbf{b}$. The method used is a parallel direct method based on a sparse multifrontal variant of Gaussian elimination. An initial ordering for the pivotal sequence is chosen using the pattern of the matrix $\mathbf{A} + \mathbf{A}^T$ and is later modified for reasons of numerical stability. Thus this code performs best on matrices whose pattern is symmetric, or nearly so. For symmetric sparse matrices or for very unsymmetric and very sparse matrices, other software might be more appropriate (for example, MA57 or MA48).

ATTRIBUTES — **Version:** 1.2.0. **Types:** Real (single, double). **Calls:** MC21, MC29, MC47, I_AMAX, _GEMV, _TRSV. **Language:** Fortran 77 **Date:** September 1995, revised August 2007. **Origin:** P.R. Amestoy, ENSEEIHT, Toulouse, France and I.S. Duff, Rutherford Appleton Laboratory.

MA42 Sparse unsymmetric system: out-of-core frontal method

To solve one or more sets of sparse linear equations, $\mathbf{Ax} = \mathbf{b}$ or $\mathbf{A}^T\mathbf{x} = \mathbf{b}$, by the frontal method, optionally using direct-access files for the matrix factors. Use is made of high level BLAS kernels. The code has low in-core memory requirements. The matrix \mathbf{A} may be input by the user in either of the following ways:

- (i) by elements in a finite-element calculation,
- (ii) by equations (matrix rows).

In both cases, the coefficient matrix and right-hand side(s) are of the form

$$\mathbf{A} = \sum_{k=1}^m \mathbf{A}^{(k)}, \quad \mathbf{b} = \sum_{k=1}^m \mathbf{b}^{(k)}.$$

In case (i), the summation is over finite elements. $\mathbf{A}^{(k)}$ is nonzero only in those rows and columns which correspond to variables in the k -th element. $\mathbf{b}^{(k)}$ is nonzero only in those rows which correspond to variables in element k .

In case (ii), the summation is over equations and $\mathbf{A}^{(k)}$ and $\mathbf{b}^{(k)}$ are nonzero only in row k .

ATTRIBUTES — **Version:** 1.2.0. **Types:** Real (single, double). **Remark:** MA42 supersedes MA32. **Calls:** I_AMAX, _AXPY, _GER, _GEMV, _TPSV, _GEMM, _TRSM. **Helpful:** MC62, MC63. **Language:** Fortran 77. **Date:** December 1992. **Origin:** I.S. Duff and J.A. Scott, Rutherford Appleton Laboratory.

HSL_MA42 Sparse unsymmetric system: out-of-core frontal method

The module HSL_MA42 solves one or more sets of sparse linear equations, $\mathbf{Ax} = \mathbf{b}$ or $\mathbf{A}^T \mathbf{x} = \mathbf{b}$, by the frontal method, optionally using direct-access files for the matrix factors. Use is made of high level BLAS kernels. The code has low in-core memory requirements. The matrix \mathbf{A} may be input by the user in either of the following ways:

- (i) by elements in a finite-element calculation,
- (ii) by equations (matrix rows).

In both cases, the coefficient matrix and right-hand side(s) are of the form

$$\mathbf{A} = \sum_{k=1}^m \mathbf{A}^{(k)}, \quad \mathbf{b} = \sum_{k=1}^m \mathbf{b}^{(k)}.$$

In case (i), the summation is over finite elements. The matrix $\mathbf{A}^{(k)}$ is nonzero only in those rows and columns which correspond to variables in the k -th element. The vector $\mathbf{b}^{(k)}$ is nonzero only in those rows which correspond to variables in element k .

In case (ii), the summation is over equations and $\mathbf{A}^{(k)}$ and $\mathbf{b}^{(k)}$ are nonzero only in row k .

ATTRIBUTES — **Version:** 1.2.0. **Types:** Real (single, double). **Remark:** HSL_MA42 is a Fortran 95 version of MA42. **Calls:** I_AMAX, _AXPY, _GER, _GEMV, _TPSV, _GEMM, _TRSM. **Helpful:** MC62, MC63. **Language:** Fortran 95. **Date:** April 1994. **Origin:** I.S. Duff and J.A. Scott, Rutherford Appleton Laboratory.

HSL_MA42_ELEMENT Unsymmetric finite-element system: out-of-core frontal method (real and complex)

HSL_MA42_ELEMENT solves one or more sets of sparse linear unsymmetric unassembled finite-element equations, $\mathbf{AX} = \mathbf{B}$, or $\mathbf{A}^T \mathbf{X} = \mathbf{B}$, or $\mathbf{A}^H \mathbf{X} = \mathbf{B}$, by the frontal method. The system may be real or complex (\mathbf{A}^H denotes the conjugate transpose of \mathbf{A}). There are options for automatically ordering the elements, for supplying the elements using a reverse communication interface, for holding the matrix factors in direct-access files, and for preserving a partial factorization.

The $n \times n$ coefficient matrix \mathbf{A} must have a symmetric structure and must be in elemental form

$$\mathbf{A} = \sum_{k=1}^{nelt} \mathbf{A}^{[k]},$$

where $\mathbf{A}^{[k]}$ is nonzero only in those rows and columns that correspond to variables in the k -th finite element. The elements must be square elements, with the row indices equal to the column indices. For each k , the user must supply a list specifying which rows/columns of \mathbf{A} are associated with $\mathbf{A}^{[k]}$ and an array containing the nonzero entries. The right-hand sides \mathbf{B} may be supplied in elemental form (that is, $\mathbf{B} = \sum_{k=1}^{nelt} \mathbf{B}^{[k]}$) or in assembled form.

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double), Complex (single, double). **Precision:** At least 8-byte arithmetic is recommended. **Remark:** For assembled systems of equations, HSL_MA42 may be used. **Calls:** MC63, HSL_MC73, I_AMAX, _AXPY, _GER, _GERU, _GEMV, _TPSV, _GEMM, _TRSM. **Language:** Fortran 95 + TR 15581 (allocatable components). **Date:** June 2004. **Origin:** I.S. Duff and J.A. Scott, Rutherford Appleton Laboratory.

MA43 Sparse unsymmetric system: row-by-row frontal method

To solve one or more sets of variable-band linear equations, $\mathbf{Ax} = \mathbf{b}$ or $\mathbf{A}^T\mathbf{x} = \mathbf{b}$ by the frontal method.

MA43 provides the user with a straightforward interface to the HSL routine MA42 when entry is by equations and auxiliary storage is not required. If the user requires more sophisticated facilities, MA42 should be employed.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** MA42, MC59. **Helpful:** MC62. **Language:** Fortran 77. **Date:** March 1993, revised July 2001. **Origin:** J.A. Scott, Rutherford Appleton Laboratory.

MA44 Over-determined linear system: least-squares solution

To solve an equality-constrained linear least squares problem. Given an over-determined system of m linear equations in n unknowns,

$$\sum_{j=1}^n a_{ij}x_j = b_i, \quad i = 1, 2, \dots, m, \quad m \geq n \geq 1,$$

it calculates the solution vector \mathbf{x} which satisfies the first m_1 equations ($0 \leq m_1 \leq n$) and minimizes the sum of squares of residuals

$$S(\mathbf{x}) = \sum_{i=1}^m r_i^2,$$

where

$$r_i = \sum_{j=1}^n a_{ij}x_j - b_i, \quad i = 1, 2, \dots, m.$$

The matrix $\mathbf{A} = a_{ij}$ must have rank n , that is its columns must be linearly independent. Also the first m_1 rows of \mathbf{A} must be linearly independent.

There is a re-entry facility which allows further systems having the same left-hand sides to be solved economically.

There is an entry to obtain solution standard deviations and the variance-covariance matrix. These are calculated on the assumption that the first m_1 equations are exact and that the remaining equations have errors which are independent random variables with the same variance.

The automatic printing of results and the calculation of equation residuals are options.

The subroutine can be used to solve the general linear least squares data fitting problem with or without equality side conditions.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** _DOT. **Language:** Fortran 77. **Date:** January 1985. **Origin:** J.K. Reid, Harwell.

MA46 Sparse unsymmetric finite-element system: multifrontal

To solve one or more set of sparse unsymmetric linear equations $\mathbf{AX} = \mathbf{B}$ from finite-element applications, using a multifrontal elimination scheme. The matrix \mathbf{A} must be input by elements and be of the form

$$\mathbf{A} = \sum_{k=1}^m \mathbf{A}^{(k)}$$

where $\mathbf{A}^{(k)}$ is nonzero only in those rows and columns that correspond to variables of the nodes of the k -th element. Optionally, the user may pass an additional matrix \mathbf{A}_d of coefficients for the diagonal. \mathbf{A} is then of the form

$$\mathbf{A} = \sum_{k=1}^m \mathbf{A}^{(k)} + \mathbf{A}_d$$

The right-hand side \mathbf{B} should be assembled through the summation

$$\mathbf{B} = \sum_{k=1}^m \mathbf{B}^{(k)},$$

before calling the solution routine.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** `_GEMM`, `_GEMV`, `_GER`, `I_AMAX`, `_SCAL`, `_SWAP`, `_TRSM`, `_TRSV`. **Language:** Fortran 77. **Date:** September 1995. **Origin:** A.C. Damhaug, Det Norske Veritas Research AS and J.K. Reid, Rutherford Appleton Laboratory.

MA48 Sparse unsymmetric system: driver for conventional direct method

To solve a sparse unsymmetric system of m linear equations in n unknowns using Gaussian elimination. There are facilities for block triangularization, choosing a good pivot order, factorizing a matrix with a given pivot order, and solving a system of equations using the factorized matrix. Error estimates are available.

ATTRIBUTES — **Version:** 2.1.0. **Types:** Real (single, double). **Remark:** Supersedes MA28. **Calls:** `MA50`, `MC13`, `MC21`, `MC71`. **Language:** Fortran 77. **Date:** April 1993, revised August 2001. **Origin:** I.S. Duff and J.K. Reid, Rutherford Appleton Laboratory.

HSL_MA48 Sparse unsymmetric system: driver for conventional direct method

To solve a sparse unsymmetric system of linear equations. Given a sparse matrix $\mathbf{A} = \{a_{ij}\}_{m \times n}$ and a vector \mathbf{b} , this subroutine solves the system $\mathbf{Ax} = \mathbf{b}$ or the system $\mathbf{A}^T \mathbf{x} = \mathbf{b}$. The matrix \mathbf{A} can be rectangular. There is an option for iterative refinement and return of error estimates.

This Fortran 95 code offers additional features to the Fortran 77 code `HSL_MA48`. For example, there is an option to analyse the matrix and generate the factors with a single call. The storage required for the factorization is chosen automatically and, if there is insufficient space for the factorization, more space is allocated and the factorization is restarted. The Fortran 95 version also returns the number of entries in the factors and has facilities for computing the determinant when the matrix is square and for identifying the rows and columns that are treated specially when the matrix is singular or rectangular.

ATTRIBUTES — **Version:** 2.1.1. **Types:** Real (single, double). **Remark:** `HSL_MA48` is a Fortran 95 encapsulation of `MA48` and offers some additional facilities to the Fortran 77 version. **Calls:** `MA48`, `MA50`, `MC13`, `MC21`, `MC71`, `_GEMM`, `_TPSV`, `HSL_ZD11`. **Language:** Fortran 95 + TR 15581 (allocatable components). **Date:** November 2001, revised December 2006. **Origin:** I.S. Duff and J.K. Reid, Rutherford Appleton Laboratory.

MA49 Sparse over-determined system: least squares by QR

To compute an orthogonal factorization of a sparse overdetermined matrix \mathbf{A} and optionally to solve the least squares problem $\min \|\mathbf{b} - \mathbf{Ax}\|_2$. Given a sparse matrix \mathbf{A} of order $m \times n$, $m \geq n$, of full column rank, this subroutine computes the **QR** factorization $\mathbf{A} = \mathbf{Q} \begin{bmatrix} \mathbf{R} \\ \mathbf{0} \end{bmatrix}$ where \mathbf{Q} is an $m \times m$ orthogonal matrix and \mathbf{R} is an $n \times n$ upper triangular matrix.

Given an n -vector \mathbf{b} , this subroutine may also compute the minimum 2-norm solution of the linear system $\mathbf{A}^T \mathbf{x} = \mathbf{b}$, by solving $[\mathbf{R}^T \mathbf{0}] \mathbf{z} = \mathbf{b}$ and performing the multiplication $\mathbf{x} = \mathbf{Qz}$, or, if the \mathbf{Q} factor is not stored, by solving $\mathbf{R}^T \mathbf{Rz} = \mathbf{b}$ and performing the multiplication $\mathbf{x} = \mathbf{Az}$.

The subroutine can also solve systems with the coefficient matrix $\begin{bmatrix} \mathbf{R} \\ \mathbf{0} \end{bmatrix}$ or $[\mathbf{R}^T \mathbf{0}]$, or will compute the product of \mathbf{Q} or \mathbf{Q}^T with a vector.

The method used is based on the multifrontal approach and makes use of Householder transformations. Because an ordering for the columns is chosen using the pattern of the matrix $\mathbf{A}^T \mathbf{A}$, this code is not designed for matrices with full rows.

ATTRIBUTES — **Version:** 1.2.0. **Types:** Real (single, double). **Calls:** MC71, and MC47; BLAS: I_AMAX, _AXPY, _COPY, _DOT, _GEMM, _GER, _GEMV, _NRM2, _SYRK, _SWAP, _TRSM, _TRSV; and GENBTF (Alex Pothén, Old Dominion University, North Carolina). **Language:** Fortran 77 **Date:** December 1999, revised August 2001. **Origin:** P.R. Amestoy, ENSEEIHT-IRIT, Toulouse, France; I.S. Duff, Rutherford Appleton Laboratory; and C. Puglisi, CERFACS, Toulouse, France.

MA50 Sparse unsymmetric system: conventional direct method

These subroutines are for the solution of a general sparse $m \times n$ system of linear equations (the most usual case being square, $m = n$), stored by columns. No block triangularization, iterative refinement, or error estimation is included.

If the user requires a more convenient data interface, the MA48 package should be used. The MA48 subroutines call the MA50 subroutines after checking and sorting the user's input data and optionally using MC21 and MC13 to permute the matrix to block triangular form.

ATTRIBUTES — **Version:** 2.0.0. **Types:** Real (single, double). **Remark:** This supersedes MA30 and is normally called through the MA48 package. **Calls:** _AXPY, _DOT, _GEMM, _GEMV, I_MAX, _SCAL, _SWAP, _TRSM, _TRSV. **Language:** Fortran 77. **Date:** April 1993, revised August 2001. **Origin:** I.S. Duff and J.K. Reid, Rutherford Appleton Laboratory.

MA51 Auxiliary for MA48 and MA50: identify ignored rows or columns in the rectangular or rank-deficient case, compute determinant

This is for use in conjunction with the MA48 and MA50 packages for solving sparse unsymmetric sets of linear equations. If the matrix is singular or rectangular, it identifies the rows and columns that are treated specially. If the matrix is square, it computes the determinant.

It identifies which equations are ignored when solving $\mathbf{Ax} = \mathbf{b}$ and which solution components are always set to zero. The roles are reversed for $\mathbf{A}^T \mathbf{x} = \mathbf{b}$. There are such equations and/or components in the singular or rectangular case. Note that if $\mathbf{Ax} = \mathbf{b}$ or $\mathbf{A}^T \mathbf{x} = \mathbf{b}$ is not consistent, there may be large residuals for the equations that are ignored.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** None. **Language:** Fortran 77. **Date:** May 1994, revised July 2004. **Origin:** J.K. Reid, Rutherford Appleton Laboratory.

MA52 Sparse unsymmetric finite-element system: out-of-core multiple front method

This collection of subroutines, when used in conjunction with the MA42 package, solves finite-element equations using a multiple front algorithm. It is assumed that the underlying finite-element mesh has been partitioned into (non-overlapping) subdomains. In the multiple front algorithm, a frontal method is applied to each subdomain separately. This can be done in parallel. Using multiple fronts can also reduce the amount of work required.

MA52 provides routines for generating lists of variables belonging to more than one subdomain, for preserving the partial factorization of a matrix when the sequence of calls to MA42B/BD (using element or equation entry) is incomplete, and for performing forward or back-substitution on a subdomain.

MA52 uses reverse communication.

The use of HSL routine MC53 to obtain an efficient element ordering in each subdomain is recommended before MA52 is used.

MA52 was revised in October 1999 and in August 2001. This involved adding extra subroutines MA52F/FD and MA52K/KD, which allow MA52 to be used without the restriction to diagonal pivoting required by MA52B/BD and also allows MA52 to be used to solve general linear systems of equations (not in finite element form) using the multiple front method. In this case, the user must partition the system matrix into blocks of rows and the frontal method should then be applied to each block separately. At the end of the assembly and elimination processes, for each block there will remain a rectangular frontal matrix and corresponding right-hand side vector. These may be preserved using subroutine MA52F/FD or, in sparse form, using MA52K/KD.

ATTRIBUTES — **Version:** 1.0.1. **Types:** Real (single, double). **Calls:** MA42. **Helpful:** MC53. **Language:** Fortran 77. **Date:** March 1994, revised October 1999, August 2001. **Origin:** J.A. Scott, Rutherford Appleton Laboratory.

HSL_MA54 Definite symmetric full matrix: partial or complete factorization and solution

For a matrix that is full, symmetric and positive definite, this module performs partial factorizations and solutions of corresponding sets of equations, paying special attention to the efficient use of cache memory. It uses a modification of the code of Andersen, Gunnels, Gustavson, Reid, and Wasniewski (ACM Trans. on Math. Software, **31**, 2005, 201-227). It is designed as a kernel for use in a frontal or multifrontal solver, but may also be used for the direct solution of a full set of equations.

The modification involves limiting the eliminations to the leading p columns. The factorization takes the form

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{21}^T \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{pmatrix} = \begin{pmatrix} \mathbf{L}_{11} & \\ & \mathbf{L}_{21} \quad \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{I} & \\ & \mathbf{S}_{22} \end{pmatrix} \begin{pmatrix} \mathbf{L}_{11}^T & \mathbf{L}_{21}^T \\ & \mathbf{I} \end{pmatrix}$$

where \mathbf{L}_{11} is lower triangular and both \mathbf{A}_{11} and \mathbf{L}_{11} have order p .

The input format for \mathbf{A} is that its lower triangular part is held in lower packed format (that is, packed by columns). This format is also used for \mathbf{S}_{22} on return. The matrices \mathbf{L}_{11} and \mathbf{L}_{21} are held as block matrices with blocks of size at most $nb \times nb$. Internally, we use this blocked format also for \mathbf{A}_{22} .

The module has facilities for rearranging a lower triangular matrix in lower packed format (that is, packed by columns) to this blocked format and vice-versa.

Subroutines are provided for partial forward and back substitution, that is, solving equations of the form

$$\begin{pmatrix} \mathbf{L}_{11} & \\ & \mathbf{L}_{21} \quad \mathbf{I} \end{pmatrix} \mathbf{X} = \mathbf{B} \quad \text{and} \quad \begin{pmatrix} \mathbf{L}_{11}^T & \mathbf{L}_{21}^T \\ & \mathbf{I} \end{pmatrix} \mathbf{X} = \mathbf{B}$$

and the corresponding equations for a single right-hand side \mathbf{B} and solution \mathbf{x} .

There are also subroutines for solving one or more sets of equations after a full factorization ($p = n$) and for extracting the diagonal of \mathbf{L}_{11} .

ATTRIBUTES — **Version:** 1.4.0. **Types:** Real (single, double). **Remark:** The code has been tuned only for 8-byte arithmetic. **Language:** Fortran 95. **Calls:** _COPY, _DOT, _GEMM, _GEMV, _SYRK, _TPSV, _TRSM. **Date:** July 2007. **Origin:** J.K. Reid and J.A. Scott, Rutherford Appleton Laboratory.

HSL_MA55 Band symmetric positive-definite system

This module solves a system of linear equations whose matrix is banded, symmetric and positive definite. It uses block Cholesky factorization, taking advantage of any variation in bandwidth. It has an option for storing the matrix itself as well as its factorization. A secondary entry provides for further systems with the same matrix but different right-hand sides to be solved economically. The secondary entry can also be used to calculate residuals, needed for iterative refinement, provided the matrix itself has been stored. For very large systems or if restart facilities are desired, HSL_MA55 uses a direct-access file for the factors and a sequential file for the matrix.

The user must first specify all the row lengths. The matrix itself is supplied by blocks of rows using reverse communication. The blocking for input is chosen by the user and is independent of the blocking used for solution.

MC60 may be used to get a good ordering.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** _GEMM, _SYRK, _TRSM, _TRSV. **Helpful:** MC60. **Language:** Fortran 95. **Date:** July 1999. **Origin:** J.K. Reid, Rutherford Appleton Laboratory.

MA57 Sparse symmetric system: multifrontal method

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 \mathbf{b} (or an $n \times s$ matrix \mathbf{B}), this subroutine solves the system $\mathbf{Ax} = \mathbf{b}$ ($\mathbf{AX} = \mathbf{B}$). The matrix \mathbf{A} need not be definite.

The multifrontal method is used. It is a direct method based on a sparse variant of Gaussian elimination.

The matrix is optionally prescaled by using a symmetrization of the MC64 scaling. Other ordering options are provided including hooks to MeTiS. The user can avoid additional fill-in to that predicted by the analysis by using static pivoting.

ATTRIBUTES — **Version:** 3.6.0. **Types:** Real (single, double). **Remark:** Supersedes MA27. **Calls:** MC71, MC47, MC64, BLAS routines _GEMM, _TPSV, I_AMAX, and (optionally) MeTiS. **Language:** Fortran 77. **Date:** September 2000, revised August 2001, July 2004. **Origin:** I.S. Duff, Rutherford Appleton Laboratory.

HSL_MA57 Sparse symmetric system: multifrontal method

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 \mathbf{b} or a matrix $\mathbf{B} = \{b_{ij}\}_{n \times r}$, this subroutine solves the system $\mathbf{Ax} = \mathbf{b}$ or the system $\mathbf{AX} = \mathbf{B}$. The matrix \mathbf{A} need not be definite. There is an option for iterative refinement.

The method used is a direct method based on a sparse variant of Gaussian elimination.

The matrix is optionally prescaled by using a symmetrization of the MC64 scaling. Other ordering options are provided including hooks to MeTiS. The user can avoid additional fill-in to that predicted by the analysis by using static pivoting.

ATTRIBUTES — **Version:** 4.2.1. **Types:** Real (single, double). **Remark:** HSL_MA57 is a Fortran 95 version of MA57. **Calls:** MA57, HSL_ZD11. **Language:** Fortran 95 + TR 15581 (allocatable components). **Date:** September 2000, revised August 2001, July 2004. **Origin:** I.S. Duff and J.K. Reid, Rutherford Appleton Laboratory.

MA60 Iterative refinement and error estimation

This subroutine performs iterative refinement and provides information on the error in the resulting solution of a set of n sparse linear equations $\mathbf{Ax} = \mathbf{b}$, either as an upper bound on the actual error (**forward error**) or as a bound on the perturbation to the matrix elements which would make the solution obtained the exact solution of a perturbed problem with the same sparsity pattern as the original matrix (**backward error**). All estimates are provided in the ∞ -norm, that is $\|\mathbf{x}\|_{\infty} = \max_i |x_i|$.

There is an option for avoiding iterative refinement, in which case bounds on the errors associated with the given solution are provided.

ATTRIBUTES — **Version:** 1.2.0. **Types:** Real (single, double). **Calls:** MC71, L_MAX. **Language:** Fortran 77. **Date:** 1988, revised July 2001. **Remark:** MA60 is a threadsafe version of MA40. **Origin:** M. Arioli, I.S. Duff, Harwell.

MA61 Sparse symmetric positive-definite system: incomplete factorization

To solve a symmetric, sparse and positive definite set of linear equations $\mathbf{Ax} = \mathbf{b}$ i.e.

$$\sum_{j=1}^n a_{ij}x_j = b_i, \quad i = 1, 2, \dots, n.$$

The solution is found by a preconditioned conjugate gradient technique, where the preconditioning is done by incomplete factorization.

- (a) MA61A performs the incomplete factorization based on an \mathbf{LDL}^T decomposition. New entries which have small numerical values compared to the corresponding diagonal entries are dropped, and the diagonal entries are modified to ensure positive definiteness. This results in a preconditioning matrix \mathbf{C} held in \mathbf{LDL}^T form.
- (b) MA61B performs the iteration procedure using the preconditioned coefficient matrix (i.e. \mathbf{AC}^{-1}) as the iteration matrix for the conjugate gradient algorithm.

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double). **Calls:** None. **Language:** Fortran 77. **Date:** 1979, revised June 2001. **Remark:** MA61 is a threadsafe version of MA31. **Origin:** N. Munksgaard, Danish Natural Science Research Council, Grant. No. 511-10069.

MA62 Sparse symmetric finite-element system: out-of-core frontal method

To solve one or more sets of sparse symmetric linear unassembled finite-element equations, $\mathbf{AX} = \mathbf{B}$, by the frontal method, optionally holding the matrix factor out-of-core in direct-access files. The package is primarily designed for positive-definite matrices since numerical pivoting is not performed. Use is made of high-level BLAS kernels. The coefficient matrix \mathbf{A} must of the form

$$\mathbf{A} = \sum_{k=1}^m \mathbf{A}^{(k)}$$

with $\mathbf{A}^{(k)}$ nonzero only in those rows and columns that correspond to variables in the k -th element.

The frontal method is a variant of Gaussian elimination and involves the factorization

$$\mathbf{A} = \mathbf{PLD}(\mathbf{PL})^T,$$

where \mathbf{P} is a permutation matrix, \mathbf{D} is a diagonal matrix, and \mathbf{L} is a unit lower triangular matrix. The solution process is completed by performing the forward elimination

$$(\mathbf{PL})\mathbf{DY} = \mathbf{B},$$

followed by the back substitution

$$(\mathbf{PL})^T\mathbf{X} = \mathbf{Y}.$$

MA62 stores the reals of the factors and their indices separately. A principal feature of MA62 is that, by holding the factors out-of-core, large problems can be solved using a predetermined

and relatively small amount of in-core memory. At an intermediate stage of the solution, l say, the ‘front’ contains those variables associated with one or more of $\mathbf{A}^{(k)}$, $k = 1, 2, \dots, l$, which are also present in one or more of $\mathbf{A}^{(k)}$, $k = l + 1, \dots, m$. For efficiency, the user should order the $\mathbf{A}^{(k)}$ so that the number of variables in the front (the ‘front size’) is small. For example, a very rectangular grid should be ordered pagewise parallel to the short side of the rectangle. The elements may be preordered using the HSL routine MC63.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** `_AXPY`, `_GER`, `_GEMV`, `_TPSV`, `_TRSV`, `_GEMM`, `_TRSM`. **Helpful:** MC63. **Language:** Fortran 77. **Date:** April 1997. **Origin:** I.S. Duff and J.A. Scott, Rutherford Appleton Laboratory.

HSL_MA64 Indefinite symmetric full matrix: partial or complete factorization and solution

For a full matrix that is real symmetric, complex Hermitian, or complex symmetric, this module performs partial or full factorizations and performs solutions of corresponding sets of equations, paying special attention to the efficient use of cache memory. In the real and complex Hermitian cases, the matrix need not be positive definite. The module performs symmetric interchanges for stability and uses both 1×1 and 2×2 pivots, assuming that the matrix is **well scaled** in the sense that changes that are small compared to the largest entry can be tolerated. It is suitable for use in a frontal or multifrontal solver, but may also be used for the direct solution of a full set of equations. Optionally, it may be compiled to use OpenMP.

Eliminations are limited to the leading p rows and columns. Stability considerations may lead to $q \leq p$ eliminations being performed, but there is an option to force all p eliminations to be performed. Using an asterisk to represent taking the transpose or Hermitian transpose of a matrix, the factorization takes the form

$$\mathbf{PAP}^* = \begin{pmatrix} \mathbf{L}_{11} & \\ & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{D} & \\ & \mathbf{S}_{22} \end{pmatrix} \begin{pmatrix} \mathbf{L}_{11}^* & \mathbf{L}_{21}^* \\ & \mathbf{I} \end{pmatrix}$$

where \mathbf{A} has order n , \mathbf{P} is a permutation matrix, \mathbf{L}_{11} is a unit lower triangular matrix of order q , \mathbf{D} is block diagonal of order q , and \mathbf{S}_{22} is a matrix of order $n - q$. The permutation matrix \mathbf{P} has the form

$$\mathbf{P} = \begin{pmatrix} \mathbf{P}_1 & \\ & \mathbf{I} \end{pmatrix}$$

where \mathbf{P}_1 is of order p . Each diagonal block of \mathbf{D} has size one or two.

The input format for \mathbf{A} is that its lower triangular part is held in lower packed format (that is, packed by columns). This format is also used for \mathbf{S}_{22} on return. The matrix $\begin{pmatrix} \mathbf{L}_{11} \\ \mathbf{L}_{21} \end{pmatrix}$ is returned as a sequence of block columns, each of which consists of a triangular matrix packed by columns followed by a rectangular matrix packed by columns.

Subroutines are provided for partial solutions, that is, solving equations of the form

$$\begin{pmatrix} \mathbf{L}_{11} & \\ \mathbf{L}_{21} & \mathbf{I} \end{pmatrix} \mathbf{X} = \mathbf{B}, \quad \begin{pmatrix} \mathbf{D} & \\ & \mathbf{I} \end{pmatrix} \mathbf{X} = \mathbf{B},$$

$$\begin{pmatrix} \mathbf{D} & \\ & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{L}_{11}^* & \mathbf{L}_{21}^* \\ & \mathbf{I} \end{pmatrix} \mathbf{X} = \mathbf{B}, \quad \text{and} \quad \begin{pmatrix} \mathbf{L}_{11}^* & \mathbf{L}_{21}^* \\ & \mathbf{I} \end{pmatrix} \mathbf{X} = \mathbf{B}$$

and the corresponding equations for a single right-hand side b and solution x .

ATTRIBUTES — **Version:** 6.0.0. **Types:** Real (single, double), Complex (single, double). **Language:** Fortran 95. **Remark:** HSL_MA54 should be used if A is known to be positive definite. **Calls:** `_COPY`, `_GEMM`, `_GEMV`, `_SYRK`, `_TPSV`, `_TRSM`. **Parallelism:** May use OpenMP. **Date:** June 2007, revised January 2011. **Origin:** J.K. Reid and J.A. Scott, Rutherford Appleton Laboratory.

MA65 Unsymmetric banded system of linear equations

To solve an unsymmetric banded system of linear equations. Given a unsymmetric band matrix $\mathbf{A} = \{a_{ij}\}_{n \times n}$ and an n -vector \mathbf{b} , this subroutine solves the system $\mathbf{Ax} = \mathbf{b}$.

The matrix is factorized using Gaussian elimination with row interchanges. If the lower semibandwidth is kl and the upper semibandwidth is ku , that is, if $a_{ij} = 0$ for $i > j + kl$ or $j > i + ku$, fill-in is limited to kl additional diagonals of the upper triangle and the computation is performed within an array of size $n(2kl + ku + 1)$. At each pivotal step, operations are avoided on any row with a zero in the pivot column and on the columns beyond the last to have an entry in the pivot row.

ATTRIBUTES — **Version:** 1.0.1. **Types:** Real (single, double). **Calls:** None. **Origin:** J.K. Reid, Rutherford Appleton Laboratory. **Date:** January 2001.

MA67 Sparse symmetric system, zeros on diagonal: blocked conventional

To solve a sparse symmetric indefinite system of linear equations. Given a sparse symmetric matrix $\mathbf{A} = \{a_{ij}\}_{n \times n}$ and an n -vector \mathbf{b} , this subroutine solves the system $\mathbf{Ax} = \mathbf{b}$.

The method used is a direct method using an \mathbf{LDL}^T factorization, where \mathbf{L} is unit lower triangular and \mathbf{D} is block diagonal with blocks of order 1 and 2. Advantage is taken of the extra sparsity available with 2×2 pivots (blocks of \mathbf{D}) with one or both diagonal entries of value zero. The numerical values of the entries are taken in account during the first choice of pivots.

ATTRIBUTES — **Version:** 1.0.2. **Types:** Real (single, double). **Calls:** `_GEMM`, `_TPSV`, `_GEMV`, `_TRSV`, `_TPMV`, `I_AMAX`. **Language:** Fortran 77. **Date:** September 2000. **Origin:** J.K. Reid, Rutherford Appleton Laboratory.

MA69 Unsymmetric system whose leading subsystem is easy to solve

This set of subroutines compute the solution to an extended system of $n + m$ real linear equations in $n + m$ unknowns,

$$\begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{pmatrix}$$

in the case where the n by n matrix \mathbf{A} is nonsingular and solutions to the systems

$$\mathbf{Ax} = \mathbf{b} \text{ and } \mathbf{A}^T \mathbf{y} = \mathbf{c}$$

may be obtained from an external source, such as an existing factorization. The subroutine uses reverse communication to obtain the solution to such smaller systems. The method makes use of the Schur complement matrix

$$\mathbf{S} = \mathbf{D} - \mathbf{CA}^{-1}\mathbf{B}.$$

The Schur complement is stored and factorized as a dense matrix and the subroutine is thus only appropriate if there is sufficient storage for this matrix. Special advantage is taken of symmetry and definiteness in the coefficient matrices. Provision is made for introducing additional rows and columns to, and removing existing rows and columns from, the extended matrix.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** `_AXPY`, `_DOT`, `_COPY`, `_ROTG`, `_ROT`. **Language:** Fortran 77. **Date:** 1989, revised March, 2001. **Remark:** MA69 is a threadsafe version of MA39. **Origin:** N.I.M. Gould, Harwell.

HSL_MA69 Unsymmetric system whose leading subsystem is easy to solve

HSL_MA69 is a suite of Fortran 95 procedures for computing the the solution to an extended system of $n + m$ sparse real linear equations in $n + m$ unknowns,

$$\begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{pmatrix}$$

in the case where the n by n matrix \mathbf{A} is nonsingular and solutions to the systems

$$\mathbf{Ax} = \mathbf{b} \text{ and } \mathbf{A}^T \mathbf{y} = \mathbf{c}$$

may be obtained from an external source, such as an existing factorization. The subroutine uses reverse communication to obtain the solution to such smaller systems. The method makes use of the Schur complement matrix

$$\mathbf{S} = \mathbf{D} - \mathbf{CA}^{-1}\mathbf{B}.$$

The Schur complement is stored and factorized as a dense matrix and the subroutine is thus appropriate only if there is sufficient storage for this matrix. Special advantage is taken of symmetry and definiteness in the coefficient matrices. Provision is made for introducing additional rows and columns to, and removing existing rows and columns from, the extended matrix.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** _ROT, _ROTG. **Date:** October 2001. **Origin:** N.I.M. Gould, Rutherford Appleton Laboratory, and Ph. L. Toint, University of Namur, Belgium. **Language:** Fortran 95.

MA72 Sparse symmetric finite-element system: out-of-core multiple front method

This collection of subroutines, when used in conjunction with the MA62 package, solves symmetric positive-definite finite-element equations using a multiple front algorithm. It is assumed that the underlying finite-element mesh has been partitioned into (non-overlapping) subdomains. In the multiple front algorithm, a frontal method is applied to each subdomain separately. This may be done in parallel. Using multiple fronts can also have the advantage of requiring less work than applying the frontal method to the whole domain.

MA72 provides routines for generating lists of variables belonging to more than one subdomain, for preserving the partial factorization of a matrix when the sequence of calls to the frontal solver factorization routine MA62B/BD is incomplete, and for performing forward elimination or back-substitution on a subdomain.

MA72 uses reverse communication.

The use of HSL routine MC53 to obtain an efficient element ordering in each subdomain is recommended before MA62 and MA72 are used.

For unsymmetric or symmetric indefinite problems, MA52 should be used in conjunction with MA42.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** MA62. **Helpful:** MC53. **Language:** Fortran 77. **Date:** May 1998. **Origin:** J.A. Scott, Rutherford Appleton Laboratory.

HSL_MA74 Unsymmetric full matrix: partial or complete factorization and solution

Given a dense unsymmetric $n \times n$ matrix \mathbf{A} , HSL_MA74 performs partial factorizations and solutions of corresponding sets of equations. It is designed as a kernel for use in a frontal or multifrontal solver or may be used to factorize and solve a full system of equations.

Eliminations are limited to the leading $p \leq n$ rows and columns. Stability considerations may lead to $q \leq p$ eliminations being performed. The factorization takes the form

$$\mathbf{PAQ} = \begin{pmatrix} \mathbf{L}_1 & \\ & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{D}_1 & \\ & \mathbf{A}_2 \end{pmatrix} \begin{pmatrix} \mathbf{U}_1 & \mathbf{U}_2 \\ & \mathbf{I} \end{pmatrix},$$

where \mathbf{P} and \mathbf{Q} are permutation matrices, \mathbf{L}_1 and \mathbf{U}_1 are unit lower and unit upper triangular matrices of order q , and \mathbf{D}_1 is the diagonal of order q . The permutation matrices \mathbf{P} and \mathbf{Q} are of the form

$$\mathbf{P} = \begin{pmatrix} \mathbf{P}_1 & \\ & \mathbf{I} \end{pmatrix} \text{ and } \mathbf{Q} = \begin{pmatrix} \mathbf{Q}_1 & \\ & \mathbf{I} \end{pmatrix},$$

where \mathbf{P}_1 and \mathbf{Q}_1 are of order p .

Subroutines are provided for partial solutions, that is, solving systems of the form

$$\begin{pmatrix} \mathbf{L}_1 & \\ & \mathbf{I} \end{pmatrix} \mathbf{X} = \mathbf{B}, \quad \begin{pmatrix} \mathbf{D}_1 & \\ & \mathbf{I} \end{pmatrix} \mathbf{X} = \mathbf{B}, \quad \begin{pmatrix} \mathbf{D}_1 & \\ & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{U}_1 & \mathbf{U}_2 \\ & \mathbf{I} \end{pmatrix} \mathbf{X} = \mathbf{B},$$

and

$$\begin{pmatrix} \mathbf{U}_1 & \mathbf{U}_2 \\ & \mathbf{I} \end{pmatrix} \mathbf{X} = \mathbf{B}$$

and the corresponding equations for a single right-hand side \mathbf{b} and solution \mathbf{x} .

Subroutines are also provided for partial solutions to transpose systems, that is, solving systems of the form

$$\begin{pmatrix} \mathbf{U}_1^T & \\ & \mathbf{I} \end{pmatrix} \mathbf{X} = \mathbf{B}, \quad \begin{pmatrix} \mathbf{D}_1 & \\ & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{L}_1^T & \mathbf{L}_2^T \\ & \mathbf{I} \end{pmatrix} \mathbf{X} = \mathbf{B}, \quad \begin{pmatrix} \mathbf{L}_1^T & \mathbf{L}_2^T \\ & \mathbf{I} \end{pmatrix} \mathbf{X} = \mathbf{B}$$

and the corresponding equations for a single right-hand side \mathbf{b} and solution \mathbf{x} .

Options are included for threshold partial pivoting, threshold diagonal pivoting, threshold rook pivoting, and static pivoting.

Note: If a full factorization and solution of one or more sets of equations is required ($p = n$), routines from the LAPACK library may be used (and may be more efficient).

ATTRIBUTES — **Version:** 1.4.0. **Types:** Real (single, double). **Calls:** `_GEMM`, `_GEMV`, `_GER`, `I_AMAX`, `_SWAP`, `_TRSM`, `_TRSV`. **Language:** Fortran 95. **Date:** July 2007. **Origin:** J.K. Reid and J.A. Scott, Rutherford Appleton Laboratory. **Remark:** The development of HSLMA74 was supported by EPSRC grant GR/S42170.

MA75 Sparse over-determined system: weighted least squares

This subroutine solves weighted sparse least-squares problems. Given an $m \times n$ ($m \geq n$) sparse matrix $\mathbf{A} = \{a_{ij}\}$ of rank n , an $m \times m$ diagonal matrix \mathbf{W} of weights, and an m -vector \mathbf{b} , the routine calculates the solution vector \mathbf{x} that minimizes the Euclidean norm of the weighted residual vector $\mathbf{r} = \mathbf{W}(\mathbf{Ax} - \mathbf{b})$ by solving the normal equations $\mathbf{A}^T \mathbf{W}^2 \mathbf{Ax} = \mathbf{A}^T \mathbf{W}^2 \mathbf{b}$.

Three forms of data storage are permitted for the input matrix: storage by columns, where row indices and column pointers describe the matrix; storage by rows, where column indices and row pointers describe the matrix; and the coordinate scheme, where both row and column indices describe the position of entries in the matrix. For the statistical analysis of the weighted least-squares problem, there are two entries: one to obtain a column and one to obtain the diagonal of the covariance matrix $(\mathbf{A}^T \mathbf{W}^2 \mathbf{A})^{-1}$.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** `MA27`, `MC26`, `MC46`, `MC59`. **Language:** Fortran 77. **Date:** July 1990, revised November 2001. **Remark:** MA75 is a threadsafe version of MA45. **Origin:** I.S. Duff and J.A. Scott, Rutherford Appleton Laboratory, P.P.M. de Rijk, University of Amsterdam.

HSL_MA77 Sparse symmetric system: multifrontal out of core

HSL_MA77 solves one or more sets of sparse symmetric equations $\mathbf{AX} = \mathbf{B}$ using an out-of-core multifrontal method. The symmetric matrix \mathbf{A} may be either positive definite or indefinite. It may be input by the user in either of the following ways:

- (i) by square symmetric elements, such as in a finite-element calculation, or
- (ii) by rows.

In both cases, the coefficient matrix is of order n and is of the form

$$\mathbf{A} = \sum_{k=1}^m \mathbf{A}^{(k)}.$$

In (i), the summation is over elements and $\mathbf{A}^{(k)}$ is nonzero only in those rows and columns that correspond to variables in the k th element. In (ii), the summation is over rows and $\mathbf{A}^{(k)}$ is nonzero only in row k . In both cases, for each k , the user must supply a list specifying which columns of \mathbf{A} are associated with $\mathbf{A}^{(k)}$, and an array containing $\mathbf{A}^{(k)}$ in packed form.

The multifrontal method is a variant of sparse Gaussian elimination. In the positive-definite case, it involves the Cholesky factorization

$$\mathbf{A} = (\mathbf{PL})(\mathbf{PL})^T$$

where \mathbf{P} is a permutation matrix and \mathbf{L} is lower triangular. In the indefinite case, it involves the factorization

$$\mathbf{A} = (\mathbf{PL})\mathbf{D}(\mathbf{PL})^T$$

where \mathbf{P} is a permutation matrix, \mathbf{L} is unit lower triangular, and \mathbf{D} is block diagonal with blocks of size 1×1 and 2×2 . The factorization is performed by the subroutine `MA77_factor` and is controlled by an elimination tree that is constructed by the subroutine `MA77_analyse`, which needs the lists of variables in elements or rows and an elimination sequence. Once a matrix has been factorized, any number of calls to the subroutine `MA77_solve` may be made for different right-hand sides \mathbf{B} . An option exists for computing the residuals. For large problems, the matrix data and the computed factors are held in direct-access files.

The efficiency of HSL_MA77 is dependent on the elimination order that the user supplies. The HSL routine HSL_MC68 may be used to obtain a suitable ordering.

ATTRIBUTES — **Version:** 5.6.0. **Types:** Real (single, double). **Calls:** KB07, HSL_KB22, HSL_OF01, HSL_MA54, HSL_MA64. **Date:** September 2007. **Language:** Fortran 95 + TR 15581 (allocatable components), plus allocatable dummy arguments and allocatable components of derived types. **Origin:** J.K. Reid and J.A. Scott, Rutherford Appleton Laboratory. **Remark:** The development of HSL_MA77 was supported by EPSRC grant GR/S42170.

HSL_MA78 Sparse unsymmetric finite-element system: multifrontal out of core

HSL_MA78 solves one or more sets of sparse unsymmetric equations $\mathbf{AX} = \mathbf{B}$ or $\mathbf{A}^T\mathbf{X} = \mathbf{B}$ using an out-of-core multifrontal method. The $n \times n$ matrix \mathbf{A} must be in unassembled element form, that is,

$$\mathbf{A} = \sum_{k=1}^m \mathbf{A}^{(k)}$$

where the summation is over elements and $\mathbf{A}^{(k)}$ is nonzero only in those rows and columns that correspond to variables in the k th element. For each k , the user must supply a list specifying which columns of \mathbf{A} are associated with $\mathbf{A}^{(k)}$, and an array containing $\mathbf{A}^{(k)}$ in packed form. It is permissible for some of the rows and corresponding columns to be empty, that is, to appear in none of the matrices $\mathbf{A}^{(k)}$; such rows and columns are ignored in determining whether the matrix is singular.

The multifrontal method is a variant of sparse Gaussian elimination. It involves the factorization

$$\mathbf{A} = \mathbf{PLDUQ}$$

where \mathbf{P} and \mathbf{Q} are a permutation matrices, \mathbf{L} and \mathbf{U} are unit lower and upper triangular matrices, respectively, and \mathbf{D} is a diagonal matrix. The factorization is performed by the subroutine `MA78_factor` and is controlled by an elimination tree that is constructed by the subroutine `MA78_analyse`, which needs the lists of variables in elements and an elimination sequence. Once a matrix has been factorized, any number of calls to the subroutine `MA78_solve` may be made for different right-hand sides \mathbf{B} . An option exists for computing the residuals. For large problems, the matrix data and the computed factors are held in direct-access files.

The efficiency of `HSL_MA78` is dependent on the elimination order that the user supplies. A suitable ordering may be obtained by first assembling the sparsity pattern of the matrix \mathbf{A} (`MC57` can be used to do this) and then calling the HSL package `HSL_MA68`.

ATTRIBUTES — **Version:** 3.4.0. **Types:** Real (single, double). **Calls:** `KB07`, `HSL_KB22`, `HSL_OF01`, `HSL_MA74`. **Language:** Fortran 95 + TR 15581 (allocatable components), plus allocatable dummy arguments and allocatable components of derived types. **Date:** July 2007. **Origin:** J.K. Reid and J.A. Scott, Rutherford Appleton Laboratory. **Remark:** For symmetric positive definite and symmetric indefinite systems, `HSL_MA77` should be used. **Remark:** The development of `HSL_MA78` was supported by EPSRC grant GR/S42170.

HSL_MA79 Sparse symmetric system: mixed precision

`HSL_MA79` is a mixed precision sparse symmetric solver for solving one or more linear systems $\mathbf{AX} = \mathbf{B}$. A factorization of \mathbf{A} using single precision (that is, 32-bit real arithmetic) is performed using a direct solver (`MA57` or `HSL_MA77`) and then refinement (iterative refinement and, in some cases, `FGMRES`) in double precision (that is, 64-bit real arithmetic) is used to recover higher accuracy. This technique is termed a mixed precision approach. If refinement fails to achieve the requested accuracy, a double precision factorization is performed.

Use of single precision arithmetic substantially reduces the amount of data that is moved around within a sparse direct solver, and on a number of modern architectures, it is currently significantly faster than double precision computation. Thus `HSL_MA79` offers the potential of obtaining a solution to $\mathbf{AX} = \mathbf{B}$ to double-precision accuracy more rapidly than using a direct solver in double precision. `HSL_MA79` is primarily designed for solving very large systems.

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (double). **Calls:** `HSL_MA54`, `MA57`, `HSL_MA64`, `HSL_MA77`, `MC30`, `MC34`, `MC64`, `MC77`, `MI15`, `HSL_OF01`, `HSL_ZD11`. **Date:** June 2008. **Origin:** J. D. Hogg and J. A. Scott, Rutherford Appleton Laboratory. **Language:** Fortran 95, plus allocatable dummy arguments and allocatable components of derived types. **Parallelism:** May use OpenMP through `HSL_MA77`. **Remark:** The development of `HSL_MA79` was supported by EPSRC grants EP/F006535/1 and EP/E053351/1.

HSL_MA86 Sparse symmetric indefinite system using OpenMP

`HSL_MA86` uses a direct method to solve large sparse symmetric indefinite linear systems of equations $\mathbf{AX} = \mathbf{B}$. This package uses OpenMP and is designed for multicore architectures. It computes the sparse factorization

$$\mathbf{A} = \mathbf{PLD(PL)}^*$$

where $\mathbf{L}^* = \mathbf{L}^T$ (real symmetric or complex symmetric) or $\mathbf{L}^* = \mathbf{L}^H$ (complex Hermitian, where \mathbf{L}^H denotes the conjugate transpose of \mathbf{L}), \mathbf{P} is a permutation matrix, \mathbf{L} is unit lower triangular, and \mathbf{D} is block diagonal with blocks of size 1×1 and 2×2 .

The efficiency of `HSL_MA86` is dependent on the user-supplied elimination order. The HSL package `HSL_MC68` may be used to obtain a suitable ordering.

The lower triangular part of \mathbf{A} must be supplied in compressed sparse column format. The HSL package HSL_MC69 may be used to convert data held in other sparse matrix formats and also to check the user's matrix data for errors.

If \mathbf{A} is known to be positive definite (so that pivoting for numerical stability is not required), we recommend HSL_MA87.

ATTRIBUTES — **Version:** 1.0.0.. **Types:** Real (single, double), Complex (single, double). **Calls:** HSL_MC34, HSL_MC78, BLAS subroutines `_copy`, `_axpy`, `_swap`, `_gemm`, `_gemv`, `_trsm`, `_trsv`. **Date:** Original date: January 2011. **Origin:** J.D. Hogg and J.A. Scott, Rutherford Appleton Laboratory. **Language:** Fortran 95, plus allocatable components of derived types. **Parallelism:** Uses OpenMP and its runtime library. **Remark:** Development of HSL_MA86 was supported by EPSRC grant EP/E053351/1.

HSL_MA87 Sparse symmetric positive-definite system using OpenMP

HSL_MA87 uses a direct method to solve large sparse positive-definite symmetric linear systems of equations $\mathbf{AX} = \mathbf{B}$. This package uses OpenMP and is designed for multicore architectures. It computes the sparse Cholesky factorization

$$\mathbf{A} = \mathbf{PL}(\mathbf{PL})^*$$

where $\mathbf{L}^* = \mathbf{L}^T$ (real symmetric) or $\mathbf{L}^* = \mathbf{L}^H$ (complex Hermitian), \mathbf{P} is a permutation matrix and \mathbf{L} is lower triangular.

The efficiency of HSL_MA87 is dependent on the user-supplied elimination order. The HSL package HSL_MC68 may be used to obtain a suitable ordering.

The lower triangular part of \mathbf{A} must be supplied in compressed sparse column format. The HSL package HSL_MC69 may be used to convert data held in other sparse matrix formats and also to check the user's matrix data for errors.

If \mathbf{A} is indefinite and pivoting for numerical stability is required, the package HSL_MA86 should be used.

ATTRIBUTES — **Version:** 2.0.0.. **Types:** Real (single, double), Complex (single, double). **Calls:** HSL_MC34, HSL_MC78, BLAS subroutines `_gemm`, `_gemv`, `_herk`, `_syrc`, `_trsm`, `_trsv`, and the LAPACK subroutine `_potrf`. **Date:** Original date: January 2009, Version 2.0.0 December 2010. **Origin:** J.D. Hogg, J.K. Reid and J.A. Scott, Rutherford Appleton Laboratory. **Language:** Fortran 95, plus allocatable components of derived types. **Parallelism:** Uses OpenMP and its runtime library. **Remark:** Development of HSL_MA87 was supported by EPSRC grants EP/F006535/1 and EP/E053351/1.

MC Computations with matrices and vectors

MC13 Permute a sparse matrix to block triangular form

Given the pattern of nonzeros of a sparse matrix \mathbf{A} , finds a symmetric permutation that makes the matrix block lower triangular, i.e. finds \mathbf{P} such that $\mathbf{L} = \mathbf{PAP}^{-1}$ is block lower triangular.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** None. **Language:** Fortran 77. **Date:** March 1976. **Origin:** I.S. Duff, Harwell.

MC21 Permute a sparse matrix to put entries on the diagonal

Given the pattern of nonzeros of a sparse matrix this subroutine attempts to find a row permutation that makes the matrix have no zeros on its diagonal. The method used is a simple depth first search with a look ahead.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** None. **Language:** Fortran 77. **Date:** April 1977. **Origin:** I.S. Duff, Harwell.

MC22 Permute a sparse matrix given row and column permutations

Given a sparse matrix $\mathbf{A} = \{a_{ij}\}_{n \times n}$ and a row permutation matrix \mathbf{P} and a column permutation matrix \mathbf{Q} , this subroutine performs the permutation $\hat{\mathbf{A}} = \mathbf{PAQ}$. The nonzero entries of \mathbf{A} are stored by rows in a compact form and the user defines the permutation matrices \mathbf{P} and \mathbf{Q} by index vectors of length n .

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double), Complex (single, double). **Remark:** Supersedes ME22. **Calls:** None. **Language:** Fortran 77. **Date:** November 1976. **Origin:** I.S. Duff, Harwell.

MC25 Permute a sparse matrix to block triangular form

Permutates a sparse real or complex matrix to a block lower triangular form, i.e. given a sparse matrix $\mathbf{A} = \{a_{ij}\}_{n \times n}$, it attempts to find permutation matrices \mathbf{P} and \mathbf{Q} such that $\hat{\mathbf{A}} = \mathbf{PAQ}$ is block lower triangular and it returns $\hat{\mathbf{A}}$ to the caller.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double), Complex (single, double). **Calls:** MC13, MC21. **Language:** Fortran 77. **Date:** 1977, revised April 2001. **Remark:** MC25 is a threadsafe version of Mthird and ME23. **Origin:** I.S. Duff, Harwell.

MC26 Sparse rectangular matrix: compute normal matrix

Given a sparse matrix \mathbf{A} , this subroutine computes the sparse matrix $\mathbf{A}^T \mathbf{A}$. Three forms of data storage are permitted for the input matrix: storage by columns, where the row indices and column pointers describe the matrix; storage by rows, where the column indices and row pointers describe the matrix; and the coordinate scheme, where both row and column indices describe the position of entries in the matrix.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** MC59. **Language:** Fortran 77. **Date:** 1987, revised April 2001. **Remark:** MC26 is a threadsafe version of MC35. **Origin:** I.S. Duff, Harwell.

MC29 Sparse unsymmetric matrix: calculate scaling factors

This subroutine calculates scaling factors for a sparse matrix $\mathbf{A} = \{a_{ij}\}_{m \times n}$. They may be used, for instance, to scale the matrix prior to solving a corresponding set of linear equations, and are chosen so that the scaled matrix has its entries near to unity in the sense that the sum of the squares of the logarithms of the entries is minimized. The natural logarithms of the scaling factors r_i , $i = 1, 2, \dots, m$ for the rows and c_j , $j = 1, 2, \dots, n$ for the columns are returned so that the scaled matrix has entries

$$b_{ij} = a_{ij} \exp(r_i + c_j).$$

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Remark:** Supersedes MC19. **Calls:** None. **Language:** Fortran 77. **Date:** March 1993. **Origin:** J.K. Reid, Rutherford Appleton Laboratory.

MC30 Sparse symmetric matrix: calculate scaling factors

This subroutine calculates scaling factors for a symmetric sparse matrix $\mathbf{A} = \{a_{ij}\}_{n \times n}$. They may be used, for instance, to scale the matrix prior to solving a corresponding set of linear equations, and are chosen so that the scaled matrix has its entries near to unity in the sense that the sum of the squares of the logarithms of the entries is minimized. The natural logarithms of the scaling factors s_i , $i = 1, 2, \dots, n$ for the rows and columns are returned so that the scaled matrix has entries

$$b_{ij} = a_{ij} \exp(s_i + s_j).$$

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** None. **Language:** Fortran 77. **Date:** March 1993. **Origin:** J.K. Reid, Rutherford Appleton Laboratory.

MC33 Sparse unsymmetric matrix: permute to bordered block triangular form

This subroutine **finds row and column permutations** that reorder an m by n sparse matrix to a **bordered block triangular form with full diagonal blocks**, or a **nested bordered block triangular form** with each block itself in bordered block triangular form.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** MC59. **Language:** Fortran 77. **Date:** February 1987. **Origin:** M. Arioli, I.S. Duff, N.P. Storer, Harwell.

MC34 Sparse symmetric structure: expand from lower triangle

This subroutine generates the expanded structure for a sparse symmetric matrix given the structure for the lower triangular part. Diagonal entries need not be present.

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double). **Calls:** None. **Language:** Fortran 77. **Date:** February 1987. **Origin:** I.S. Duff, Harwell.

HSL_MC34 Sparse symmetric structure: expand from lower triangle

This subroutine generates the expanded structure for a sparse symmetric matrix given the structure for the lower triangular part. Diagonal entries need not be present.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double), Complex (single, double) **Calls:** None. **Language:** Fortran 95. **Date:** October 2009. **Origin:** J.A. Scott, Rutherford Appleton Laboratory.

MC37 Sparse symmetric matrix: represent as a sum of element matrices

Given a sparse symmetric matrix **A**, this subroutine computes a set of element matrices that, if assembled, would yield the same matrix. Note that this set of elements is not unique. The matrix can be input by the user either in compressed column format (column pointer/row index scheme) or by row and column index pairs in any order.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** MC59. **Language:** Fortran 77. **Date:** March 1995, revised August 2001. **Origin:** I.S. Duff, Rutherford Appleton Laboratory.

MC38 Sparse rectangular matrix held by columns: transpose

Given a sparse matrix held in a compressed column oriented format, this subroutine generates the transpose of the matrix, holding it in compressed column format. It can also be viewed as a conversion between a column oriented scheme and a row oriented one. This subroutine differs from MC46 inasmuch as it preserves the input data and should be faster, particularly on vector machines. However, it does require storage for both the matrix and its transpose.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** None. **Language:** Fortran 77. **Date:** June 1995. **Origin:** I.S. Duff, Rutherford Appleton Laboratory.

MC44 Unassembled finite-element matrix: generate the element or supervariable connectivity graph

Given the structure of an unassembled finite-element matrix, this subroutine groups the variables into supervariables and optionally generates either the element connectivity graph or the supervariable connectivity graph.

A supervariable is a collection of one or more variables, such that each variable belongs to the same set of finite elements. In the supervariable connectivity graph, the nodes are the supervariables and the edges are constructed by making the supervariables of each finite element pairwise adjacent. The supervariable connectivity graph, together with the number of variables in each supervariable, provide a compact representation of the variable connectivity graph. In the element connectivity graph, the nodes are the elements and the edges are constructed by defining two elements to be adjacent whenever they have one or more variables in common.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Remark:** MC44 is used by MC53. **Calls:** None. **Language:** Fortran 77. **Date:** September 1995. **Origin:** J.K. Reid and J.A. Scott, Rutherford Appleton Laboratory.

MC46 Sparse rectangular matrix held by rows: transpose

This subroutine takes an m by n sparse matrix \mathbf{A} , whose entries are stored by rows, and reorders it to be stored by columns. The subroutine differs from MC38 inasmuch as it requires less storage but may be slower, particularly on vector machines.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Remark:** See also MC38. **Calls:** None. **Language:** Fortran 77. **Date:** January 1989. **Origin:** N.I.M. Gould, Harwell.

MC47 Sparse symmetric pattern: approximate minimum-degree ordering allowing dense rows

Given the sparsity pattern of a symmetric matrix \mathbf{A} , MC47 uses an approximate minimum degree algorithm to compute a pivot order that is efficient when used with a sparse Cholesky solver. MC47 optionally allows for the efficient handling of dense or almost dense rows of \mathbf{A} . At each step, the pivot selected is the one that minimizes an upper-bound on the (external) degree. A permutation corresponding to this ordering is returned, together with information that may assist in the subsequent numerical factorization of the matrix.

The code is typically faster than other minimum degree algorithms and produces comparable results to other minimum external degree algorithms in terms of fill-in and the number of floating-point operations needed to compute the factors.

ATTRIBUTES — **Version:** 2.1.0. **Types:** Real (single, double). **Calls:** MC34, MC59. **Language:** Fortran 77. **Date:** March 1995, revised October 2007. **Origin:** Version 1: P.R. Amestoy (ENSEEIH, Toulouse, France), T.A. Davis (University of Florida) and I.S. Duff (Rutherford Appleton Laboratory). Version 2: P. R. Amestoy (ENSEEIH, Toulouse, France); H. S. Dollar, J. K. Reid and J. A. Scott (Rutherford Appleton Laboratory).

MC53 Generate an ordering for finite-element matrices within a subdomain

This subroutine generates an ordering for finite-element matrices within a subdomain that is efficient when subsequently used with a multiple front algorithm. In a multiple front algorithm, the finite-element domain is partitioned into a number of subdomains and a frontal decomposition is performed on each subdomain separately. The storage required by a multiple front algorithm and the time taken to run it are dependent upon the order in which the elements in each subdomain are input; the variation in the performance of different element orderings can be significant. The ordering obtained by MC53 is designed to reduce the maximum and root mean-squared wavefronts and to reduce the floating-point operation count for the frontal solver on the subdomain. The user is required to supply a list of the variables belonging to each element in the subdomain one at a time followed by a list of the variables lying on the subdomain interface using reverse communication.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** MC44 and KB07. **Language:** Fortran 77. **Date:** March 1995, revised August 2001. **Origin:** J.A. Scott, Rutherford Appleton Laboratory.

MC54 Write a sparse matrix in Rutherford-Boeing format

To write a sparse matrix in Rutherford-Boeing format. The matrix can be input as an assembled matrix in either column-oriented or coordinate form, or as an unassembled finite-element matrix.

ATTRIBUTES — **Version:** 1.0.1. **Types:** Real (single, double), Integer, Complex (single, double). **Remark:** This package is also included in the HSL Archive. **Calls:** MC59. **Language:** Fortran 77. **Date:** September 2000, revised August 2001. **Origin:** I.S. Duff, Rutherford Appleton Laboratory. **Licence:** A third-party licence for this package is available without charge.

MC55 Write a supplementary file in Rutherford-Boeing format

To write a supplementary file in Rutherford-Boeing format. There are many types of supplementary file for which the user should read the documentation on the Rutherford-Boeing Sparse Matrix Collection (RAL Report RAL-TR-97-031). These include right-hand sides, solution vectors, orderings, eigenvalues etc.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double), Integer, Complex (single, double). **Remark:** This package is also included in the HSL Archive. **Calls:** None. **Language:** Fortran 77. **Date:** September 2000, revised August 2001. **Origin:** I.S. Duff, Rutherford Appleton Laboratory.

MC56 Read a file or a supplementary file held in Rutherford-Boeing format

To read a file held in Rutherford-Boeing format. This may contain either a sparse matrix or supplementary data. There are many types of supplementary data for which the user should read the documentation on the Rutherford-Boeing Sparse Matrix Collection (RAL Report RAL-TR-97-031). These include right-hand sides, solution vectors, orderings, eigenvalues etc.

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double), Integer, Complex (single, double). **Remark:** This package is also included in the HSL Archive. **Calls:** None. **Language:** Fortran 77. **Date:** September 2000. **Origin:** I.S. Duff, Rutherford Appleton Laboratory. **Licence:** A third-party licence for this package is available without charge.

HSL_MC56 Read a file containing a sparse matrix held in format

To read a file containing a sparse matrix held in Rutherford-Boeing format and manipulate it to the desired sparse matrix format. The user may specify the storage format the matrix is returned in, and random values can be optionally generated to match the matrix pattern. Rutherford-Boeing format can be used to specify either matrix data or supplementary data, however this package only supports matrix data. To read supplementary data, the HSL routine MC56 can be used. To write data in Rutherford-Boeing format the HSL routines MC54 and MC55 can be used for matrix data and supplementary data, respectively.

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double), Integer, Complex (single, double). **Remark:** This package is also included in the HSL Archive. **Calls:** HSL_FA14, HSL_MC34, MC56, HSL_ZD11, HSL_ZD13 **Language:** Fortran 95 + TR 15581 (allocatable components). **Date:** April 2010. **Origin:** J.D. Hogg, Rutherford Appleton Laboratory. **Licence:** A third-party licence for this package is available without charge.

MC57 Assemble a set of finite-element matrices

This subroutine assembles a set of element matrices, that is, it forms the summation

$$\mathbf{A} = \sum_l \mathbf{A}^{[l]},$$

where each element matrix $\mathbf{A}^{[l]}$ has entries only in the principal submatrix corresponding to the variables in element l . Each $\mathbf{A}^{[l]}$ must be held in packed form as a small full square matrix, together with a list of the variables associated with element l . The assembled matrix \mathbf{A} has a symmetric sparsity pattern but may be unsymmetric. An option exists for assembling only the sparsity pattern of \mathbf{A} . If the variables are not indexed contiguously, absent rows and columns may optionally be removed.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** None. **Language:** Fortran 77. **Date:** December 1999. **Origin:** J.A. Scott, Rutherford Appleton Laboratory.

MC58 Estimate rank and find independent rows/columns of a sparse unsymmetric or rectangular matrix

To **estimate the rank and find a nonsingular submatrix** of an unsymmetric or rectangular sparse matrix \mathbf{A} using Gaussian elimination. The main entry performs a sparse LU factorization of the matrix optionally using rook pivoting. The factors are not returned.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** BLAS routines: `_AXPY`, `_GEMM`, `_GEMV`, `_SCAL`, `_SWAP`, `_RTSM`, and `_TRSV`. **Origin:** I.S. Duff, Rutherford Appleton Laboratory. **Date:** July 2007.

MC59 Sort a sparse matrix to an ordering by columns

This subroutine **performs an in-place sort on a sparse matrix to an ordering by columns**. There is an option for ordering the entries within each column by increasing row indices and an option for checking for indices that are out-of-range or duplicated.

ATTRIBUTES — **Version:** 1.0.2. **Types:** Real (single, double), Integer, Complex (single, double). **Remark:** This subroutine supersedes MC20, MC39, MC49, ME20, and MF49. **Calls:** None. **Language:** Fortran 77. **Date:** December 2000, revised September 2009. **Origin:** I.S. Duff and J.A. Scott, Rutherford Appleton Laboratory.

MC60 Sparse symmetric pattern: reduce the profile and wavefront

This subroutine uses a variant of Sloan's method to calculate a symmetric permutation that aims to **reduce the profile and wavefront of a sparse matrix \mathbf{A} with a symmetric sparsity pattern**. Alternatively, the Reverse Cuthill-McKee (RCM) method may be requested to reduce the bandwidth. There are optional facilities for looking for sets of columns with identical patterns and taking advantage of them. There is also an option for computing a row order that would be appropriate for use with a row-by-row frontal solver (for example, the equation entry to MA42 or MA43). These optional facilities may also be used independently.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** None. **Language:** Fortran 77. **Date:** January 1998. **Origin:** J.K. Reid and J.A. Scott, Rutherford Appleton Laboratory.

MC61 Straightforward interface to MC60

Let \mathbf{A} be an $n \times n$ sparse matrix with a symmetric sparsity pattern. Given the sparsity pattern of \mathbf{A} , this subroutine uses a variant of Sloan's method to calculate a symmetric permutation that aims to reduce the profile and wavefront of \mathbf{A} . Alternatively, the Reverse Cuthill-McKee (RCM) method may be requested to reduce the bandwidth, or the user may request an ordering for the rows of \mathbf{A} that is efficient when used with a row-by-row frontal solver (for example, equation entry to MA42).

MC61 provides the user with a straightforward interface to the MC60 package when detailed control of the steps in constructing a symmetric permutation or row ordering is not required.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Remark:** Supersedes MC40. **Calls:** MC60. **Language:** Fortran 77. **Date:** January 1998. **Origin:** J.K. Reid and J.A. Scott, Rutherford Appleton Laboratory.

MC62 Generate a row ordering for a row-by-row frontal solver

Given an $n \times n$ matrix $\mathbf{A} = a_{ij}$ with an unsymmetric sparsity pattern, this subroutine generates a row ordering for a row-by-row frontal solver (for example, the HSL packages MA42 and MA43).

MC62 generates a row ordering that is designed to reduce the maximum and mean row and column frontsizes, the maximum and mean frontal matrix size, and the sum of the lifetimes, which in turn reduce storage requirements and operation counts for the frontal solver. Only the pattern of the matrix is used. MC62 is not recommended if \mathbf{A} has one or more rows that are full or have a large number of nonzeros.

MC62 offers the option of generating the row graph of an $m \times n$ matrix \mathbf{A} . The nodes of the row graph are the rows of \mathbf{A} and two rows i and j ($i \neq j$) are defined to be adjacent if and only if there is at least one column k of \mathbf{A} for which $a_{ik} \cdot a_{jk} \neq 0$.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** MC38, MC60. **Language:** Fortran 77. **Date:** September 1998. **Origin:** J.A. Scott, Rutherford Appleton Laboratory.

MC63 Generate an element assembly ordering for a frontal solver

This subroutine uses a variant of Sloan's algorithm to generate an element assembly ordering that is efficient when subsequently used with a frontal solver (for example, the packages MA42 and MA62). The number of floating-point operations and the storage required by a frontal solver for an unassembled finite-element matrix are dependent upon the order in which the elements are assembled; the variation in the performance of different element orderings can be significant. The assembly ordering obtained by MC63 is designed to reduce the maximum and root-mean-square (r.m.s.) wavefronts and the profile, which in turn reduce storage requirements and computation times for the frontal solver. Only the pattern of the finite elements is used.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Remark:** Supersedes MC43. **Calls:** MC60. **Language:** Fortran 77. **Date:** March 1998. **Origin:** J.A. Scott, Rutherford Appleton Laboratory.

MC64 Permute and scale a sparse unsymmetric matrix to put large entries on the diagonal

Given a sparse matrix $\mathbf{A} = \{a_{ij}\}_{n \times n}$, this subroutine attempts to find a column permutation vector that makes the permuted matrix have n entries on its diagonal. If the matrix is structurally nonsingular, the subroutine optionally returns a column permutation that maximizes the smallest element on the diagonal, maximizes the sum of the diagonal entries, or maximizes the product of the diagonal entries of the permuted matrix. For the latter option, the subroutine also finds scaling factors that may be used to scale the original matrix so that the nonzero diagonal entries of the permuted and scaled matrix are one in absolute value and all the off-diagonal entries are less than or equal to one in absolute value. The natural logarithms of the scaling factors u_i , $i = 1, \dots, n$, for the rows and v_j , $j = 1, \dots, n$, for the columns are returned so that the scaled matrix $\mathbf{B} = \{b_{ij}\}_{n \times n}$ has entries

$$b_{ij} = a_{ij} \exp(u_i + v_j).$$

ATTRIBUTES — **Version:** 1.5.0. **Types:** Real (single, double). **Calls:** FD15, MC21. **Language:** Fortran 77. **Date:** July 1999, revised June 2007. **Origin:** I.S. Duff and J. Koster, Rutherford Appleton Laboratory.

HSL_MC64 Permute and scale a sparse unsymmetric or rectangular matrix to put large entries on the diagonal

Given a sparse unsymmetric or rectangular matrix $\mathbf{A} = \{a_{ij}\}_{m \times n}$, $m \geq n$, this subroutine attempts to find a row and column permutation that makes the permuted matrix have n entries on its diagonal. If the matrix is structurally nonsingular, the subroutine optionally returns a permutation that maximizes the smallest element on the diagonal, maximizes the sum of the diagonal entries, or maximizes the product of the diagonal entries of the permuted matrix. For the latter option, the subroutine also finds scaling factors that may be used to scale the original matrix so that the nonzero diagonal entries of the permuted and scaled matrix are one in absolute value and all the off-diagonal entries are less than or equal to one in absolute value. The natural logarithms of the scaling factors u_i , $i = 1, \dots, n$, for the rows and v_j , $j = 1, \dots, n$, for the columns are returned so that the scaled matrix $\mathbf{B} = \{b_{ij}\}_{n \times n}$ has entries

$$b_{ij} = a_{ij} \exp(u_i + v_j).$$

In this Fortran 95 version, there are added facilities from the original MC64 code for working on rectangular and symmetric matrices. For the rectangular case, a **row and column permutation** are returned so that the user can permute the matching to the diagonal and identify the rows in the structurally nonsingular block. For the symmetric case, the user must only supply the lower triangle and, if a scaling is computed, it will be a symmetric scaling with the same property as in the unsymmetric case.

ATTRIBUTES — **Version:** 2.0.1. **Types:** Real (single, double). **Remark:** HSL_MC64 is a Fortran 95 encapsulation of MC64 and offers additional facilities to the Fortran 77 version. **Calls:** MC64, FD15, MC21, ZD11. **Language:** Fortran 95. **Date:** July 2007, revised August 2010 **Origin:** I.S. Duff, Rutherford Appleton Laboratory and J. Koster, Trondheim.

HSL_MC65 Construct and manipulate matrices in compressed sparse row format

A suite of Fortran 95 procedures for constructing and manipulating sparse matrix objects in compressed sparse row format.

For a general sparse matrix, the compressed sparse row format consists of three arrays, PTR, COL and VAL. PTR holds the starting positions of the rows in the COL and VAL arrays. The indices of the entries of row I are held in COL(PTR(I):PTR(I+1)-1) and the corresponding values are held in VAL(PTR(I):PTR(I+1)-1). However, the user should not need to deal with these arrays individually; HSL_MC65 encapsulates them in a sparse matrix object of the derived type HSL_ZD01_TYPE. HSL_MC65 provides procedures that perform basic operations, such as sparse matrix summation and multiplication, on these sparse matrix objects. There is an option for omitting VAL, that is, for a pattern-only matrix.

ATTRIBUTES — **Version:** 2.1.0. **Types:** Real (single, double). **Calls:** HSL_ZD11. **Date:** November 2000. **Origin:** Y.F. Hu, Daresbury Laboratory. **Language:** Fortran 95 + TR 15581 (allocatable components).

HSL_MC66 Permute an unsymmetric sparse matrix to singly bordered blocked diagonal form

To order an unsymmetric matrix \mathbf{A} into singly bordered blocked diagonal (SBBB) form. Given the sparsity pattern of a matrix, this routine generates a row and column ordering that can be used to reorder the matrix into the following SBBB form:

$$\begin{pmatrix} \mathbf{A}_{11} & & & \mathbf{S}_1 \\ & \mathbf{A}_{22} & & \mathbf{S}_2 \\ & & \cdots & \cdots \\ & & & \cdots \\ & & & \cdots \\ & & & \mathbf{A}_{KK} & \mathbf{S}_K \end{pmatrix}.$$

Here K is the user-defined number of blocks. The aim is to minimize the size of the border in the above matrix, also known as the *net-cut*, and to achieve *load balance* by ensuring that the rectangular matrices \mathbf{A}_{ii} are of similar sizes.

The MC66 algorithm uses a multilevel approach combined with a Kernighan-Lin type refinement algorithm. Full details are discussed in Hu, Maguire and Blake, *Computers and Chemical Engrg.* **21** (2000), pp.1631-1647.

MC66 may be used to preorder an unsymmetric matrix for use with the sparse matrix solver HSL_MP43.

ATTRIBUTES — **Version:** 2.0.0. **Types:** Real (single, double). **Calls:** HSL_FA14, HSL_MC65, HSL_ZD11. **Date:** January 2001. **Origin:** Y.F. Hu, Daresbury Laboratory. **Language:** Fortran 95 + TR 15581 (allocatable components).

MC67 Refine a profile-reducing permutation of a symmetric matrix

Given the sparsity pattern of an $n \times n$ symmetric matrix \mathbf{A} and a symmetric permutation that reduces the profile of \mathbf{A} , this routine computes a new symmetric permutation with a smaller **profile**. The exchange algorithms of Hager are used to refine the given permutation.

Any zeros on the diagonal of \mathbf{A} are regarded as nonzero. If m_i is the column index of the first nonzero in row i ($m_i \leq i$), the length of row i is $i - m_i + 1$ and the profile of \mathbf{A} is the sum of the lengths of the rows.

MC61 (or MC60) may be used to obtain an initial symmetric permutation.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** KB06. **Date:** February 2002. **Origin:** J. K. Reid and J. A. Scott (Rutherford Appleton Laboratory). **Language:** Fortran 77.

HSL_MC68 Symmetric sparse matrix: compute elimination orderings

Given a symmetric sparse matrix $\mathbf{A} = \{a_{ij}\}_{n \times n}$, HSL_MC68 **computes elimination orderings** that are suitable for use with a sparse direct solver. Currently the following choices are available

- Approximate minimum degree ordering (with provision for some dense, rows and columns) using MC47,
- Minimum degree ordering using the methodology of MA27,
- Nested bisection ordering using MeTiS,
- MA47 ordering for indefinite matrices which may generate a combination of both 1×1 and 2×2 pivots.

ATTRIBUTES — **Version:** 3.0.0. **Types:** Real (single, double). **Calls:** HSL_ZB01, MC47, and (optionally) METIS_NODEND. **Date:** July 2007. **Origin:** H. S. Dollar and J. A. Scott, Rutherford Appleton Laboratory. **Language:** Fortran 95 + TR 15581 (allocatable components). **Remark:** The development of this package was supported by EPSRC grant GR/S42170.

HSL_MC69 Matrix format converter

HSL_MC69 offers routines for converting matrices held in a number of sparse matrix formats to the compressed sparse column (CSC) format used by many HSL routines. This format requires the entries within each column of \mathbf{A} to be ordered by increasing row index. For symmetric, skew symmetric or Hermitian matrices, only entries in the lower triangle are held. This format is the one used by many of the recent HSL packages; we shall refer to it as the standard HSL format.

Routines are offered for converting matrices held in lower or upper compressed sparse column format or in lower or upper compressed sparse row format or in coordinate format,

and for verification and correction of matrices believed to already be in standard HSL format. The conversion routines check the user-supplied data for errors and handle duplicate entries (they are summed) and out-of-range entries (they are discarded).

ATTRIBUTES — **Version:** 1.0.0.. **Types:** Real (single, double), Complex (single, double). **Calls:** None **Date:** Original date: January 2011. **Origin:** J.D. Hogg, J.K. Reid and H.S. Thorne Rutherford Appleton Laboratory. **Language:** Fortran 95, plus allocatable components of derived types. **Remark:** Development of HSL_MC69 was partially supported by EPSRC grant EP/F006535/1.

MC71 Unsymmetric matrix: estimate 1-norm

This subroutine estimates the 1-norm

$$\max_j \sum_{i=1}^n |a_{ij}|$$

of an $n \times n$ matrix \mathbf{A} given the ability to multiply a vector by both the matrix and its transpose. Because the explicit form of \mathbf{A} is not required, the subroutine can be used for estimating the norm of matrix functions such as the inverse. Additionally this subroutine is potentially useful for estimating condition numbers of a matrix when the matrix is sparse or not available explicitly.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** I_AMAX. **Language:** Fortran 77. **Date:** 1988, revised June 2001. **Remark:** MC71 is a threadsafe version of MC41. **Origin:** M. Arioli, I.S. Duff, Harwell.

MC72 Full unsymmetric matrix: calculate scaling factors

Calculate scaling factors for the rows and columns of an n by n real or complex matrix. If the scaling is applied before Gaussian elimination with pivoting the choice of pivots will more likely lead to low growth in round-off errors.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double), Complex (single, double). **Calls:** None. **Language:** Fortran 77. **Date:** 1990, revised May 2001. **Remark:** MC72 is a threadsafe version of MC42 and MF42. **Origin:** S. Marlow, Harwell.

HSL_MC73 Sparse symmetric matrix: compute Fiedler vector and permute to reduce the profile and wavefront

Let \mathbf{A} be an $n \times n$ matrix with a symmetric sparsity pattern. HSL_MC73 has entries to compute the (approximate) Fiedler vector of the unweighted or weighted Laplacian matrix of \mathbf{A} and to compute a symmetric permutation that reduces the profile and wavefront of \mathbf{A} by using a multilevel algorithm. A number of profile reduction algorithms are offered:

- (1) The multilevel Sloan algorithm,
- (2) A multilevel spectral ordering algorithm, and
- (3) A hybrid algorithm that refines the multilevel spectral ordering (2) using MC60.

In each case, an option exists to refine the computed ordering using the Hager exchange algorithm (MC67).

If Hager exchanges are not employed, the orderings computed using (1) and (3) generally yield smaller profiles and wavefronts than the spectral ordering (2). For some problems, (1) yields smaller profiles and wavefronts than (3), but for others the converse is true. Algorithm (1) is faster than (3). Using Hager exchanges can substantially increase the ordering cost but can give worthwhile reductions in the profile and wavefront.

ATTRIBUTES — **Version:** 2.2.0. **Types:** Real (single, double). **Precision:** At least 8-byte arithmetic is recommended. **Calls:** HSL_MC65, HSL_ZD11, FA14, KB07, MC60,

MC61, MC67, _AXPY, _NRM2, _COPY, _DOT. **Language:** Fortran 95 + TR 15581 (allocatable components). **Date:** October 2002. **Origin:** Y.F. Hu, Daresbury Laboratory and J.A. Scott, Rutherford Appleton Laboratory.

MC75 Sparse unsymmetric matrix: estimate condition number

This subroutine provides estimates of the classical condition number, $\|\mathbf{A}\|_\infty\|\mathbf{A}^{-1}\|_\infty$, and Skeel's condition number, $\|\mathbf{A}^{-1}\|\|\mathbf{A}\|_\infty$, of an $n \times n$ sparse matrix \mathbf{A} . We denote by $|\mathbf{A}|$ and $|\mathbf{A}^{-1}|$ the matrices whose entries are the absolute values of the entries in \mathbf{A} and \mathbf{A}^{-1} .

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double). **Calls:** MA48, MC71, I_AMAX. **Language:** Fortran 77. **Date:** 1988, revised September 2001. **Remark:** MC75 is a threadsafe version of MC45. **Origin:** M. Arioli, I.S. Duff, Harwell.

MC77 Sparse unsymmetric matrix: calculate scaling factors

This subroutine calculates scaling factors for a sparse symmetric or unsymmetric matrix $\mathbf{A} = \{a_{ij}\}_{m \times n}$ to make the p -norms of all the rows and columns be approximately equal to 1. In the symmetric case, the scaling is symmetric. The matrix may be stored by columns, in the coordinate format, or as a packed dense matrix. If $m \neq n$, it uses the ∞ -norm. Otherwise, the user may choose any p -norm, with $p \geq 1$.

ATTRIBUTES — **Version:** 1.0.1. **Types:** Real (single, double), Complex (single, double). **Calls:** None. **Language:** Fortran 77. **Date:** July 2004. **Origin:** Daniel Ruiz, ENSEEIHT, Toulouse (France).

HSL_MC78 Analysis phase in Cholesky algorithm

Given an elimination order, HSL_MC78 performs common tasks required in the analyse phase of a symmetric sparse direct solver. Either the entire analyse may be performed or individual tasks. No checking is performed on the validity of user data, and failure to supply valid data will result in undefined behaviour.

Given the sparsity pattern of a sparse symmetric matrix \mathbf{A} and permutation \mathbf{P} , HSL_MC78 finds the pattern of the Cholesky factor \mathbf{L} such that $\mathbf{PAP}^{-1} = \mathbf{LL}^T$. The pattern of A may be provided in either assembled or elemental format. The permutation \mathbf{P} is referred to as the *elimination (pivot) order*.

To reduce the amount of matrix data read during the analysis, supervariables of \mathbf{A} may be identified. A *supervariable* is a set of columns of \mathbf{A} that have the same sparsity pattern.

An *elimination tree* is built that describes the structure of the Cholesky factor in terms of data dependence between pivotal columns. This allows permutations of the elimination order that do not affect the number of entries in \mathbf{L} to be identified and allows fast algorithms to be used in determining the exact structure of \mathbf{L} .

A *supernode* is a set of columns that have the same pattern in the matrix \mathbf{L} . This pattern is stored as a single row list for each supernode. The condensed version of the elimination tree consisting of supernodes is referred to as the *assembly tree*. To increase efficiency in a subsequent factorization phase, supernodes may be merged through a supernode amalgamation heuristic.

HSL_MC78 supports the use of 2×2 and larger block pivots.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Integer, Long integer. **Calls:** None. **Date:** October 2010. **Origin:** J. D. Hogg, Rutherford Appleton Laboratory. **Language:** Fortran 95 + TR 15581 (allocatable components).

HSL_MC79 Sparse matrix: maximum matching and Dulmage-Mendelsohn decomposition

Given the sparsity pattern of a rectangular sparse matrix $A = \{a_{ij}\}_{m \times n}$, HSL_MC79 has entries to compute a maximum matching, and a row permutation P and column permutation

Q such that PAQ is of block triangular form: a coarse Dulmage-Mendelsohn decomposition and a fine Dulmage-Mendelsohn decomposition are available.

A *matching* is a set of the rows \mathcal{R} and columns \mathcal{C} , where each row in $i \in \mathcal{R}$ is paired with a unique $j \in \mathcal{C}$ subject to $a_{ij} \neq 0$. The size of a matching is defined to be equal to the number of columns in \mathcal{C} . A *maximum matching* of A is a matching of A that has size greater than or equal to any other matching of A . The size of the maximum matching is equal to the *structural rank* of the matrix.

The *Dulmage-Mendelsohn decomposition* consists of a row permutation P and a column permutation Q such that

$$PAQ = \begin{array}{c} \mathcal{R}_1 \\ \mathcal{R}_2 \\ \mathcal{R}_3 \end{array} \begin{bmatrix} \mathcal{C}_1 & \mathcal{C}_2 & \mathcal{C}_3 \\ A_1 & A_4 & A_6 \\ 0 & A_2 & A_5 \\ 0 & 0 & A_3 \end{bmatrix},$$

where A_1 , formed by the rows in the set \mathcal{R}_1 and the columns in the set \mathcal{C}_1 , is an underdetermined matrix with m_1 rows and n_1 columns ($m_1 < n_1$ or $m_1 = n_1 = 0$); A_2 , formed by the rows in the set \mathcal{R}_2 and the columns in the set \mathcal{C}_2 , is a square, well-determined matrix with m_2 rows; A_3 , formed by the rows in the set \mathcal{R}_3 and the columns in the set \mathcal{C}_3 , is an overdetermined matrix with m_3 rows and n_3 columns ($m_3 > n_3$ or $m_3 = n_3 = 0$). In particular, let the set of rows \mathcal{R} and the set of columns \mathcal{C} form a maximum matching of A . The sets \mathcal{R}_1 and \mathcal{R}_2 are subsets of \mathcal{R} , and $\mathcal{R}_3 \cap \mathcal{R}$ has n_3 entries. The sets \mathcal{C}_2 and \mathcal{C}_3 are subsets of \mathcal{C} , and $\mathcal{C}_1 \cap \mathcal{C}$ has m_1 entries.

The *coarse Dulmage-Mendelsohn decomposition* orders the unmatched columns as the first columns in PAQ and orders the unmatched rows as the last rows in PAQ . The output from the coarse Dulmage-Mendelsohn decomposition can be used to find a node separator from an edge separator of a graph.

The *fine Dulmage-Mendelsohn decomposition* computes a row permutation P and a column permutation Q such that A_1 and A_3 are block diagonal and each diagonal block is irreducible, and A_2 is block upper triangular with strongly connected (square) diagonal blocks. If A is reducible and nonsingular, the fine Dulmage-Mendelsohn decomposition of a matrix A can be used to solve the linear systems $Ax = b$ with block back-substitution.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Integer. **Calls:** None. **Date:** January 2011. **Origin:** H.S. Thorne, Rutherford Appleton Laboratory. **Language:** Fortran 95 + TR 15581 (allocatable components).

ME Solution of complex linear systems and other calculations for complex matrices

ME22 Permute a sparse matrix given row and column permutations

Given a sparse complex matrix $\mathbf{A} = \{a_{ij}\}_{n \times n}$ and a row permutation matrix \mathbf{P} and a column permutation matrix \mathbf{Q} , this subroutine performs the permutation $\hat{\mathbf{A}} = \mathbf{PAQ}$. The non-zero elements of \mathbf{A} are stored by rows in a compact form and the user defines the permutation matrices \mathbf{P} and \mathbf{Q} by index vectors of length n .

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Remark:** Superseded by MC22. **Calls:** None. **Language:** Fortran 77. **Date:** August 1979. **Origin:** I.S. Duff, Harwell.

ME38 Sparse unsymmetric system: unsymmetric multifrontal method

This package solves a sparse unsymmetric complex system of n linear equations in n unknowns using an unsymmetric multifrontal variant of Gaussian elimination. There are facilities for choosing a good pivot order, factorizing another matrix with a nonzero pattern identical to that of a previously factorized matrix, and solving a system of equations using

the factorized matrix. An option exists for solving triangular systems using the factors from the Gaussian elimination.

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double). **Calls:** FD15, MC13, MC21, _TRSV, _TRSM, _GEMV, I_AMAX, _GEMM. **Language:** Fortran 77. **Date:** September 2000. **Origin:** T.A. Davis, University of Florida, and I.S. Duff, Rutherford Appleton Laboratory.

ME42 Sparse unsymmetric system: out-of-core frontal method

To solve one or more sets of sparse linear complex equations, $\mathbf{Ax} = \mathbf{b}$, $\mathbf{A}^T \mathbf{x} = \mathbf{b}$ or $\mathbf{A}^H \mathbf{x} = \mathbf{b}$, by the frontal method, optionally using direct-access files for the matrix factors so that large problems can be solved in a relatively small in-core memory ($\mathbf{A}^H \mathbf{x} = \mathbf{b}$ is the transpose of the complex conjugate of \mathbf{A}). Use is made of high level BLAS kernels. The code has low in-core memory requirements. The complex matrix \mathbf{A} may be input by the user in either of the following ways:

- (i) by elements in a finite-element calculation,
- (ii) by equations (matrix rows).

ATTRIBUTES — **Version:** 1.2.0. **Types:** Real (single, double). **Remark:** ME42 is a complex version of MA42. **Calls:** _AXPY, GERU, _GEMV, _TPSV, _TRSM, _GEMM. **Language:** Fortran 77. **Date:** March 1993. **Origin:** I.S. Duff and J.A. Scott, Rutherford Appleton Laboratory.

ME43 Sparse unsymmetric system: row-by-row frontal method

To solve one or more sets of variable-band complex linear equations, $\mathbf{Ax} = \mathbf{b}$, $\mathbf{A}^T \mathbf{x} = \mathbf{b}$ or $\mathbf{A}^H \mathbf{x} = \mathbf{b}$, by the frontal method ($\mathbf{A}^H \mathbf{x} = \mathbf{b}$ is the transpose of the complex conjugate of \mathbf{A}). ME43 provides the user with a straightforward interface to the HSL routine ME42 when entry is by equations and auxiliary storage is not required. If the user requires more sophisticated facilities, ME42 should be employed.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Remark:** ME43 is a complex version of MA43. **Calls:** ME42, MC59. **Language:** Fortran 77. **Date:** March 1993, revised August 2001. **Origin:** J.A. Scott, Rutherford Appleton Laboratory.

ME48 Sparse unsymmetric system: driver for conventional direct method

To solve a sparse unsymmetric system of m complex linear equations in n unknowns using Gaussian elimination. There are facilities for block triangularization, choosing a good pivot order, factorizing a matrix with a given pivot order, and solving a system of equations using the factorized matrix. Error estimates are available.

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double). **Remark:** Supersedes ME28. **Calls:** FD15, ME50, MC13, MC21, MF71. **Language:** Fortran 77. **Date:** March 1993, revised August 2001. **Origin:** I.S. Duff and J.K. Reid, Rutherford Appleton Library.

ME50 Sparse unsymmetric system: conventional direct method

These subroutines are for the solution of a general sparse $m \times n$ system of complex linear equations (the most usual case being square, $m = n$), stored by columns. No block triangularization, iterative refinement, or error estimation is included.

If the user requires a more convenient data interface, the ME48 package should be used. The ME48 subroutines call the ME50 subroutines after checking and sorting the user's input data and optionally using MC21 and MC13 to permute the matrix to block triangular form.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Remark:** Supersedes ME30 and is normally called through the ME48 package. **Calls:** `_AXPY`, `_DOTU`, `_DOTC`, `_GEMM`, `_GEMV`, `_SCAL`, `_SWAP`, `_TRSM`, `_TRSV`. **Language:** Fortran 77. **Date:** March 1993, revised August 2001. **Origin:** I.S. Duff and J.K. Reid, Rutherford Appleton Library.

ME57 Sparse Hermitian or complex symmetric: multifrontal method

To solve a sparse Hermitian or complex symmetric system of linear equations. Given a sparse Hermitian or complex symmetric matrix $\mathbf{A} = \{a_{ij}\}_{n \times n}$ and an n -vector \mathbf{B} (or an $n \times s$ matrix \mathbf{B}), this subroutine solves the system $\mathbf{Ax} = \mathbf{b}$ ($\mathbf{AX} = \mathbf{B}$).

ME57 is a complex version of the code MA57 that is described in the technical report “MA57 - a new code for the solution of sparse symmetric indefinite systems” by Duff (RAL-TR-2002-024). This can be obtained from the web site <http://www.numerical.rl.ac.uk/reports/reports.html>.

ATTRIBUTES — **Version:** 2.1.0. **Types:** Real (single, double). **Calls:** FD15, MF71, MC47 and BLAS routines `_GEMM`, `_TPSV`, and `_GEMV`. **Date:** October 2001. **Language:** Fortran 77. **Origin:** I. S. Duff, Rutherford Appleton Laboratory.

HSL_ME57 Sparse solver for complex symmetric or Hermitian linear systems

To solve a sparse complex symmetric or Hermitian system of linear equations. Given a sparse complex symmetric or Hermitian matrix $\mathbf{A} = \{a_{ij}\}_{n \times n}$ and an n -vector \mathbf{b} or a matrix $\mathbf{B} = \{b_{ij}\}_{n \times r}$, this subroutine solves the system $\mathbf{Ax} = \mathbf{b}$ or the system $\mathbf{AX} = \mathbf{B}$. The matrix \mathbf{A} can be either complex symmetric or Hermitian. There is an option for iterative refinement.

The multifrontal method is used. HSL_ME57 is a complex version of the code HSL_MA57. 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. Relevant work on pivoting and scaling strategies is given by Duff and Pralet, SIAM Journal Matrix Analysis and Applications **27** (2005), 313-340. More recent work on static pivoting is given by Duff and Pralet, SIAM Journal Matrix Analysis and Applications **29** (2007), 1007-1024.

The HSL_ME57 package has a range of options including several sparsity orderings, multiple right-hand sides, partial solutions, error analysis, scaling, a matrix modification facility, the efficient factorization of highly rank deficient systems, 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.

The package has facilities for automatic restarts when storage limits are exceeded, the return of information on pivots, permutations, scaling, modifications, and the possibility to alter the pivots a posteriori.

ATTRIBUTES — **Version:** 1.0.0.. **Types:** Real (single, double). **Remark:** HSL_ME57 is a Fortran 95 encapsulation of ME57 and offers some additional facilities to the Fortran 77 version. **Calls:** ME57, `_GEMM`, `_TPSV`, FD15, MF71, HSL_ZD11. **Language:** Fortran 95 + TR 15581 (allocatable components). **Date:** Original: August 2010. **Origin:** I.S. Duff, Rutherford Appleton Laboratory.

ME62 Sparse Hermitian or complex symmetric finite-element system: out-of-core frontal method

To solve one or more sets of sparse Hermitian or complex symmetric linear unassembled finite-element equations, $\mathbf{AX} = \mathbf{B}$, by the frontal method, optionally holding the matrix factor out-of-core in direct-access files. Numerical pivoting is **not** performed so for Hermitian

matrices it is primarily designed for the positive-definite case. Use is made of high-level BLAS kernels. The coefficient matrix \mathbf{A} must of the form

$$\mathbf{A} = \sum_{k=1}^m \mathbf{A}^{(k)},$$

with $\mathbf{A}^{(k)}$ nonzero only in those rows and columns that correspond to variables in the k -th element.

The frontal method is a variant of Gaussian elimination and involves the factorization

$$\mathbf{A} = \mathbf{PLD}(\mathbf{PL})^H \text{ (Hermitian case),}$$

or

$$\mathbf{A} = \mathbf{PLD}(\mathbf{PL})^T \text{ (symmetric case),}$$

where \mathbf{P} is a permutation matrix, \mathbf{D} is a diagonal matrix, and \mathbf{L} is a unit lower triangular matrix. The solution process is completed by performing the forward elimination

$$(\mathbf{PL})\mathbf{DY} = \mathbf{B},$$

followed by the back substitution

$$(\mathbf{PL})^H \mathbf{X} = \mathbf{Y} \text{ (Hermitian case)}$$

or

$$(\mathbf{PL})^T \mathbf{X} = \mathbf{Y} \text{ (symmetric case).}$$

ME62 stores the values of the entries in the factors and their indices separately. A principal feature of ME62 is that, by holding the factors out-of-core, large problems can be solved using a predetermined and relatively small amount of in-core memory. At an intermediate stage of the solution, l say, the ‘front’ contains those variables associated with one or more of $\mathbf{A}^{(k)}$, $k = 1, 2, \dots, l$, which are also present in one or more of $\mathbf{A}^{(k)}$, $k = l + 1, \dots, m$. For efficiency, the user should order the $\mathbf{A}^{(k)}$ so that the number of variables in the front (the ‘front size’) is small. For example, a very rectangular grid should be ordered pagewise parallel to the short side of the rectangle. The elements may be preordered using the HSL routine MC63.

ATTRIBUTES — **Version:** 1.0.1. **Types:** Real (single, double). **Remark:** Complex version of MA62. **Calls:** `_AXPY`, `_GERU`, `_GEMV`, `_TPSV`, `_TRSV`, `_GEMM`, `_TRSM`. **Helpful:** MC63. **Language:** Fortran 77. **Date:** November 1999. **Origin:** I.S. Duff and J.A. Scott, Rutherford Appleton Laboratory.

MF Computations with complex matrices and vectors

MF29 Sparse unsymmetric matrix: calculate scaling factors

This subroutine calculates scaling factors for a complex sparse matrix $\mathbf{A} = \{a_{ij}\}_{m \times n}$. They may be used, for instance, to scale the matrix prior to solving a corresponding set of linear equations, and are chosen so that the scaled matrix has its entries near to unity in the sense that the sum of the squares of the logarithms of the moduli of the entries is minimized. The natural logarithms of the scaling factors r_i , $i = 1, 2, \dots, m$ for the rows and c_j , $j = 1, 2, \dots, n$ for the columns are returned so that the scaled matrix has entries

$$b_{ij} = a_{ij} \exp(r_i + c_j).$$

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** None. **Language:** Fortran 77. **Date:** March 1993. **Origin:** J.K. Reid, Rutherford Appleton Laboratory.

MF30 Sparse symmetric matrix: calculate scaling factors

This subroutine calculates scaling factors for a Hermitian or complex symmetric sparse matrix $\mathbf{A} = \{a_{ij}\}_{n \times n}$. They may be used, for instance, to scale the matrix prior to solving a corresponding set of linear equations, and are chosen so that the scaled matrix has its entries near to unity in the sense that the sum of the squares of the logarithms of the entries is minimized. The natural logarithms of the scaling factors s_i , $i = 1, 2, \dots, n$ for the rows and columns are returned so that the scaled matrix has entries

$$b_{ij} = a_{ij} \exp(s_i + s_j).$$

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** None. **Language:** Fortran 77. **Date:** March 1993. **Origin:** J.K. Reid, Rutherford Appleton Laboratory.

MF64 Permute and scale a sparse complex unsymmetric matrix to put large entries on the diagonal

Given a sparse complex matrix $\mathbf{A} = \{a_{ij}\}_{n \times n}$, this subroutine attempts to find a column permutation vector that makes the permuted matrix have n entries on its diagonal. If the matrix is structurally nonsingular, the subroutine optionally returns a column permutation that maximizes the smallest modulus of an entry on the diagonal, maximizes the sum of the moduli of the diagonal entries, or maximizes the product of the moduli of the diagonal entries of the permuted matrix. For the latter option, the subroutine also finds scaling factors that may be used to scale the original matrix so that the diagonal entries of the permuted and scaled matrix are one in absolute value and all the off-diagonal entries are less than or equal to one in absolute value. The natural logarithms of the scaling factors u_i , $i = 1, \dots, n$, for the rows and v_j , $j = 1, \dots, n$, for the columns are returned so that the scaled matrix $\mathbf{B} = \{b_{ij}\}_{n \times n}$ has entries

$$b_{ij} = a_{ij} \exp(u_i + v_j).$$

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** FD15, MC21, MC64. **Date:** August 2007. **Origin:** I. S. Duff and J. Koster (Rutherford Appleton Laboratory). **Language:** Fortran 77.

MF71 Unsymmetric matrix: estimate 1-norm

This subroutine estimates the 1-norm ($\max_j \sum_{i=1}^n |a_{ij}|$) of an $n \times n$ complex matrix \mathbf{A} given the ability to multiply a vector by both the matrix and its conjugate transpose. Because the explicit form of \mathbf{A} is not required, the subroutine can be used for estimating the norm of matrix functions such as the inverse. Additionally this subroutine is potentially useful for estimating condition numbers of a matrix when the matrix is sparse or not available explicitly.

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double). **Calls:** None. **Language:** Fortran 77. **Date:** 1993, revised June 2001. **Remark:** MF71 is a threadsafe version of MF41. **Origin:** M. Arioli, I.S. Duff, Rutherford Appleton Laboratory.

MI Iterative methods for sparse linear systems**HSL_MI02 Symmetric possibly-indefinite system: SYMMBK method**

This routine uses the SYMMBK method to solve the $n \times n$ symmetric but possibly indefinite linear system $\mathbf{Ax} = \mathbf{b}$, optionally using preconditioning. If \mathbf{PP}^T is the preconditioning matrix, the routine actually solves the preconditioned system

$$\bar{\mathbf{A}}\bar{\mathbf{x}} = \bar{\mathbf{b}},$$

with $\bar{\mathbf{A}} = \mathbf{P}\mathbf{A}\mathbf{P}^T$ and $\bar{\mathbf{b}} = \mathbf{P}\mathbf{b}$ and recovers the solution $\mathbf{x} = \mathbf{P}^T\bar{\mathbf{x}}$. Reverse communication is used for preconditioning operations and matrix-vector products of the form $\mathbf{A}\mathbf{z}$.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** `_LAEV2`. **Language:** Fortran 95. **Date:** August 1996. **Origin:** N.I.M. Gould, Rutherford Appleton Laboratory.

MI11 Unsymmetric system: incomplete LU factorization

This routine forms an incomplete **LU** factorization of an $n \times n$ sparse unsymmetric matrix **A**. No fill-in is allowed. The entries of **A** are stored by rows. If **A** has zeros on the diagonal, the routine first finds a row permutation **Q** which makes the matrix have nonzeros on the diagonal. The incomplete **LU** factorization of the permuted matrix **QA** is then formed. **L** is lower triangular and **U** is unit upper triangular. The incomplete factorization may be used as a preconditioner when solving the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$. A second entry performs the preconditioning operations

$$\mathbf{y} = \mathbf{P}\mathbf{z} \quad \text{and} \quad \mathbf{y} = \mathbf{P}^T\mathbf{z},$$

where $\mathbf{P} = (\mathbf{LU})^{-1}\mathbf{Q}$ is the preconditioner.

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double). **Calls:** `FD15`, `MC21`, `MC22`, `MC38`. **Language:** Fortran 77. **Date:** April 1995, revised August 2001. **Origin:** N.I.M. Gould and J.A. Scott, Rutherford Appleton Laboratory.

MI12 Unsymmetric system: approximate-inverse preconditioner

This routine finds an approximate inverse **M** of an $n \times n$ sparse unsymmetric matrix **A** by attempting to minimize the difference between \mathbf{AM} and the identity matrix in the Frobenius norm. The process may be improved by first performing a block triangularization of **A** and then finding approximate inverses to the resulting diagonal blocks.

A second entry allows the user to form the matrix-vector products

$$\mathbf{y} = \mathbf{M}\mathbf{z} \quad \text{and} \quad \mathbf{y} = \mathbf{M}^T\mathbf{z}.$$

The principal use of such an approximate inverse is likely to be in preconditioning iterative methods for solving the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$.

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double). **Calls:** `FD15`, `MC25`, `_NRM2`, `_AXPY`, `_COPY`, `_SCAL`, `_GEMV`, and `_TRSV`. **Language:** Fortran 77. **Date:** April 1995, revised August 2001. **Origin:** N.I.M. Gould and J.A. Scott, Rutherford Appleton Laboratory.

HSL_MI13 Preconditioners for saddle-point systems

Given a block symmetric matrix

$$\mathbf{K}_H = \begin{pmatrix} \mathbf{H} & \mathbf{A}^T \\ \mathbf{A} & -\mathbf{C} \end{pmatrix},$$

where **H** has n rows and columns and **A** has m rows and n columns, this package constructs preconditioners of the form

$$\mathbf{K}_G = \begin{pmatrix} \mathbf{G} & \mathbf{A}^T \\ \mathbf{A} & -\mathbf{C} \end{pmatrix}.$$

Here, the leading block matrix **G** is a suitably chosen approximation to **H**; it may either be prescribed explicitly, in which case a symmetric indefinite factorization of \mathbf{K}_G will be

formed using HSL_MA57, or implicitly. In the latter case, \mathbf{K}_G will be ordered to the form

$$\mathbf{K}_G = \mathbf{P} \begin{pmatrix} \mathbf{G}_{11} & \mathbf{G}_{21}^T & \mathbf{A}_1^T \\ \mathbf{G}_{21} & \mathbf{G}_{22} & \mathbf{A}_2^T \\ \mathbf{A}_1 & \mathbf{A}_2 & -\mathbf{C} \end{pmatrix} \mathbf{P}^T$$

where \mathbf{P} is a permutation and \mathbf{A}_1 is an invertible sub-block (“basis”) of the columns of \mathbf{A} ; the selection and factorization of \mathbf{A}_1 uses HSL_MA48 — any dependent rows in \mathbf{A} are removed at this stage. Once the preconditioner has been constructed, solutions to the preconditioning system

$$\begin{pmatrix} \mathbf{G} & \mathbf{A}^T \\ \mathbf{A} & -\mathbf{C} \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \begin{pmatrix} \mathbf{a} \\ \mathbf{b} \end{pmatrix}$$

may be computed.

Full advantage is taken of any zero coefficients in the matrices \mathbf{H} , \mathbf{A} and \mathbf{C} .

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double). **Calls:** KB07, MC59, HSL_ZB01, HSL_ZD11, HSL_MA57, HSL_MA48, _GEMV, _POTRF, _POTRS. **Date:** July 2007. **Language:** Fortran 95 + TR 15581 (allocatable components). **Origin:** H. S. Dollar and N. I. M. Gould, Rutherford Appleton Laboratory. **Remark:** The development of this package was supported by EPSRC grant GR/S42170.

MI15 Unsymmetric system: flexible GMRES

This routine uses the ‘Flexible Generalized Minimal Residual’ method with restarts every m iterations, FGMRES(m), to solve the $n \times n$ unsymmetric linear system $\mathbf{Ax} = \mathbf{b}$, optionally using preconditioning. FGMRES(m) generalises the preconditioned GMRES(m) allowing the possibility of using a different right preconditioner $\mathbf{P}_R^{(i)}$ at each step in solving the preconditioned system

$$\bar{\mathbf{A}}\mathbf{x} = \bar{\mathbf{b}} \tag{1.2}$$

where $\bar{\mathbf{A}} = \mathbf{P}_L\mathbf{A}$ and $\bar{\mathbf{b}} = \mathbf{P}_L\mathbf{b}$ and \mathbf{P}_L is a left preconditioner. Reverse communication is used for preconditioning operations and matrix-vector products of the form \mathbf{Az} .

If $\mathbf{P}_R^{(i)}$ does not change at each step i , in exact arithmetic the algorithm computes the same solution as GMRES(m) applied to (1.2). FGMRES(m) needs more memory than GMRES(m) and, in particular, stores an additional $n \times n$ real matrix. However, in floating point arithmetic, FGMRES(m) is numerically more stable than the GMRES(m) method.

If Gaussian elimination with static pivoting option has been used to compute an approximate LU factorization of \mathbf{A} , FGMRES(m) can be used to recover full backward error stability. In this case the left preconditioner should be chosen to be the identity and the right preconditioner should be chosen to be $\mathbf{P}_R^{(i)} = \mathbf{P}_R = (\mathbf{LU})^{-1}$.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** FD15, _COPY", _DOT", _NRM2", _SCAL", _AXPY", _ROT", _ROTG", _TRSV", _GEMV". **Date:** April 2007. **Language:** Fortran 77. **Origin:** M. Arioli, Rutherford Appleton Laboratory.

HSL_MI20 Unsymmetric system: algebraic multigrid preconditioner

Given an $n \times n$ sparse unsymmetric matrix \mathbf{A} and an n -vector \mathbf{z} , HSL_MI20 computes the vector $\mathbf{x} = \mathbf{Mz}$, where \mathbf{M} is an algebraic multigrid (AMG) v -cycle preconditioner for \mathbf{A} . A classical AMG method is used. The matrix \mathbf{A} must have positive diagonal entries and (most of) the off-diagonals entries must be negative (the diagonal should be large compared to the sum of the off-diagonals). During the multigrid coarsening process, positive off-diagonal entries are ignored and, when calculating the interpolation weights, positive off-diagonal entries are added to the diagonal.

ATTRIBUTES — **Version:** 1.3.1. **Types:** Real (single, double). **Precision:** At least 8-byte arithmetic is recommended. **Calls:** HSL_MA48, HSL_MC65, HSL_ZD11, and the LAPACK routines _GETRF and _GETRS. **Date:** September 2006. **Language:** Fortran 95

+ TR 15581 (allocatable components), plus allocatable dummy arguments and allocatable components of derived types. **Origin:** J. W. Boyle, University of Manchester and J. A. Scott, Rutherford Appleton Laboratory. **Remark:** The development of HSL_MI20 was funded by EPSRC grants EP/C000528/1 and GR/S42170.

MI21 Symmetric positive-definite system: conjugate gradient method

This routine uses the Conjugate Gradient method to solve the $n \times n$ symmetric positive-definite linear system $\mathbf{Ax} = \mathbf{b}$, optionally using preconditioning. If \mathbf{PP}^T is the preconditioning matrix, the routine actually solves the preconditioned system

$$\bar{\mathbf{A}}\bar{\mathbf{x}} = \bar{\mathbf{b}},$$

with $\bar{\mathbf{A}} = \mathbf{PAP}^T$ and $\bar{\mathbf{b}} = \mathbf{Pb}$ and recovers the solution $\mathbf{x} = \mathbf{P}^T\bar{\mathbf{x}}$. Reverse communication is used for preconditioning operations and matrix-vector products of the form \mathbf{Az} .

ATTRIBUTES — **Version:** 1.2.1. **Types:** Real (single, double). **Calls:** FD15, _COPY, _DOT, _NRM2, _SCAL, _AXPY. **Language:** Fortran 77. **Date:** 1995, revised March 2001. **Remark:** MI21 is a threadsafe version of MI01A. **Origin:** N.I.M. Gould and J.A. Scott, Rutherford Appleton Laboratory.

MI23 Unsymmetric system: CGS (conjugate gradient squared) method

This routine uses the CGS (Conjugate Gradient Squared) method to solve the $n \times n$ unsymmetric linear system $\mathbf{Ax} = \mathbf{b}$, optionally using preconditioning. If $\mathbf{P}_L, \mathbf{P}_R$ are the preconditioning matrices, the routine actually solves the preconditioned system

$$\bar{\mathbf{A}}\bar{\mathbf{x}} = \bar{\mathbf{b}},$$

with $\bar{\mathbf{A}} = \mathbf{P}_L\mathbf{AP}_R$ and $\mathbf{bBAR} = \mathbf{P}_L\mathbf{b}$ and recovers the solution $\mathbf{x} = \mathbf{P}_R\bar{\mathbf{x}}$. If $\mathbf{P}_L = \mathbf{I}$, preconditioning is said to be from the right, if $\mathbf{P}_R = \mathbf{I}$, it is said to be from the left, and otherwise it is from both sides. Reverse communication is used for preconditioning operations and matrix-vector products of the form \mathbf{Az} .

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double). **Calls:** FD15, _COPY, _DOT, _NRM2, _SCAL, _AXPY. **Language:** Fortran 77. **Date:** 1995, revised March 2001. **Remark:** MI23 is a threadsafe version of MI03A. **Origin:** N.I.M. Gould and J.A. Scott, Rutherford Appleton Laboratory.

MI24 Unsymmetric system: GMRES (generalized minimal residual) method

This routine uses the Generalized Minimal Residual method with restarts every m iterations, GMRES(m), to solve the $n \times n$ unsymmetric linear system $\mathbf{Ax} = \mathbf{b}$, optionally using preconditioning. If $\mathbf{P}_L, \mathbf{P}_R$ are left and right preconditioning matrices, the routine actually solves the preconditioned system

$$\bar{\mathbf{A}}\bar{\mathbf{x}} = \bar{\mathbf{b}},$$

with $\bar{\mathbf{A}} = \mathbf{P}_L\mathbf{AP}_R$ and $\bar{\mathbf{b}} = \mathbf{P}_L\mathbf{b}$. The solution may be recovered as $\mathbf{x} = \mathbf{P}_R\bar{\mathbf{x}}$. If $\mathbf{P}_L = \mathbf{I}$, preconditioning is said to be from the right, if $\mathbf{P}_R = \mathbf{I}$, it is said to be from the left, and otherwise it is from both sides. Reverse communication is used for preconditioning operations and matrix-vector products of the form \mathbf{Az} .

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double). **Calls:** FD15, _COPY, _DOT, _NRM2, _SCAL, _AXPY, _ROT, _ROTG, _TRSV, _GEMV. **Language:** Fortran 77. **Date:** 1995, revised March 2001. **Remark:** MI24 is a threadsafe version of MI04A. **Origin:** N.I.M. Gould and J.A. Scott, Rutherford Appleton Laboratory.

MI25 Unsymmetric system: BiCG (BiConjugate Gradient) method

This routine uses the BiCG (BiConjugate Gradient) method to solve the $n \times n$ unsymmetric linear system $\mathbf{Ax} = \mathbf{b}$, optionally using preconditioning. If \mathbf{P}_L , \mathbf{P}_R are the preconditioning matrices, the routine actually solves the preconditioned system

$$\bar{\mathbf{A}}\bar{\mathbf{x}} = \bar{\mathbf{b}},$$

with $\bar{\mathbf{A}} = \mathbf{P}_L\mathbf{A}\mathbf{P}_R$ and $\bar{\mathbf{b}} = \mathbf{P}_L\mathbf{b}$ and recovers the solution $\mathbf{x} = \mathbf{P}_R\bar{\mathbf{x}}$. If $\mathbf{P}_L = \mathbf{I}$, preconditioning is said to be from the right, if $\mathbf{P}_R = \mathbf{I}$, it is said to be from the left, and otherwise it is from both sides. Reverse communication is used for preconditioning operations \mathbf{Pz} and $\mathbf{P}^T\mathbf{z}$, where $\mathbf{P} = \mathbf{P}_L\mathbf{P}_R$, and for matrix-vector products of the form \mathbf{Az} and $\mathbf{A}^T\mathbf{z}$.

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double). **Calls:** FD15, _COPY, _DOT, _NRM2, _AXPY. **Language:** Fortran 77. **Date:** 1995, revised March 2001. **Remark:** MI25 is a threadsafe version of MI05A. **Origin:** N.I.M. Gould and J.A. Scott, Rutherford Appleton Laboratory.

MI26 Unsymmetric system: BiCGStab (BiConjugate Gradient Stabilized) method

This routine uses the BiCGStab (BiConjugate Gradient Stabilized) method to solve the $n \times n$ unsymmetric linear system $\mathbf{Ax} = \mathbf{b}$, optionally using preconditioning. If \mathbf{P}_L , \mathbf{P}_R are the preconditioning matrices, the routine actually solves the preconditioned system

$$\bar{\mathbf{A}}\bar{\mathbf{x}} = \bar{\mathbf{b}},$$

with $\bar{\mathbf{A}} = \mathbf{P}_L\mathbf{A}\mathbf{P}_R$ and $\bar{\mathbf{b}} = \mathbf{P}_L\mathbf{b}$ and recovers the solution $\mathbf{x} = \mathbf{P}_R\bar{\mathbf{x}}$. If $\mathbf{P}_L = \mathbf{I}$, preconditioning is said to be from the right, if $\mathbf{P}_R = \mathbf{I}$, it is said to be from the left, and otherwise it is from both sides. Reverse communication is used for preconditioning operations and matrix-vector products of the form \mathbf{Az} .

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double). **Calls:** FD15, _COPY, _DOT, _NRM2, _SCAL, _AXPY. **Language:** Fortran 77. **Date:** 1995, revised March 2001. **Remark:** MI26 is a threadsafe version of MI06A. **Origin:** N.I.M. Gould and J.A. Scott, Rutherford Appleton Laboratory.

HSL_MI27 Projected preconditioned conjugate gradient method for saddle-point systems

This package uses the projected preconditioned conjugate gradient method to solve $(n + m) \times (n + m)$ saddle-point systems of the form

$$\begin{bmatrix} \mathbf{A} & \mathbf{B}^T \\ \mathbf{B} & -\mathbf{C} \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} = \begin{bmatrix} \mathbf{C} \\ \mathbf{d} \end{bmatrix},$$

where \mathbf{A} is an $n \times n$ real and symmetric matrix, \mathbf{C} is an $m \times m$ real, symmetric and positive semi-definite (possibly zero) matrix, and $m \leq n$. A preconditioner of the form

$$\mathbf{P} = \begin{bmatrix} \mathbf{G} & \mathbf{B}^T \\ \mathbf{B} & -\mathbf{C} \end{bmatrix}$$

must be available where \mathbf{G} is a real and symmetric matrix. The following assumptions are assumed to hold:

- if \mathbf{C} is positive definite, both $\mathbf{A} + \mathbf{B}^T\mathbf{C}^{-1}\mathbf{B}$ and $\mathbf{G} + \mathbf{B}^T\mathbf{C}^{-1}\mathbf{B}$ are positive definite;
- if $\mathbf{C} = 0$ and the columns of the $n \times (n - m)$ matrix \mathbf{Z} span the nullspace of \mathbf{B} , both $\mathbf{Z}^T\mathbf{AZ}$ and $\mathbf{Z}^T\mathbf{GZ}$ are positive definite;

- if $\mathbf{C} = \mathbf{E}\mathbf{D}\mathbf{E}^T$, where \mathbf{d} is a $p \times p$ nonsingular matrix with $0 < p < m$, the columns of the $m \times (m - p)$ matrix \mathbf{F} span the nullspace of \mathbf{C} and the columns of the $n \times (n - m + p)$ matrix \mathbf{Z} span the nullspace of $\mathbf{F}^T\mathbf{B}$, then both $\mathbf{Z}^T(\mathbf{A} + \mathbf{B}^T\mathbf{E}\mathbf{D}^{-1}\mathbf{E}^T\mathbf{B})\mathbf{Z}$ and $\mathbf{Z}^T(\mathbf{G} + \mathbf{B}^T\mathbf{E}\mathbf{D}^{-1}\mathbf{E}^T\mathbf{B})\mathbf{Z}$ are positive definite.

If these assumptions do not hold, then negative curvature may occur and, consequently, the method terminates with an error.

The projected preconditioned conjugate gradient method iteratively finds the vector \mathbf{x} and then, once \mathbf{x} has been computed to a high enough level of accuracy, the vector \mathbf{y} is computed by performing one additional solve with the preconditioner \mathbf{P} . Reverse communication is used for preconditioning and matrix-vector products of the form $\mathbf{A}\mathbf{s}$, $\mathbf{B}\mathbf{s}$, $\mathbf{B}^T\mathbf{s}$ and $\mathbf{C}\mathbf{s}$. HSL_MI13 may be used to efficiently form suitable preconditioners and carry out the required preconditioning solves; HSL_MC65 may be used to form the required matrix-vector products. HSL_MI13 and HSL_MC65 are both available as part of HSL 2011.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Uses:** The BLAS subroutines `_AXPY`, `_COPY`, `_DOT`, `_NRM2`, `_SCAL`. **Date:** January 2011. **Origin:** H. S. Thorne, Rutherford Appleton Laboratory. **Language:** Fortran 95, plus allocatable dummy arguments and allocatable components of derived types. **Remark:** The development of HSL_MI27 was funded by EPSRC grant EP/E053351/1.

HSL_MI31 Symmetric positive-definite system: conjugate gradient method, stopping according to the A-norm of the error

The package uses the preconditioned conjugate gradient method to solve the $n \times n$ symmetric positive-definite linear system

$$\mathbf{A}\mathbf{u} = \mathbf{b},$$

and implements several stopping criteria based on lower and upper bounds of the A-norm of the error. If $\mathbf{M} = \mathbf{U}^T\mathbf{U}$ is the preconditioning matrix, the routine actually solves the preconditioned system

$$\bar{\mathbf{A}}\mathbf{Y} = \bar{\mathbf{b}},$$

with $\bar{\mathbf{A}} = \mathbf{U}^{-T}\mathbf{A}\mathbf{U}^{-1}$ and $\bar{\mathbf{b}} = \mathbf{U}^{-T}\mathbf{b}$ and recovers the solution $\mathbf{u} = \mathbf{U}^{-1}\mathbf{y}$.

Reverse communication is used. Control is returned to the user for preconditioning operations and the products of \mathbf{A} with a vector \mathbf{z} .

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double). **Calls:** `FD15`, `_DOT`, `_NRM2`, `_SCAL`, `_AXPY`. **Language:** Fortran 95. **Date:** July 2004. **Origin:** M. Arioli, Rutherford Appleton Laboratory, and Gianmarco Manzini, IMATI-CNR, Pavia, Italy.

MP Multiprocessor specific packages

HSL_MP01 MPI constants

The module HSL_MP01 is a simple package that encapsulates all necessary include files for the use of MPI. It is intended to be used in place of an `include 'mpif.h'` statement, thus allowing other library MPI packages to be written in free source form.

ATTRIBUTES — **Version:** 1.0.0. **Types:** None. **Calls:** None. **Language:** Fortran 95. **Date:** July 2008. **Origin:** J. D. Hogg, Rutherford Appleton Laboratory.

HSL_MP42 Unsymmetric finite-element system: multiple-front method, element entry

The module HSL_MP42 uses the multiple front method to solve sets of finite-element equations $\mathbf{A}\mathbf{X} = \mathbf{B}$ that have been divided into non-overlapping subdomains. The HSL routines MA42 and MA52 are used with MPI for message passing.

The coefficient matrix \mathbf{A} must be of the form

$$\mathbf{A} = \sum_{k=1}^m \mathbf{A}^{(k)}$$

where the summation is over finite elements. The element matrix $\mathbf{A}^{(k)}$ is nonzero only in those rows and columns which correspond to variables in the k -th element. The right-hand side(s) \mathbf{B} may optionally be in the form

$$\mathbf{B} = \sum_{k=1}^m \mathbf{B}^{(k)}$$

where $\mathbf{B}^{(k)}$ is nonzero only in those rows which correspond to variables in element k .

In the multiple front method, a frontal decomposition is performed on each subdomain separately. Thus, on each subdomain, \mathbf{L} and \mathbf{U} factors are computed. Once all possible eliminations have performed within a subdomain, there remain the interface variables, which are shared by more than one subdomain together with any variables that are not eliminated because of stability or efficiency considerations. If \mathbf{F}_i is the remaining frontal matrix for subdomain i , and \mathbf{C}_i is the corresponding right-hand side matrix, then the remaining problem is

$$\mathbf{F}\mathbf{Y} = \mathbf{C}, \quad (1.3)$$

where $\mathbf{F} = \sum_i \mathbf{F}_i$ and $\mathbf{C} = \sum_i \mathbf{C}_i$. By treating each \mathbf{F}_i as an element matrix, the interface problem (3) is also solved by the frontal method. Once (1.3) has been solved, back-substitution on the subdomains completes the solution.

The element data and/or the matrix factors are optionally held in direct-access files.

ATTRIBUTES — **Version:** 2.0.0. **Types:** Real (single, double). **Calls:** HSL_MP01, KB08, MA42, MA52, MC53, MC63. **Language:** Fortran 95 + TR 15581 (allocatable components). **Date:** September 1999. **Origin:** J.A. Scott, Rutherford Appleton Laboratory.

HSL_MP43 Sparse unsymmetric system: multiple-front method, equation entry

The module HSL_MP43 uses the multiple front method to solve sets of linear equations $\mathbf{A}\mathbf{x} = \mathbf{b}$ (or $\mathbf{A}\mathbf{X} = \mathbf{B}$) where \mathbf{A} has been preordered to singly-bordered block-diagonal form

$$\begin{pmatrix} \mathbf{A}_{11} & & & \mathbf{C}_1 \\ & \mathbf{A}_{22} & & \mathbf{C}_2 \\ & & \dots & \dots \\ & & & \dots \\ & & & \mathbf{A}_{NN} & \mathbf{C}_N \end{pmatrix}.$$

The HSL routines MA42 and MA52 are used with MPI for message passing.

In the multiple front method, a partial frontal decomposition is performed on each of the submatrices $(\mathbf{A}_l \mathbf{C}_l)$ separately. Thus, on each submatrix, \mathbf{L} and \mathbf{U} factors are computed. Once all possible eliminations have performed, for each submatrix there remains a frontal matrix \mathbf{F}_l . The variables that remain in the front are called interface variables and the interface matrix \mathbf{F} is formed by summing the matrices \mathbf{F}_l . The interface matrix \mathbf{F} is also factorized using the frontal method. Block back-substitution completes the solution.

The matrix data and/or the matrix factors are optionally held in direct-access files.

ATTRIBUTES — **Version:** 2.0.0. **Types:** Real (single, double). **Remark:** If \mathbf{A} is a sum of finite elements, use HSL_MP42 or HSL_MP62. **Calls:** HSL_MP01, KB08, MA42, MA52, MC62. **Language:** Fortran 95 + TR 15581 (allocatable components). **Date:** September 2000. **Origin:** J.A. Scott, Rutherford Appleton Laboratory.

HSL_MP48 Sparse unsymmetric system: parallel direct method

The module HSL_MP48 solves sets of $n \times n$ unsymmetric linear systems of equations $\mathbf{Ax} = \mathbf{b}$, in parallel using Gaussian elimination. The matrix \mathbf{A} must have been preordered to singly-bordered block-diagonal form

$$\begin{pmatrix} \mathbf{A}_{11} & & & \mathbf{C}_1 \\ & \mathbf{A}_{22} & & \mathbf{C}_2 \\ & & \cdots & \cdots \\ & & & \cdots \\ & & & \mathbf{A}_{NN} & \mathbf{C}_N \end{pmatrix}.$$

MPI is used for message passing.

A partial \mathbf{LU} decomposition is performed on each of the submatrices $(\mathbf{A}_l \mathbf{C}_l)$ separately. Once all possible eliminations have been performed, for each submatrix there remains a Schur complement matrix \mathbf{F}_l . The variables that remain are called interface variables and the interface matrix \mathbf{F} is formed by summing the matrices \mathbf{F}_l . Gaussian elimination is used to factorize \mathbf{F} , using the HSL sparse direct solver MA48. Block forward elimination and back substitution completes the solution.

The user's matrix data may optionally be held in unformatted sequential files. In addition, \mathbf{L} and \mathbf{U} factors for the submatrices may optionally be written to sequential files. This reduces main memory requirements when the number N of submatrices is greater than the number of processes used.

The HSL package HSL_MC66 may be used for preordering the matrix to singly-bordered block-diagonal form.

ATTRIBUTES — **Version:** 2.0.0. **Types:** Real (single, double). **Calls:** HSL_MP01, KB08, MA48, MA52, MC46 and the BLAS routines I_AMAX, _AXPY, _SCAL, _SWAP, _GEMV, _TPSV, _GEMM, _TRSM, _TRSV. **Remark:** HSL_MC66 may be used for preordering. **Language:** Fortran 95 + TR 15581 (allocatable components). **Date:** March 2003. **Origin:** I.S. Duff and J.A. Scott, Rutherford Appleton Laboratory.

HSL_MP54 Parallel Cholesky solver

For a matrix that is full, symmetric and positive definite, this package performs parallel partial and complete factorisations and solutions of corresponding sets of equations, using OpenMP.

We consider the factorization

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{21}^T \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{pmatrix} = \begin{pmatrix} \mathbf{L}_{11} & \\ & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{I} & \\ & \mathbf{S}_{22} \end{pmatrix} \begin{pmatrix} \mathbf{L}_{11}^T & \mathbf{L}_{21}^T \\ & \mathbf{I} \end{pmatrix} = \mathbf{LSL}^T$$

where \mathbf{A} is order n , \mathbf{L}_{11} is lower triangular and both \mathbf{A}_{11} and \mathbf{L}_{11} have order $p \leq n$.

Subroutines are also provided for the complementary partial forward and backward substitutions, that is, solving

$$\mathbf{LX} = \mathbf{B} \quad \text{and} \quad \mathbf{L}^T \mathbf{X} = \mathbf{B}.$$

ATTRIBUTES — **Version:** 1.0.0.. **Types:** Real (single, double). **Calls:** _AXPY, _COPY, _GEMM, _SYR, _SYRK, _TPSV, _TRSM **Date:** August 2008. **Origin:** J. D. Hogg, Rutherford Appleton Laboratory. **Language:** Fortran 95, plus allocatable components of derived types. **Parallelism:** Uses OpenMP and its runtime library. **Remark:** Development of HSL_MP54 was supported by EPSRC grant EP/F006535/1.

HSL_MP62 Symmetric finite-element system: multiple-front method

The module HSL_MP62 uses the multiple front method to solve sets of symmetric positive-definite finite-element equations $\mathbf{AX} = \mathbf{B}$ that have been divided into non-overlapping subdomains. The HSL routines MA62 and MA72 are used with MPI for message passing.

The coefficient matrix \mathbf{A} must be of the form

$$\mathbf{A} = \sum_{k=1}^m \mathbf{A}^{(k)}$$

where the summation is over finite elements. The element matrix $\mathbf{A}^{(k)}$ is nonzero only in those rows and columns which correspond to variables in the k -th element. The right-hand side(s) \mathbf{B} may optionally be in the form

$$\mathbf{B} = \sum_{k=1}^m \mathbf{B}^{(k)}$$

where $\mathbf{B}^{(k)}$ is nonzero only in those rows which correspond to variables in element k .

In the multiple front method, a frontal decomposition is performed on each subdomain separately. Thus, on each subdomain, L and U factors are computed. Once all possible eliminations have performed within a subdomain, there remain the interface variables, which are shared by more than one subdomain. If \mathbf{F}_i is the remaining frontal matrix for subdomain i , and \mathbf{C}_i is the corresponding right-hand side matrix, then the remaining problem is

$$\mathbf{F}\mathbf{Y} = \mathbf{C}, \tag{1.4}$$

where $\mathbf{F} = \sum_i \mathbf{F}_i$ and $\mathbf{C} = \sum_i \mathbf{C}_i$. By treating each \mathbf{F}_i as an element matrix, the interface problem (1.4) is also solved by the frontal method. Once (1.4) has been solved, back-substitution on the subdomains completes the solution.

The element data and/or the matrix factors are optionally held in direct-access files.

ATTRIBUTES — **Version:** 2.0.0. **Types:** Real (single, double). **Calls:** HSL_MP01, KB08, MA62, MA72, MC53, MC63. **Language:** Fortran 95 + TR 15581 (allocatable components). **Date:** September 1999. **Origin:** J.A. Scott, Rutherford Appleton Laboratory.

O - INPUT/OUTPUT

OF File management

HSL_OF01 Fortran virtual memory

This package provides read/write facilities for one or more direct-access files through a single in-core buffer, so that actual input-output operations are often avoided. The buffer is divided into fixed-length *pages* and all input-output is performed by transferring a single page to or from a single record of a file (the length of a record is equal to the length of a page).

Each set of data is addressed as a virtual array, that is, as if it were a very large array. The lower bound of the virtual array is 1. Each element of the virtual array has initial value zero. Any contiguous section of the virtual array may be read or written, without regard to page boundaries.

The virtual array is permitted to be too large to be accommodated in a single file, in which case HSL_OF01 opens secondary files with names that it constructs from the name of the primary file by appending '1', '2', We refer to the set of files as a **superfile**. Each superfile is identified by the name of its primary file or the index that it is given when it is opened. To allow the secondary files to reside on different devices, the user is required to supply an array of path names; the full name of a file is the concatenation of a path name with the file name.

To facilitate finite-element assembly and the multifrontal method, there is an option to add data from the virtual array to a given array under the control of a map.

ATTRIBUTES — **Version:** 3.2.0. **Types:** Real (single, double), Complex (single, double), Integer. **Calls:** _COPY. **Language:** Fortran 95 + TR 15581 (allocatable components). **Date:** October 2006. **Origin:** J.K. Reid and J.A. Scott, Rutherford Appleton Laboratory.

P - POLYNOMIAL AND RATIONAL FUNCTIONS

PA Zeros of polynomials

PA16 Complex coefficients: all roots by the method of Madsen and Reid

To find all the real and complex roots of a polynomial with complex coefficients, i.e. calculate the zeros of

$$a_0 + a_1x + a_2x^2 + \dots + a_nx^n = 0$$

The user can supply error bounds on the coefficients of the polynomial and the routine returns bounds on the moduli of the errors in the roots. The roots are found by the method of Madsen and error bounds by the application of Rouché's theorem.

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double). **Precision:** At least 8-byte arithmetic is recommended. **Calls:** FD15. **Language:** Fortran 77. **Date:** August 1990. **Origin:** J.K. Reid, Rutherford Appleton Laboratory.

PA17 Real coefficients: all roots by the method of Madsen and Reid

To find all the real and complex roots of a polynomial with real coefficients, i.e. calculate the zeros of

$$a_0 + a_1x + a_2x^2 + \dots + a_nx^n = 0$$

The user can supply error bounds on the coefficients of the polynomial and the routine returns bounds on the moduli of the errors in the roots. The roots are found by the method of Madsen and error bounds by the application of Rouché's theorem.

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double). **Precision:** At least 8-byte arithmetic is recommended. **Calls:** FD15. **Language:** Fortran 77. **Date:** August 1990. **Origin:** J.K. Reid, Rutherford Appleton Laboratory.

Y - TEST PROGRAM GENERATORS

YM Generate test programs for chapter M of the library

YM11 Generate a random sparse matrix

These subroutines generate an m by n random sparse matrix with user-specified options such as structural nonsingularity and bandedness. The matrix is held in a packed form in a standard sparse matrix format, and there is an option to write it to a file in Rutherford Boeing format (Report RAL-TR-97-031).

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double), Complex (single, double), Integer. **Calls:** FA14, MC56, MC59. **Date:** 1987, revised November 2001.

Remark: YM11 is a threadsafe version of YM01 and includes entries for generating complex valued and integer valued matrices. **Origin:** I.S. Duff, Rutherford Appleton Laboratory.

Z - FORTRAN SYSTEM FACILITIES

ZB Array allocation

HSL_ZB01 Reallocate an array

Given a rank-one or rank-two allocatable array, HSL_ZB01 reallocates the array to have a different size, and can copy all or part of the original array into the new array. This will use a temporary array or, if there is insufficient memory, one or more temporary files will be used. The user may optionally force HSL_ZB01 to only use temporary files and not to attempt to use a temporary array. The user may also optionally supply the name of the temporary file and the filesize; if no name is supplied, then a scratch file will be used. If more than one file is required and a filename has been supplied, then HSL_ZB01 opens files with names that it constructs from the filename by appending '1', '2',... to the end. All temporary files are deleted upon successful exit. If the array given as input was not already allocated, then HSL_ZB01 allocates the array to have the desired size.

ATTRIBUTES — **Version:** 2.0.0. **Types:** Real (single, double), Complex (single, double), Integer (default, long). **Calls:** none. **Language:** Fortran 95 + TR 15581 (allocatable components). **Date:** July 2007. **Origin:** H. S. Dollar, Rutherford Appleton Laboratory. **Remark:** The development of this package was supported by EPSRC grant GR/S42170.

ZD Derived types

HSL_ZD11 Derived type for sparse matrix storage schemes

This package defines a derived type capable of supporting a variety of sparse matrix storage schemes. Its principal use is to allow exchange of data between HSL subprograms and other codes.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double), Integer, Complex (single, double). **Calls:** None. **Language:** Fortran 95 + TR 15581 (allocatable components). **Date:** February 2006. **Origin:** N.I.M. Gould and J.K. Reid, Rutherford Appleton Laboratory.

Index

- algebraic multigrid, HSL_MI20
- allocatable array
 - reallocate an array, HSL_ZB01
- analyse phase
 - sparse symmetric, HSL_MC78
- approximate minimum degree ordering, MC47, HSL_MC68
- approximate-inverse preconditioner, MI12
- Arnoldi's method, EB13
- assemble finite-element matrix, MC57
- augmented system, MA69, HSL_MA69

- backward error estimate based on iterative refinement, MA60
- balancing a matrix, *see* scaling factors for matrices
- banded matrix
 - complex unsymmetric
 - row-by-row frontal, ME42, ME43
 - real symmetric positive definite, HSL_MA55
 - real unsymmetric, MA65
 - row-by-row frontal, MA43, HSL_MP43
- bandwidth reduction, MC60, MC61, HSL_MC73
- block triangular form, *see* permutation
- bordered block triangular form, MC33

- Cholesky factorization
 - analyse phase, HSL_MC78
 - banded, HSL_MA55
 - complex sparse matrix, HSL_MA87
 - real dense matrix, HSL_MA54, HSL_MP54
 - real sparse matrix, MA62, HSL_MA77, HSL_MA87, HSL_MP62
- condition number estimate
 - 1-norm or ∞ -norm
 - complex matrix, MF71
 - real matrix, MC71
 - classical, MC75
 - Skeel's, MC75
- conjugate gradients
 - preconditioned, MA61
 - saddle-point systems, HSL_MI27
- covariance matrix, MA75
- Cuthill-McKee, MC60, MC61

- data fitting
 - linear, MA44
- Dulmage-Mendelsohn decomposition, HSL_MC79

- eigenvalues
 - real symmetric sparse

- generalised eigenvalue problem, EA16, HSL_EA19
 - Lanczos, EA16, EA25, EP25
 - subspace iteration, HSL_EA19
- eigenvectors, *see* eigenvalues
- element ordering, MC63
- equilibration, *see* scaling factors for matrices
- error estimates
 - solutions of linear equations, MA60
 - zeros of polynomials, PA16, PA17
- estimate
 - condition number, *see* condition number estimate
 - error in the solution of linear equations, MA60
- factorization
 - symbolic, HSL_MC78
- factorization, complex sparse matrix
 - Hermitian, HSL_MA86, HSL_MA87, HSL_ME57, ME57
 - rectangular, ME48, ME50
 - symmetric, HSL_MA86, HSL_ME57, ME57
 - unsymmetric, HSL_MA42_ELEMENT, ME38, ME42, ME43, ME48, ME50
- factorization, dense matrix
 - complex, HSL_MA64
 - real symmetric indefinite, HSL_MA64
 - real symmetric positive definite, HSL_MA54, HSL_MP54
- factorization, incomplete
 - real symmetric positive definite, MA61
 - real unsymmetric, MI11
- factorization, partial
 - complex, HSL_MA64
 - real symmetric indefinite, HSL_MA64
 - real symmetric positive definite, HSL_MA54
 - real unsymmetric, HSL_MA74
- factorization, real sparse matrix
 - approximate minimum degree ordering, MC47, HSL_MC68
 - nested bisection ordering, HSL_MC68
 - rectangular, HSL_MA48, MA48, MA50, MA51
 - find rank, MC58
 - symmetric, HSL_MA87
 - banded, HSL_MA55
 - multifrontal, HSL_MA57, MA57, HSL_MA77
 - multifrontal out of core, HSL_MA77
 - multiple front, MA72, HSL_MP62
 - zeros on diagonal, MA67
 - symmetric frontal, MA62
 - symmetric indefinite, HSL_MA57, MA57, HSL_MA77
 - multicore, HSL_MA86
 - unsymmetric, HSL_MA48, MA48, MA50, MA51
 - banded, MA65
 - finite element, MA42, HSL_MA42_ELEMENT, HSL_MA42, MA46, HSL_MA78
 - multifrontal, MA38, MA41
 - multiple front, MC77, HSL_MP42, HSL_MP43
 - parallel, HSL_MP42, HSL_MP43, HSL_MP48
 - row-by-row frontal, MA43
 - update following rank-one change, LA15, MA69, HSL_MA69
- Fiedler vector, HSL_MC73
- file management, HSL_OF01
- finite elements

- analyse phase, HSL_MC78
- assemble matrix, MC57
- assembly order for frontal method, MC63
- assembly order within subdomain, MC53
- complex
 - Hermitian frontal, ME62
 - symmetric frontal, ME62
 - unsymmetric frontal, HSL_MA42_ELEMENT, ME42
- find connectivity graph, MC44
- find supervariables, MC44, HSL_MC78
- real symmetric
 - frontal, MA62
 - multifrontal, HSL_MA77
 - multiple front, MA72, HSL_MP62
- real unsymmetric
 - frontal, MA42, HSL_MA42_ELEMENT, HSL_MA42
 - multifrontal, MA46, HSL_MA78
 - multiple front, MA52
- finite-element matrix
 - represent as, MC37
- fractional power sparse self-adjoint positive-definite pencils, EA20
- frontal method
 - complex
 - Hermitian, ME62
 - symmetric, ME62
 - unsymmetric, HSL_MA42_ELEMENT, ME42
 - unsymmetric row-by-row, ME43
 - element assembly order, MC63
 - ordering for row-by-row, MC62
 - real
 - parallel, HSL_MP42, HSL_MP43, HSL_MP62
 - symmetric, MA62
 - unsymmetric, MA42, HSL_MA42_ELEMENT, HSL_MA42
 - unsymmetric row-by-row, MA43
- Gaussian elimination, *see* factorization, matrix
- generalised eigenvalue problem, EA16, EA20
- GMRES method, MI15
- graph
 - column connectivity, HSL_MC65
 - element connectivity, MC44
 - row connectivity, HSL_MC65
 - supervariable connectivity, MC44, HSL_MC78
- Hager's exchange algorithms, HSL_MC67
- heapsort, KB22
- Householder reduction
 - over-determined systems, MA44, MA49
- HSL standard sparse matrix format, HSL_MC69
- incomplete factorization
 - symmetric, MA61
 - unsymmetric, MI11
- iterative method
 - with preconditioning, MI02, HSL_MI20, MI15, MI21, MI23, MI24, MI25, MI26, HSL_MI31
- iterative refinement, MA60

- kernel linear solvers, HSL_MA54, HSL_MA64, HSL_MA74
- Lanczos algorithm, EA16, HSL_EA19, EA20
- least squares
 - dense linear equality constraints, MA44
 - sparse
 - linear, MA49
 - weighted, MA75
- linear equations, complex sparse
 - Hermitian
 - multifrontal, HSL_ME57, ME57
 - symmetric
 - frontal, ME62
 - multicore, HSL_MA86, HSL_MA87
 - multifrontal, HSL_ME57, ME57
 - unsymmetric, ME48, ME50
 - multifrontal, ME38
 - row-by-row frontal, ME43
 - unsymmetric frontal, HSL_MA42_ELEMENT, ME42
- linear equations, real dense
 - factorization, HSL_MA54, HSL_MA64, HSL_MA74
 - least squares, MA44
- linear equations, real sparse
 - augmented system, MA69, HSL_MA69
 - least squares, MA49
 - symmetric
 - banded, HSL_MA55
 - iterative, MI02, MI15, MI21, HSL_MI31
 - multicore, HSL_MA86, HSL_MA87
 - multifrontal, HSL_MA57, MA57, HSL_MA77
 - multifrontal out of core, HSL_MA77
 - zeros on diagonal, MA67
 - unsymmetric, LA15, HSL_MA48, MA48, MA50, MA51
 - banded, MA65
 - iterative, MI11, HSL_MI20, MI23, MI24, MI25, MI26
 - multifrontal, MA41, MA46
 - multiple front, MA52, HSL_MP42, HSL_MP43
 - preconditioning, MI12
 - row-by-row frontal, MA43
 - unsymmetric frontal, MA42, HSL_MA42_ELEMENT, HSL_MA42
 - weighted least squares, MA75
- linear equations, saddle-point systems
 - iterative, HSL_MI27
 - preconditioning, HSL_MI13
- linear least squares, *see* least squares
- linear programming
 - revised simplex method, LA04
 - update basis, LA15
- matrix format conversion to CSC, HSL_MC69
- matrix multiplication
 - sparse, HSL_MC65
- maximum matching, HSL_MC79
- maximum transversal, MC21
- minimum degree ordering
 - approximate, MC47, HSL_MC68
- mixed precision

- sparse symmetric system, HSL_MA79
- MONET algorithm, HSL_MC66
- MPI, EP25, HSL_MP42, HSL_MP43, HSL_MP48, HSL_MP62
- multifrontal method
 - symmetric, HSL_MA57, MA57, HSL_ME57, ME57
 - finite elements, MA46, HSL_MA77, HSL_MA78
 - unsymmetric, MA38, MA41, ME38
- multiple front method
 - ordering, MC53, HSL_MC66
 - real symmetric, MA72
 - parallel, HSL_MP62
 - real unsymmetric, MA52
 - parallel, HSL_MP42, HSL_MP43
- nonlinear least squares, *see* least squares
- norm
 - estimate of 1-norm or ∞ -norm, MC71, MF71
- normal equations, MA75
- OpenMP, HSL_MA86, HSL_MA87, HSL_MP54
- ordering
 - approximate minimum degree, HSL_MC68
 - finite-element matrices
 - frontal method, MC63
 - multiple front, MC53
 - nested bisection, HSL_MC68
 - numbers
 - ascending order, KB05, KB22
 - ascending order with index array, KB07
 - descending order, KB06
 - descending order with index array, KB08
 - profile and wavefront reduction, MC60, MC61, HSL_MC67, HSL_MC73
 - row-by-row frontal, MC62
 - singly bordered block diagonal form, HSL_MC66
 - spectral, HSL_MC73
- out-of-core factorization
 - complex
 - symmetric frontal, ME62
 - unsymmetric frontal, HSL_MA42_ELEMENT, ME42
 - real
 - symmetric frontal, MA62
 - symmetric multifrontal, HSL_MA77
 - unsymmetric frontal, MA42, HSL_MA42_ELEMENT, HSL_MA42
 - unsymmetric multifrontal, HSL_MA78
- over-determined system
 - dense least squares, MA44
 - sparse, MA49
 - weighted least squares, MA75
- parallel
 - eigenvalues, EP25
 - solution of linear equations
 - dense symmetric, HSL_MP54
 - sparse symmetric, HSL_MA86, HSL_MA87, HSL_MP62
 - sparse unsymmetric, HSL_MP42, HSL_MP43, HSL_MP48
- permutations
 - block triangular form
 - symmetric matrix, MC13

- unsymmetric matrix, MC25
 - row, to force diagonal nonzeros for real matrix, MC21
 - symmetric
 - to reduce profile, MC60, MC61, HSL_MC67, HSL_MC73
 - to place large entries on the diagonal, MC64, HSL_MC64
 - unsymmetric, MC22
 - unsymmetric, to block triangular form
 - complex matrix, ME22
- preconditioning
 - algebraic multigrid, MI20
 - approximate-inverse, MI12
 - incomplete factorization
 - symmetric, MA61
 - unsymmetric, MI11
 - saddle-point systems, HSL_MI13
- profile reduction, MC60, MC61, HSL_MC67
- pseudo-random numbers, *see* random numbers
- Quicksort
 - numbers
 - ascending order, KB05
 - ascending order with index array, KB07
 - descending order, KB06
 - descending order with index array, KB08
- random matrix, YM11
- rank, real sparse matrix
 - unsymmetric, MC58
- Reverse Cuthill-McKee, MC60, MC61
- roots
 - polynomial (complex coefficients), PA16
 - polynomial (real coefficients), PA17
- Rouche's theorem, PA16, PA17
- row ordering
 - symmetric, MC60, MC61, HSL_MC67, HSL_MC73
 - unsymmetric, MC62
- Rutherford-Boeing format, HSL_MC56, MC54, MC55, YM11
- saddle-point systems
 - iterative, HSL_MI27
 - preconditioner, HSL_MI13
- scaling factors for matrices
 - complex
 - dense unsymmetric, MC77
 - sparse Hermitian, MF30
 - sparse symmetric, MC77, MF30
 - sparse unsymmetric, MC77, MF29, MF64
 - real
 - dense unsymmetric, MC72, MC77
 - sparse rectangular, HSL_MC64
 - sparse symmetric, MC30, HSL_MC64, MC77
 - sparse unsymmetric, MC29, MC64, HSL_MC64, MC77
- Schur complement, MA69, HSL_MA69
- simplex method, LA04
- simultaneous iteration
 - symmetric matrix, EA22
 - unsymmetric matrix, EB22
- solution of linear equations, *see* linear equations

- sorting
 - nonzeros of sparse matrix, MC59
- sparse
 - eigenvalue problems, *see* eigenvalues
 - least squares, *see* least squares
 - linear equations, *see* linear equations
 - over-determined system, *see* over-determined system
- sparse matrix
 - basic operations, HSL_MC65
 - create structure for $A^T A$, MC26
 - expand structure, HSL_MC34, MC34
 - factorization, *see* factorization, matrix
 - HSL standard sparse matrix format, HSL_MC69
 - scaling factors, *see* scaling factors for matrices
 - sorting nonzeros, *see* sorting
 - test matrices, *see* Rutherford-Boeing format
 - transpose, *see* transpose sparse matrix
- spectral ordering, HSL_MC73
- subspace iteration
 - symmetric matrix, HSL_EA19, EA22
 - unsymmetric matrix, EB22
- SYMMBK method, MI02
- symmetrize sparse matrix, HSL_MC65

- Tarjan's method, MC13
- tearing, MC33
- transpose sparse matrix, MC38, MC46, HSL_MC65

- virtual memory, HSL_OF01

- wavefront reduction, MC60, MC61, MC63, HSL_MC73
- weighted least squares, MA75

- zeros of
 - polynomial (complex coefficients), PA16
 - polynomial (real coefficients), PA17