

HSL 2007

A CATALOGUE OF SUBROUTINES

HSL 2007 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 2007 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 2007 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

CONTENTS

1 INTRODUCTION	9
1.1 HSL 2007 and HSL Archive	9
1.1.1 Free use of HSL 2007 by UK academics	9
1.1.2 Free non-commercial use of HSL Archive by anyone.....	9
1.1.3 Design and language of HSL	9
1.1.4 Special versions of packages	10
1.1.5 HSL Archive packages called from HSL 2007	10
1.1.6 Use of BLAS by HSL	10
1.1.7 Use of LINPACK and LAPACK by HSL	10
1.1.8 Use of MeTiS by HSL	10
1.1.9 Use of MPI by HSL	11
1.1.10 Use of reverse communication by HSL	11
1.1.11 Changes in HSL 2007	11
1.2 Using HSL facilities	13
1.2.1 Queries service	13
1.2.2 Classifications	13
1.2.3 Naming convention	13
1.2.4 Marking of versions	14
1.2.5 Finding the package to do the job	14
1.2.6 Specification documents	14
1.3 The catalogue	14
1.4 Licensing arrangements	15
1.4.1 Conditions of using HSL 2007.....	15
1.4.2 Use of HSL 2007 as a part of Application Software	16
1.5 Use of HSL 2007 by UK academics	16
1.5.1 Acknowledging the use of HSL 2007	16
1.5.2 UK academic free-licence conditions	16
1.5.3 Do you have a valid licence?	18
1.6 Matlab interfaces for selected HSL packages	18
A COMPUTER ALGEBRA	19
AD Automatic differentiation.....	19
HSL_AD02 Automatic differentiation	19
D DIFFERENTIAL EQUATIONS.....	20
DC Initial-value ordinary differential equation problems	20
HSL_DC05 Ordinary differential equations or differential algebraic equations of index 1	20
DD Two-point boundary value ordinary differential equation problems	21
DD14 Two-point boundary-value ordinary differential equation	21

E EIGENVALUES AND EIGENVECTORS	22
EA Eigenvalues and eigenvectors of real symmetric matrices	22
EA16 Compute selected eigenpairs using rational Lanczos method	22
HSL_EA19 Sparse symmetric or Hermitian: leftmost eigenpairs	23
EA22 Sparse symmetric: simultaneous iteration	24
EA23 Full symmetric: Jacobi's method	24
EA25 Sparse symmetric: Lanczos for the spectrum	24
EB Eigenvalues and eigenvectors of real general matrices	25
EB13 Sparse unsymmetric: Arnoldi's method	25
EB22 Sparse unsymmetric: subspace iteration	25
EC Eigenvalues and eigenvectors of Hermitian matrices	25
EC23 Full Hermitian matrix: classical Jacobi's method	25
EP Packages dependent on MPI	26
EP25 Sparse symmetric: Lanczos for the spectrum	26
F MATHEMATICAL FUNCTIONS	27
FA Random numbers	27
FA14 Uniform distribution	27
HSL_FA14 Uniform distribution	27
FD Simple functions	28
FD15 Real-valued machine constants	28
K SORTING	29
KB Sorting numbers	29
KB05 Sort numbers into ascending order using Quicksort	29
KB06 Sort numbers into descending order using Quicksort	29
KB07 Sort numbers into ascending order with indexing using Quicksort...	29
KB08 Sort numbers into descending order with indexing using Quicksort	29
HSL_KB22 Sorting reals using the Heapsort method	29
L LINEAR PROGRAMMING	31
LA Linear programming, i.e. minimization of a linear function subject to linear constraints	31
LA04 Sparse linear programming: steepest-edge simplex method	31
LA15 Sparse mathematical programming bases: factorize and update	31
M LINEAR ALGEBRA	32
MA Solution of linear systems	32
MA38 Sparse unsymmetric system: unsymmetric multifrontal method	32
MA41 Sparse unsymmetric system: unsymmetric multifrontal method	32
MA42 Sparse unsymmetric system: out-of-core frontal method	33
HSL_MA42 Sparse unsymmetric system: out-of-core frontal method	33
HSL_MA42_ELEMENT Unsymmetric finite-element system: out-of-core frontal method (real and complex)	34
MA43 Sparse unsymmetric system: row-by-row frontal method	34
MA44 Over-determined linear system: least-squares solution	34
MA46 Sparse unsymmetric finite-element system: multifrontal	35

MA48 Sparse unsymmetric system: driver for conventional direct method	36
HSL_MA48 Sparse unsymmetric system: driver for conventional direct method	36
MA49 Sparse over-determined system: least squares by QR	36
MA50 Sparse unsymmetric system: conventional direct method	37
MA51 Auxiliary for MA48 and MA50: identify ignored rows or columns in the rectangular or rank-deficient case, compute determinant	37
MA52 Sparse unsymmetric finite-element system: out-of-core multiple front method	38
HSL_MA54 Kernel code for HSL_MA77 (definite case)	38
HSL_MA55 Band symmetric positive-definite system	39
MA57 Sparse symmetric system: multifrontal method	40
HSL_MA57 Sparse symmetric system: multifrontal method	40
MA60 Iterative refinement and error estimation	40
MA61 Sparse symmetric positive-definite system: incomplete factorization	41
MA62 Sparse symmetric finite-element system: out-of-core frontal method	41
HSL_MA64 Kernel code for HSL_MA77 (indefinite case)	42
MA65 Unsymmetric banded system of linear equations	43
MA67 Sparse symmetric system, zeros on diagonal: blocked conventional	43
MA69 Unsymmetric system whose leading subsystem is easy to solve	43
HSL_MA69 Unsymmetric system whose leading subsystem is easy to solve	44
MA72 Sparse symmetric finite-element system: out-of-core multiple front method	44
HSL_MA74 Kernel code for HSL_MA78	45
MA75 Sparse over-determined system: weighted least squares	46
HSL_MA77 Sparse symmetric system: multifrontal out of core	46
HSL_MA78 Sparse unsymmetric system: multifrontal out of core	48
MC Computations with matrices and vectors	49
MC13 Permute a sparse matrix to block triangular form	49
MC21 Permute a sparse matrix to put entries on the diagonal	49
MC22 Permute a sparse matrix given row and column permutations	49
MC25 Permute a sparse matrix to block triangular form	50
MC26 Sparse rectangular matrix: compute normal matrix	50
MC29 Sparse unsymmetric matrix: calculate scaling factors	50
MC30 Sparse symmetric matrix: calculate scaling factors	50
MC33 Sparse unsymmetric matrix: permute to bordered block triangular form	51
MC34 Sparse symmetric structure: expand from lower triangle	51
MC37 Sparse symmetric matrix: represent as a sum of element matrices ..	51
MC38 Sparse rectangular matrix held by columns: transpose	51
MC44 Unassembled finite-element matrix: generate the element or supervariable connectivity graph	51
MC46 Sparse rectangular matrix held by rows: transpose	52
MC47 Sparse symmetric pattern: approximate minimum-degree ordering allowing dense rows	52

MC53	Generate an ordering for finite-element matrices within a subdomain.....	53
MC54	Write a sparse matrix in Rutherford-Boeing format	53
MC55	Write a supplementary file in Rutherford-Boeing format	53
MC56	Read a file or a supplementary file held in Rutherford-Boeing format	53
MC57	Assemble a set of finite-element matrices	54
MC58	Estimate rank and find independent rows/columns of a sparse unsymmetric or rectangular matrix	54
MC59	Sort a sparse matrix to an ordering by columns	54
MC60	Sparse symmetric pattern: reduce the profile and wavefront	55
MC61	Straightforward interface to MC60	55
MC62	Generate a row ordering for a row-by-row frontal solver	55
MC63	Generate an element assembly ordering for a frontal solver	56
MC64	Permute and scale a sparse unsymmetric matrix to put large entries on the diagonal	56
HSL_MC64	Permute and scale a sparse unsymmetric or rectangular matrix to put large entries on the diagonal	56
HSL_MC65	Construct and manipulate matrices in compressed sparse row format	57
HSL_MC66	Permute an unsymmetric sparse matrix to singly bordered blocked diagonal form	58
MC67	Refine a profile-reducing permutation of a symmetric matrix	58
HSL_MC68	Symmetric sparse matrix: compute elimination orderings	59
MC71	Unsymmetric matrix: estimate 1-norm.....	59
MC72	Full unsymmetric matrix: calculate scaling factors	59
HSL_MC73	Sparse symmetric matrix: compute Fiedler vector and permute to reduce the profile and wavefront	60
MC75	Sparse unsymmetric matrix: estimate condition number	60
MC77	Sparse unsymmetric matrix: calculate scaling factors	60
ME	Solution of complex linear systems and other calculations for complex matrices	61
ME22	Permute a sparse matrix given row and column permutations	61
ME38	Sparse unsymmetric system: unsymmetric multifrontal method	61
ME42	Sparse unsymmetric system: out-of-core frontal method	61
ME43	Sparse unsymmetric system: row-by-row frontal method	62
ME48	Sparse unsymmetric system: driver for conventional direct method	62
ME50	Sparse unsymmetric system: conventional direct method	62
ME57	Sparse Hermitian or complex symmetric: multifrontal method	63
ME62	Sparse Hermitian or complex symmetric finite-element system: out-of-core frontal method	63
MF	Computations with complex matrices and vectors	64
MF29	Sparse unsymmetric matrix: calculate scaling factors	64
MF30	Sparse symmetric matrix: calculate scaling factors	64
MF64	Permute and scale a sparse complex unsymmetric matrix to put large entries on the diagonal.....	65
MF71	Unsymmetric matrix: estimate 1-norm.....	65
MI	Iterative methods for sparse linear systems.....	65
HSL_MI02	Symmetric possibly-indefinite system: SYMMBK method	65

MI11	Unsymmetric system: incomplete LU factorization.....	66
MI12	Unsymmetric system: approximate-inverse preconditioner.....	66
HSL_MI13	Preconditioners for saddle point systems	67
MI15	Unsymmetric system: flexible GMRES	67
HSL_MI20	Unsymmetric system: algebraic multigrid	68
MI21	Symmetric positive-definite system: conjugate gradient method ...	68
MI23	Unsymmetric system: CGS (conjugate gradient squared) method ...	69
MI24	Unsymmetric system: GMRES (generalized minimal residual) method	69
MI25	Unsymmetric system: BiCG (BiConjugate Gradient) method.....	69
MI26	Unsymmetric system: BiCGStab (BiConjugate Gradient Stabilized) method	70
HSL_MI31	Symmetric positive-definite system: conjugate gradient method, stopping according to the A-norm of the error	70
MP	Packages dependent on MPI	71
HSL_MP42	Unsymmetric finite-element system: multiple-front method, element entry	71
HSL_MP43	Sparse unsymmetric system: multiple-front method, equation entry	71
HSL_MP48	Sparse unsymmetric system: parallel direct method	72
HSL_MP62	Symmetric finite-element system: multiple-front method	73
N	NONLINEAR EQUATIONS	74
NS	Solution of systems of nonlinear equations in several unknowns	74
NS12	Sparse nonlinear equations: Powell dog-leg algorithm	74
NS23	Sparse nonlinear over-determined equations: Marquardt method ...	74
O	INPUT/OUTPUT	76
OF	File management	76
HSL_OF01	Fortran virtual memory	76
P	POLYNOMIAL AND RATIONAL FUNCTIONS	77
PA	Zeros of polynomials	77
PA16	Complex coefficients: all roots by the method of Madsen and Reid	77
PA17	Real coefficients: all roots by the method of Madsen and Reid	77
T	INTERPOLATION AND APPROXIMATION	78
TD	Estimation of derivatives by finite differences	78
TD22	Approximate Jacobian matrix using finite differences	78
V	OPTIMIZATION AND NONLINEAR DATA FITTING	79
VA	Minimization of general functions and sums of squares of functions of several variables	79
VA08	Minimize a function of several variables: Fletcher-Reeves method	79
VA34	Minimize a function of a huge number of variables: conjugate gradients	79
VA35	Minimize a function of many variables: limited-memory BFGS method	79

VE	Minimization of a general function subject to linear constraints	80
VE08	Minimize a sum of finite-element functions	80
VE10	Minimize a sum of squares of element functions	81
HSL_VE12	Quadratic programming problem: interior-point trust-region method	82
HSL_VE13	Constrained least distance problem: interior-point trust-region method	82
HSL_VE15	Quadratic programming problem: reorder the problem	83
HSL_VE19	Quadratic programming problem: working-set method	84
VE24	Quadratic programming problem within a region defined by simple bounds	85
VF	Minimization of a general function subject to nonlinear constraints	85
HSL_VF05	Approximate minimization of a sparse quadratic function with norm constraint.....	85
HSL_VF06	Global minimization of a sparse quadratic function with norm constraint	85
VH	Minimization of functions of integer variables	86
HSL_VH01	Genetic algorithm for the smallest value of a function of binary variables.....	86
Y	TEST PROGRAM GENERATORS	87
YM	Generate test programs for chapter M of the library	87
YM11	Generate a random sparse matrix	87
Z	FORTRAN SYSTEM FACILITIES	88
ZA	Timing, machine constants, etc	88
HSL_ZA03	Kind values for 1- and 2-byte LOGICAL variables	88
ZA12	CPU time	88
ZB	Array allocation	88
HSL_ZB01	Reallocate an array	88
ZD	Derived types	89
HSL_ZD02	Derived type for quadratic programming	89
HSL_ZD11	Derived type for sparse matrix storage schemes	89
INDEX	91

1 INTRODUCTION

1.1 HSL 2007 and HSL Archive

HSL (formerly the Harwell Subroutine Library) is a collection of Fortran packages for large-scale scientific computation written and developed by the Numerical Analysis Group at the Rutherford Appleton Laboratory and by other experts and collaborators. The Library was started back 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. HSL is used by more than 2000 organisations world-wide and HSL packages have been incorporated under licence into more than 100 commercially available software products.

Many of the older HSL packages have gradually been superseded by newer versions, with increases in functionality, improved interfaces, or speed of execution. As a result, HSL is now arranged in two parts, the main library, HSL 2007; 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 2007; there is a comparable catalogue for the HSL Archive.

1.1.1 Free use of HSL 2007 by UK academics

While HSL 2007 is a commercial product, it is also available without charge to UK academics for teaching and academic research purposes. This innovation is a direct result of much of the core funding for the Numerical Analysis Group at the Rutherford Appleton Laboratory being provided by a grant from the Engineering and Physical Science Research Council (GR/S42170). Details of how UK academics can obtain HSL 2007 are given in Section 1.5.

1.1.2 Free non-commercial use of HSL Archive by anyone

The HSL Archive may be licensed without charge to anyone (in the UK and elsewhere), so long as the packages are not then supplied to a third party. Access to the HSL Archive is by way of the HSL web page

<http://www.cse.clrc.ac.uk/nag/hsl>

1.1.3 Design and language of HSL

HSL 2007 and the HSL Archive are arranged as collections of 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 and some HSL Archive packages contain block data subprograms to provide default values for variables in common blocks. 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 both as a PostScript file and as a PDF file.

All the HSL Fortran 77 packages adhere to the Fortran 77 Standard, with the exception of a few COMPLEX*16 declarations and corresponding arithmetic operations

and intrinsic procedure calls, and a long name for a call to MeTiS in MA57. Both 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. A few 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.

1.1.4 Special versions of packages

There are three packages, FD15, HSL_ZA03, and ZA12, that may need special code. The FD15 package is widely used in the library and returns machine dependent parameters; the default version adheres to the Fortran 95 standard. The others are much less used in the library. HSL_ZA03 returns kind values for 1- and 2- byte logical variables and will need to be altered if these values are other than 1 and 2. ZA12 returns the cpu time; the default version adheres to the Fortran 95 standard. Machine-specific Fortran 77 versions of FD15 and ZA12 are available.

CRAY users should note that the single precision versions are suitable for them. It is no longer recommended that the double precision versions be used with the compiler option set to disable double precision.

1.1.5 HSL Archive packages called from HSL 2007

Four of the HSL Archive packages (FA01, MA27, MC20, ME20) are referenced from HSL 2007. They are included in the distributed HSL 2007, but are not referenced in this catalogue outside this section. They may be considered to be auxiliary packages; we do not expect the user to need to call them directly.

1.1.6 Use of BLAS by HSL

Many HSL routines make use of the BLAS (Basic Linear Algebra Subprograms). Fortran code for the BLAS is included with the distribution of HSL 2007 but users are advised that, for HSL packages to be efficient, they should always use the 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/>

1.1.7 Use of LINPACK and LAPACK by HSL

Several HSL routines make use of codes from the public domain libraries LAPACK and LINPACK. These are included with the distribution of HSL 2007 and are suitable, but users are strongly encouraged to use optimized versions from the vendor or the public domain.

1.1.8 Use of MeTiS by HSL

Two HSL packages (HSL_MA57 and MA57) offer the option of using the MeTiS ordering library, see

<http://www-users.cs.umn.edu/~karypis/metis/metis/download.html>

and may give the best results by doing so. Included with the distribution of HSL 2007 is a stub subroutine that simply returns an error condition if the option is chosen. If the option is to be invoked, the stub must be replaced by the MeTiS code, which must be acquired separately.

1.1.9 Use of MPI by HSL

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

1.1.10 Use of reverse communication by HSL

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. The following packages are structured in this way: HSL_DC05, EA16, HSL_EA19, EB13, EB22, LA04, MA42, HSL_MA42, HSL_MA42_ELEMENT, MA46, MA52, HSL_MA55, MA60, MA62, MA69, HSL_MA69, MA72, MC53, MC71, ME42, ME62, HSL_MI02, MI15, MI21, MI23, MI24, MI25, MI26, HSL_MI31, NS12, NS23, TD22, VA35, HSL_VF05, and HSL_VH01.

1.1.11 Changes in HSL 2007

The significant new packages in HSL 2007 are:

HSL_EA19 computes the p leftmost eigenvalues and corresponding eigenvectors of a sparse real symmetric (or Hermitian) generalized eigenvalue problem.

HSL_KB22 is a replacement for HSL_KB12 to allow generic calls for sorting of sets of long integers as well as default integers, and default and double-precision reals.

HSL_MA54 is a kernel code for the symmetric positive-definite multifrontal method. It performs partial factorizations and solutions of dense sets of linear equations.

HSL_MA64 (**not available in initial release**) is a kernel code for the symmetric indefinite multifrontal method. It performs partial factorizations and solutions of dense sets of linear equations.

HSL_MA74 is a kernel code for the unsymmetric multifrontal method. It performs partial factorizations and solutions of dense sets of linear equations.

HSL_MA77 (**code for the indefinite case not available in initial release**) solves one or more sets of sparse symmetric equations using an out-of-core multifrontal method. The matrix may be either positive definite or indefinite. Input may be either by rows or by elements.

HSL_MA78 solves one or more sets of sparse unsymmetric equations using an out-of-core multifrontal method. Input may be either by rows or by elements.

MC58 (**not available in initial release**) estimates the rank and finds a nonsingular submatrix of an unsymmetric or a rectangular sparse matrix.

HSL_MC64 permutes and scales a sparse real matrix that may be rectangular to put large entries on the diagonal.

MF64 permutes and scales a sparse complex unsymmetric matrix to put large

entries on the diagonal.

HSL_MC68 computes elimination orderings for a symmetric sparse matrix. It is a driver that provides a uniform front-end to several ordering packages.

HSL_MF64 permutes and scales a sparse complex matrix that may be rectangular to put large entries on the diagonal.

HSL_MI13 constructs reconditioners for saddle point systems.

MI15 uses the ‘Flexible Generalized Minimal Residual’ (Flexible GMRES) iterative method to solve an unsymmetric linear system, optionally using preconditioning.

HSL_MI20 computes a classical algebraic multigrid (AMG) v-cycle preconditioner for an unsymmetric linear system.

HSL_OF01 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.

HSL_ZB01 reallocates an allocatable array to have a different size, and can copy all or part of the original array into the new array.

HSL_ZD11 replaces HSL_ZD01 so that allocatable components are used instead of pointer components.

Significant improvements have been made to these packages:

MA48 now has two additional options:

- (a) requiring an immediate return from the factorize entry if the main arrays are too small, without continuing the decomposition to compute the size necessary, and
- (b) requiring on a JOB=2 call to the factorize entry that if the entries in one of the blocks on the diagonal are unsuitable for the pivot sequence chosen on the previous call, the block is refactorized as on a JOB=1 call.

MA50 now has the option of requiring no attempt to be made to compute a recommended value for LFACT if it is too small.

MA57 now has an option for the automatic selection of an ordering strategy and an option for discarding entries below a threshold. The latter option is particularly useful if the system is very rank deficient and can result in very much faster execution.

HSL_MA57 incorporates the extra facilities offered by MA57.

MC47 now includes the automatic detection and efficient handling of dense or almost dense rows. These can make the old version run very slowly.

HSL_MC65 Pointer arrays replaced by allocatables.

HSL_MC66 Pointer arrays replaced by allocatables.

HSL_MC73 Pointer arrays replaced by allocatables.

ME57 (**improved version not available in initial release**) now includes options for scaling and for calling MeTiS as well as the additional options mentioned for MA57 above.

HSL_VE12 Pointer arrays replaced by allocatables.

HSL_VE13 Pointer arrays replaced by allocatables.

HSL_VE19 Pointer arrays replaced by allocatables.

HSL_VF06 Pointer arrays replaced by allocatables and call of ZA12. removed.

1.2 Using HSL facilities

1.2.1 Queries service Hyprotech UK Limited runs a queries and information service that gives advice and answers questions on HSL matters. Whether the query is administrative (to do with the availability of HSL) or technical (about features of specific packages, for example), the initial contact should be made through the HSL Manager:

HSL Manager,
Hyprotech UK Limited (a subsidiary of AspenTechnology Inc),
C1,
Reading International Business Park,
Basingstoke Road,
Reading,
Berkshire,
RG2 6DT, UK.

Tel: 01189 226405 (UK) +44 1189 226405 (International)

Fax: 01189 226401 (UK) +44 1189 226401 (International)

E-mail: hsl@aspentech.com

1.2.2 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.

1.2.3 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.

C Single precision complex.

Z Double precision complex.

The name of each Fortran 90/95 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>_COMPLEX</code>	Single precision complex.
<code>_DOUBLE_COMPLEX</code>	Double precision complex.

If there are any external procedures, their names follow the naming rules used for HSL codes written in Fortran 77.

1.2.4 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 September 2007).

1.2.5 Finding the package to do the job The first step in choosing a package is normally to consult this catalogue. Its contents list and comprehensive index should be used to search for packages. Each entry summarizes the purpose of a package, with sufficient detail to allow a potential user to decide which of the packages, if any, are likely to be suitable for his or her particular problem. For those holding a free UK academic licence (see Section 1.5), the Numerical Analysis Group at the Rutherford Appleton Laboratory may, if necessary, provide assistance with the choice of package. All other users should use the HSL queries service (see Section 1.2.1) if help on the choice of package is needed.

1.2.6 Specification documents More detailed instructions on how to use an individual package is given in its specification document. Each is available as both a PostScript file and 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.

1.3 The catalogue

This catalogue provides an overview of HSL 2007. It lists the names and purposes of the packages. The entries are listed in alphabetical order using the HSL naming convention (see Section 1.2.3), ignoring `HSL_` in the case of a Fortran 90/95 package. At the end of each entry, certain attributes are listed; these have been presented in a formal manner so as to conserve space:

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 HSL staff. Queries concerning HSL packages should not be directed to the authors personally but should be made through the HSL queries service (see Section 1.2.1).

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

1.4 Licensing arrangements

1.4.1 Conditions of using HSL 2007

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

For anyone working in an academic institution in the UK, a licence for academic research and teaching use of any HSL 2007 package may be obtained without charge through the site

<http://hsl.rl.ac.uk/acuk/hslforacuk.html>

(for terms and conditions, see Section 1.5).

All other users of HSL **must** obtain a licence from the Hyprotech UK Limited (details in Section 1.2.1).

Terms and conditions for non-UK academic users may be found at

<http://hsl.rl.ac.uk/hsl2007/cou/academic.html>

Terms and conditions for all non-academic users may be found at

<http://hsl.rl.ac.uk/hsl2007/cou/commercial.html>

A licence may be obtained without charge for personal use of any package from the HSL Archive. The terms and conditions for such use of the HSL Archive may be found at

<http://hsl.rl.ac.uk/archive/cou.html>

1.4.2 Use of HSL 2007 as a part of Application Software

SUPPLY OF ANY PART OF ANY PACKAGE BY THE LICENSEE TO A THIRD PARTY (INCLUDING SUPPLY AS PART OF ANOTHER SOFTWARE PACKAGE) REQUIRES THE SEPARATE PRIOR WRITTEN AGREEMENT OF HYPROTECH UK LIMITED, WHICH MAY INCLUDE FINANCIAL TERMS.

1.5 Use of HSL 2007 by UK academics

HSL 2007 is available to UK academics for academic research and teaching use without charge. A UK academic is someone who is part of the UK Academic Community and has an electronic mail address ending in .ac.uk . He or she is only permitted to use the software on a UK academic machine, that is one that lies in the .ac.uk internet domain. HSL 2007 packages may be obtained by UK academics by way of the HSL web page

<http://www.cse.clrc.ac.uk/nag/hsl>

Access is by means of short-lived individual password-controlled accounts. Potential users are asked for brief details of the use they intend to make of the package(s) they aim to download. Users are also asked to accept a set of terms and conditions (see Section 1.5.2), and are not permitted to divulge their userid and password to anyone else, nor to distribute any HSL packages to a third party, nor to make any commercial use of any HSL package without prior written agreement of Hyprotech UK Limited (see Section 1.2.1).

A registered user may download one HSL 2007 package at a time. When requesting a package, the user may choose to download the Fortran source for the package, any dependent routines from HSL 2007, the HSL Archive, BLAS or the public domain libraries LAPACK and LINPACK, specification documentation (in either PostScript or PDF format), and, in most cases, a simple Fortran test code that is included in the specification documentation. Where there is more than one version available, the user must choose which version to download (for example, single or double precision).

1.5.1 Acknowledging the use of HSL 2007

It is a condition of use (see below) that all publications written by academic users of HSL 2007 that include results obtained with the help of one or more HSL 2007 or HSL Archive packages shall acknowledge the use of the packages. Users should reference HSL 2007 as follows:

HSL (2007). *A collection of Fortran codes for large scale scientific computation*.
<http://www.numerical.rl.ac.uk/hsl>

1.5.2 UK academic free-licence conditions

The terms and conditions for a UK Academic Licence may be found at

<http://hsl.rl.ac.uk/acuk/cou.html>

and are as follows:

1. AGREEMENT: Acceptance of this order by Council for the Central Laboratory of the Research Councils (CCLRC) constitutes a licence agreement made in England (and subject to the laws of England and to the sole jurisdiction of the Courts of England) between the Licensee and the Central Laboratory of the Research Councils Rutherford Appleton Laboratory, Chilton, Oxfordshire OX11 0QX, United Kingdom (CLRC). No modification expressed to be an

- amendment to this Agreement shall have any effect unless made in writing and signed by one authorised person from each party hereto.
2. **DEFINITIONS:** Library means HSL 2007 as detailed in the Library catalogue dated September 2007. Package means collectively any named subroutine from the Library together with any other subroutine(s) called by it. Software means:
(a) those computer files of the Library (and associated documentation therefor) which are supplied to the Licensee under the provisions of this Agreement and;
(b) all computer files which contain copies whether in whole or in part of any code that has originated by any means from any of the said computer files supplied and;
(c) all documentation which contains copies whether in whole or in part of any text that has originated by any means from any of the said associated documentation and;
(d) any alteration or addition made to the said computer files supplied and/or said associated documentation for the purposes of extending the capabilities or enhancing the performance of the said computer files supplied, whether made by or on behalf of the Licensee or anyone else insofar as such alteration or addition does not infringe the rights of any third party.
 3. The Packages may only be used for academic research or teaching purposes by the Licensee, and must not be copied by the Licensee for use by any other persons. The Licensee must be a part of the UK Academic Community, must have an electronic mail address ending in .ac.uk , and is only permitted to use the Software on a UK academic machine, that is one that lies in the .ac.uk internet domain. All information on the Packages is provided to the Licensee on the understanding that the details thereof are confidential.
 4. All publications issued by the Licensee that include results obtained with the help of one or more of the Packages shall acknowledge the use of the Packages. The Licensee will notify the Numerical Analysis Group at Rutherford Appleton Laboratory of any such publication.
 5. The Packages may be modified by or on behalf of the Licensee for such use in research applications but at no time shall such Packages or modifications thereof become the property of the Licensee. The Licensee shall make available free of charge to CLRC and CLRC may use (and authorise others to use) for any purpose, all information relating to any modification.
 6. **NO LIABILITY:** CCLRC does not make any representations regarding the use, or the results of use, of the Software. The Software is provided as is without any warranty of any kind either expressed or implied. The entire risk as to the results and performance of the Software is assumed by Licensee. CCLRC SPECIFICALLY DISCLAIMS ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING ANY IMPLIED TERM, CONDITION, REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, THAT THE SOFTWARE WILL MEET LICENSEE'S REQUIREMENTS OR THAT THE OPERATION OF THE SOFTWARE WILL BE UNINTERRUPTED OR ERROR FREE. IN NO EVENT SHALL CCLRC, HYPROTECH UK LIMITED, OR ANYONE ELSE WHO HAS BEEN INVOLVED IN THE CREATION, PRODUCTION OR DELIVERY OF THE SOFTWARE BE LIABLE FOR SPECIAL, INDIRECT, INCIDENTAL, CONSEQUENTIAL, EXEMPLARY OR PUNITIVE DAMAGES FOR ANY BREACH OF CLRC'S OBLIGATIONS OR TERMS, CONDITIONS OR WARRANTIES, EXPRESSED OR IMPLIED, IN, OR

RESULTING FROM, THIS LICENCE OR THIS AGREEMENT.

7. **TERMINATION:** This Licence and this Agreement will terminate immediately: (a) upon mutual agreement of both parties; (b) at CLRC's option if the Licensee becomes insolvent or bankrupt or if the Licensee fails to comply with any material provision of this Agreement. Upon any termination, the Licensee shall cease to use the Software, render unusable all copies of it, and deliver to CLRC a certificate (the Certificate) that there are no other copies of the Software within the Licensee's control or possession. If the Licensee fails to return the Certificate and render the Software unusable within 10 days of CLRC's notice, the Licensee shall pay the then current commercial licence fee for each outstanding copy of the Software.
8. The Numerical Analysis Group at Rutherford Appleton Laboratory shall provide limited assistance and support by telephone consultation or other electronic communication, and only if the assistance or support relates to the general operation of the Software. Unless otherwise agreed to in writing, CLRC shall not supply the Licensee with updates or training in the use of the Software or any other maintenance or support.
9. **SUPPLY OF ANY PART OF THE LIBRARY BY THE LICENSEE TO A THIRD PARTY (INCLUDING AS PART OF ANOTHER SOFTWARE PACKAGE) OR USE OF ANY PART OF THE LIBRARY BY THE LICENSEE IN A COMMERCIAL APPLICATION REQUIRES THE SEPARATE PRIOR WRITTEN AGREEMENT OF HYPROTECH UK LIMITED, C1, Reading International Business Park, Basingstoke Road, Reading RG2 6DT, United Kingdom, WHICH MAY INCLUDE FINANCIAL CONDITIONS.**

1.5.3 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 Hyprotech UK Limited, see Section 1.2.1, promptly for confirmation or advice.

1.6 Matlab interfaces for selected HSL packages

We have facilities for building Matlab (MEX) interfaces to several HSL 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 the HSL web page

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

or, directly from

http://www.numerical.rl.ac.uk/hsl_matlab.html

Users must have a licence for the package.

Any problems should be addressed to Mario Arioli (m.arioli@rl.ac.uk).

A – COMPUTER ALGEBRA

AD Automatic differentiation

HSL_AD02 Automatic differentiation

This Fortran 95 package provides automatic differentiation facilities for variables specified by Fortran code. Each independent variable and each variable whose value depends on the value of any independent variable must be declared to be of type `AD02_REAL` instead of default `REAL` (double precision `REAL` in the `DOUBLE` version). Note that Fortran variables of type default `REAL` (double precision `REAL` in the `DOUBLE` version) and default `INTEGER` may enter the computation provided their values do not vary with the values of the independent variables. Both the backward and the forward method are available.

First and second derivatives are available with both the forward and backward methods. Derivatives of any order are available with the forward method. They are stored in a hyper-triangular format so that only one copy of identical derivatives is held.

Unlike `HSL_AD01`, this code allows for several independent calculations being performed at the same time. Each calculation is stored in a structure to which each of its variables has access through a pointer.

A record is kept of the number of occurrences of errors. By default, execution continues after an error in a motoring mode where each operation is executed as an immediate return. Alternatively, an immediate stop may be requested.

ATTRIBUTES — **Version:** 1.1.0. **Types:** Forward(single, double), Backward(single, double). **Calls:** `KB07`. **Language:** Fortran 95. **Date:** June 1995, revised November 2001. **Remark:** This is a threadsafe version of `HSL_AD01` and supersedes it. **Origin:** J.K. Reid, Rutherford Appleton Laboratory and D. Cowey, RMCS Shrivenham.

D – DIFFERENTIAL EQUATIONS

DC Initial-value ordinary differential equation problems

HSL_DC05 Ordinary differential equations or differential algebraic equations of index 1

HSL_DC05 is a suite of Fortran 95 procedures for the solution of a system of ordinary differential equations (ODE) or differential algebraic equations (DAE) of index 1:

$$\mathbf{F}(t, \mathbf{Y}, \mathbf{Y}') = 0 \quad (1)$$

where $\mathbf{Y}' = d\mathbf{Y}/dt$. It provides a powerful optional method of avoiding the incorrect occurrence of negative values for solution components that should remain positive throughout. Some of the components of \mathbf{F} and \mathbf{Y} are purely algebraic; the algebraic equations do not involve \mathbf{Y}' and the rest do not involve the derivatives of the algebraic components of \mathbf{Y} . If there are N_{alg} such algebraic equations and variables, we can split \mathbf{Y} into two vectors: \mathbf{Y}_1 of length N_{alg} and \mathbf{Y}_2 of length $N_{eq} - N_{alg}$, where N_{eq} is the total number of equations (algebraic and differential). In turn, equation (1) can be written as:

$$\mathbf{F}_1(t, \mathbf{Y}_1, \mathbf{Y}_2) = \mathbf{0}, \quad (2)$$

$$\mathbf{F}_2(t, \mathbf{Y}_1, \mathbf{Y}_2, \mathbf{Y}_2') = \mathbf{0}, \quad (3)$$

where \mathbf{F}_1 and \mathbf{F}_2 are vectors of length N_{alg} and $N_{eq} - N_{alg}$, respectively. Components \mathbf{Y}_1 of \mathbf{Y} are purely algebraic, their derivatives \mathbf{Y}_1' being absent. The system must be of index no higher than 1, i.e. it must be non-singular in the sense that equation (2) can in principle be solved for \mathbf{Y}_1 and equation (3) for \mathbf{Y}_2 . HSL_DC05 cannot solve DAE systems of index greater than 1, which give difficulties not present in those of index 1.

The solution starts from given initial values of \mathbf{Y}_2 at $t = T$, using a variable order Backward Differentiation Formula (BDF) method, also known as Gear's method, of order 1 to 5. The method automatically chooses step-size (h) and order of integration formulae (k), and is especially efficient on stiff systems in particular those arising from mass action kinetics. HSL_DC05 has been designed to handle DAE problems, which can be regarded in some sense as limiting cases of ODE problems of infinite stiffness. The methods used in HSL_DC05 are not too inefficient to be acceptable on many non-stiff problems. Function evaluation and linear algebra are carried out in the calling program by using 'reverse communication'.

ATTRIBUTES — **Version:** 1.1.1 **Types:** Real (single, double). **Precision:** At least 8-byte arithmetic is recommended. **Calls:** _GEFA, _GESL. **Language:** Fortran 95. **Date:** July 2004. **Origin:** A.R. Curtis, ARC Scientific, L.C. Daniels, Hyprotech UK Ltd., and J.K. Reid, Rutherford Appleton Laboratory.

DD Two-point boundary value ordinary differential equation problems

DD14 Two-point boundary-value ordinary differential equation

To solve a system of linear or nonlinear first order differential equations

$$y'_i = f_i(x, y_1, y_2, \dots, y_m) \quad i=1, 2, \dots, m$$

where $a \leq x \leq b$, with linear or nonlinear two-point boundary conditions.

$$g_1(y_1(a), \dots, y_m(a), y_1(b), \dots, y_m(b)) = 0 \quad i=1, 2, \dots, m$$

The subroutine should adequately solve problems with mild boundary layers or spikes. It also solves smooth and linear problems efficiently and accurately.

The subroutine requires the user to calculate analytic derivatives and returns a solution according to a user-specified absolute accuracy. For highly nonlinear problems the user may specify an initial mesh and/or an approximation to the solution. The use of a continuation option is also provided. The subroutine will refine the initial uniform or user-supplied mesh adaptively inserting mesh points non-uniformly according to an estimate of the error in the solution.

The method used is a modification of that of Lentini and Pereyra, SIAM J. Numer. Anal., **14**, 91-111 (1977).

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double). **Precision:** At least 8-byte arithmetic is recommended. **Calls:** FD15. **Language:** Fortran 77. **Date:** 1978, revised April 2001. **Remark:** DD14 is a threadsafe version of DD04. **Origin:** Lentini and Pereyra, Univ. of Caracas, modified for HSL by I.S. Duff.

E – EIGENVALUES AND EIGENVECTORS

EA Eigenvalues and eigenvectors of real symmetric matrices

EA16 Compute selected eigenpairs using rational Lanczos method

This routine uses an **implicitly restarted block Lanczos method** or **rational Lanczos method** to compute selected eigenpairs of $\mathbf{Ax} = \lambda\mathbf{x}$ where \mathbf{A} is a large real symmetric matrix or $\mathbf{Ax} = \lambda\mathbf{Mx}$, with \mathbf{A} and \mathbf{M} large real symmetric matrices and either \mathbf{A} or \mathbf{M} positive (semi) definite.

The computed approximate eigenvalues are called Ritz values and the corresponding approximate eigenvectors are Ritz vectors. If we denote by OP the operator that is applied to the vectors in the Lanczos process, EA16 may be used to compute Ritz pairs for one of the following problems:

1. Standard eigenvalue problem : $\mathbf{Ax} = \lambda\mathbf{x}$, \mathbf{A} symmetric.

This is solved using one of the following modes:

- (a) Regular mode. Here $\text{OP} = \mathbf{A}$.
- (b) Shift-invert mode. Here $\text{OP} = (\mathbf{A} - \sigma\mathbf{I})^{-1}$ with σ not an eigenvalue of \mathbf{A} .

The computed Ritz vectors are orthogonal with respect to the standard inner product $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^T \mathbf{y}$.

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

This is solved using one of the following modes:

- (a) Regular inverse mode. Here $\text{OP} = \mathbf{M}^{-1}\mathbf{A}$ with \mathbf{M} nonsingular.
- (b) Shift-invert mode. Here $\text{OP} = (\mathbf{A} - \sigma\mathbf{M})^{-1}\mathbf{M}$ with σ not an eigenvalue of $\mathbf{Ax} = \lambda\mathbf{Mx}$.

The computed Ritz vectors are orthogonal with respect to the \mathbf{M} inner product $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^T \mathbf{M} \mathbf{y}$.

3. Buckling problem : $\mathbf{Ax} = \lambda\mathbf{Mx}$, \mathbf{A} symmetric positive (semi) definite, \mathbf{M} symmetric.

This is solved using the following mode:

- (a) Buckling mode. Here $\text{OP} = (\mathbf{A} - \sigma\mathbf{M})^{-1}\mathbf{A}$ with σ not equal to zero or to an eigenvalue of $\mathbf{Ax} = \lambda\mathbf{Mx}$.

The computed Ritz vectors are orthogonal with respect to the \mathbf{A} inner product $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^T \mathbf{A} \mathbf{y}$.

In the shift-invert and buckling modes, σ is called the pole.

The method is described in detail by Meerbergen and Scott (2000), *The design of a block rational Lanczos code with partial reorthogonalization and implicit restarting*, Rutherford Technical Report RAL-TR-2000-011.

Note that EA16 is designed for computing a limited number of eigenvalues and eigenvectors. EA16 **cannot** be used for computing all the eigenpairs of $\mathbf{Ax} = \lambda\mathbf{x}$ or $\mathbf{Ax} = \lambda\mathbf{Mx}$. The matrices \mathbf{A} and \mathbf{M} need not be available in an explicit form.

ATTRIBUTES — **Version:** 1.1.0. **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, ZA12 (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

New in this release.

HSL_EA19 is designed to compute the p leftmost eigenvalues $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_p$ and corresponding eigenvectors x_1, \dots, x_p of a real symmetric (or Hermitian) operator \mathbf{A} acting in the n -dimensional real (or complex) Euclidean space \mathfrak{R}^n , or, more generally, of the problem

$$\mathbf{Ax} = \lambda\mathbf{Bx}, \quad (1)$$

where \mathbf{B} a real symmetric (or Hermitian) positive definite operator. By applying HSL_EA19 to $-\mathbf{A}$, the user can compute the p 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 λ_1 . The operator \mathbf{T} is called *the preconditioner*.

We stress that neither \mathbf{A} nor \mathbf{B} nor \mathbf{T} needs to be available explicitly: HSL_EA19 only requires the multiplication of sets of vectors by \mathbf{A} , \mathbf{B} , and \mathbf{T} .

HSL_EA19 implements a subspace iteration method, i.e. it generates a sequence of subspaces \mathcal{V}^i , $i = 1, 2, \dots$, that contain approximations to the eigenvectors of the problem. All subspaces \mathcal{V}^i are of the same dimension $m \geq p$. The simplest choice for m is $m = p$ but, in case the problem has clustered eigenvalues, it is desirable that there is a significant gap $\lambda_{m+1} - \lambda_p$ both for a satisfactory rate of convergence to right-most eigenvalues of interest and for the error estimation scheme. The subspace iteration method implemented by HSL_EA19 is based on the Jacobi-conjugate preconditioned gradients (JCPG) scheme of Ovchinnikov. This method requires at least $4m$ vectors of length n (which include m approximate eigenvectors) and two $2m$ by $2m$ matrices to be stored in main memory. $2m$ additional vectors of length n are needed if the user opts for reducing the number of multiplications by \mathbf{A} ; in the case of the generalized problem (1), the same applies to \mathbf{B} .

The user may supply any number $n_i \leq m$ of vectors to be used by the package for the construction of a basis in the initial subspace: this option may be used to reduce the computation time in cases where good approximations to some eigenvectors are available. One way of providing such approximations is by restarting from previously calculated eigenvectors.

ATTRIBUTES — **Version:** 1.0.0 **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.

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:** 1.1.0. **Types:** Real (single, double). **Calls:** `FA14`, `FD15`. **Language:** Fortran 77. **Date:** 1976, revised May 2001. **Remark:** EA22 is a threadsafe version of EA12. **Origin:** I.S. Duff, Harwell.

EA23 Full symmetric: Jacobi's method

Given a real symmetric matrix \mathbf{A} , finds all its eigenvalues λ_i and eigenvectors $\mathbf{x}_i, i=1,2,\dots,n$, where $\mathbf{A}\mathbf{x}_i = \lambda_i\mathbf{x}_i$.

The classical Jacobi method is used.

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double). **Remark:** Supersedes EA03. **Calls:** `FA01`. **Language:** Fortran 77. **Date:** 1976, revised May 2001. **Remark:** EA23 is a threadsafe version of EA13. **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`, `FD15`. **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 λ_i , $i = 1, 2, \dots, r$, 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 , $i = 1, 2, \dots, r$, 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, FD15, 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, FD15, 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.

EC Eigenvalues and eigenvectors of Hermitian matrices

EC23 Full Hermitian matrix: classical Jacobi's method

This subroutine calculates all the eigenvalues and eigenvectors of a complex Hermitian matrix. Given an n by n matrix \mathbf{H} with elements $h_{ij} = \text{conj}\{h_{ji}\}$ it finds solutions λ_i and \mathbf{x}_i , $i = 1, 2, \dots, n$, where $\mathbf{H}\mathbf{x}_i = \lambda_i\mathbf{x}_i$. A variant of the classical Jacobi method is used.

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double). **Calls:** None. **Language:** Fortran 77. **Date:** 1982, revised May 2001. **Remark:** EC23 is a threadsafe version of EC13. **Origin:** S. Clough, Harwell.

EP Packages dependent on MPI

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 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:** 1.1.0. **Types:** Real (single, double). **Calls:** FA14, FD15. **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. **Licence:** A third-party licence for this package is available without charge.

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:** 1.0.0. **Types:** Real (single, double). **Remark:** HSL_FA14 is a Fortran 95 version of FA14. **Calls:** None. **Language:** Fortran 90 or 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.

A nonzero floating-point number is stored in the form $\pm m \beta^e$, where β is known as the base (or radix) of the arithmetic, m is the mantissa (or significand or fraction) and e is the exponent (or characteristic). The mantissa is usually normalized so that any floating-point number has a unique representation. Individual machines differ in the way that the normalization is performed. The exponent is stored as a sequence of binary digits (bits); the sign of the exponent either occupies one of these digits, or, more commonly, the actual value of the exponent is obtained by adding the stored binary representation to a fixed negative bias. The mantissa is represented as

$$m = \sum_{i=1}^n m_i \beta^{-i+j},$$

where $0 \leq m_i < \beta$, j is usually 0 or 1 and m is usually normalized so that $m_1 > 0$. (Under special circumstances, m_1 may be zero; such circumstances are typically associated with implementations of gradual underflow on a particular machine.)

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Remark:** FD15 replaces FD05 and ID05. **Calls:** None. **Language:** Fortran 77. **Date:** February 2005. **Licence:** A third-party licence for this package is available without charge.

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

New in this release.

HSL_KB22 is a suite of Fortran 90 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.

L – 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

$$\text{minimize } \mathbf{c}^T \mathbf{x} = \sum_{j=1}^n c_j x_j \quad (1.1)$$

subject to the constraints

$$\mathbf{Ax} = \mathbf{b}, \quad \text{that is, } \sum_{j=1}^n a_{ij} x_j = b_i, \quad i = 1, 2, \dots, m \quad (1.2)$$

$$l_j \leq x_j \leq u_j, \quad 1 \leq j \leq k, \quad (1.3)$$

$$x_j \geq 0, \quad l \leq j \leq n. \quad (1.4)$$

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 (1.3) 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, FD15, LA15, MC29, ZA12. **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:** FD15, 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:** FD15, 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, MA47 or MA48).

There are versions of the code for shared-memory multiprocessors. They are machine dependent but only require simple features like starting parallel tasks and locks. In principle, a code can be supplied for any shared memory parallel machine but the only two platforms on which this has been tested extensively are the CRAY Y-MP and the ALLIANT FX/80.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** MC21, MC29, MC47, I_AMAX, _GEMV, _TRSV. **Language:** Fortran 77 with parallel extensions to Fortran language. **Date:** September 1995, revised August 2001. **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:** 1.1.0. **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:** 1.1.0. **Types:** Real (single, double). **Remark:** HSL_MA42 is a Fortran 90 version of MA42. **Calls:** I_AMAX, _AXPY, _GER, _GEMV, _TPSV, _GEMM, _TRSM. **Helpful:** MC62, MC63. **Language:** Fortran 90. **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

Improved in this release.

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:** FD15, 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 an n -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.

There are also options for returning the determinant of a square matrix and identifying ignored rows or columns in the rectangular or rank-deficient case.

ATTRIBUTES — **Version:** 2.0.0. **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, FD15, MC13, MC21, MC71, _GEMM, _TPSV, HSL_ZD01. **Language:** Fortran 95 + TR 15581 (allocatable components). **Date:** November 2001, revised July 2004. **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.

Versions exist for globally addressable parallel computers. The parallel versions are machine dependent but only require simple features like starting parallel tasks and locks. In principle, a code can be supplied for any shared memory parallel machine, but the only platforms on which the shared memory code has been tested are the SGI Origin 2000 and the CRAY C98.

ATTRIBUTES — **Version:** 1.1.0. **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 Pothen, Old Dominion University, North Carolina). **Language:** Fortran 77 with parallel extensions to Fortran language. **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

Improved in this release.

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:** 1.0.0. **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 Kernel code for HSL_MA77 (definite case)

New in this release.

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

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

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

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

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:** 1.0.0 **Types:** Real (single, double). **Remark:** The code has been tuned only for 8-byte arithmetic. **Language:** Fortran 95. **Calls:** `_COPY`, `_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 90. **Date:** July 1999. **Origin:** J.K. Reid, Rutherford Appleton Laboratory.

MA57 Sparse symmetric system: multifrontal method

Improved in this release.

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.

In this new release, several extra functionalities are provided. 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:** 3.0.2. **Types:** Real (single, double). **Remark:** Supersedes MA27. **Calls:** FD15, MC71, MC47, 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

Improved in this release.

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.

In this new release, several extra functionalities are provided. 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:** 4.0.0. **Types:** Real (single, double). **Remark:** HSL_MA57 is a Fortran 90 version of MA57. **Calls:** MA57, HSL_ZD01. **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:** 1.1.0. **Types:** Real (single, double). **Calls:** MC71, I_AMAX. **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 and MA61AD perform 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 and MA61BD perform 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

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

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 Kernel code for HSL_MA77 (indefinite case)

New, but not available in initial release.

For a full symmetric matrix that need not be 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 performs symmetric interchanges for stability and uses both 1×1 and 2×2 pivots. 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.

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. The factorization takes the form

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

where \mathbf{A} has order n , \mathbf{P} is a permutation matrix, \mathbf{L}_{11} is a lower triangular matrix of order q , \mathbf{D} is block diagonal of order q , and \mathbf{S}_{22} is a symmetric 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 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 matrices \mathbf{L}_{11} and \mathbf{L}_{21} are held as block matrices with blocks of size at most $nb \times nb$.

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

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

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

ATTRIBUTES — **Version:** 1.0.0 **Types:** Real (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`. **Date:** June 2007. **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:** 1.0.0. **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:** 1.0.0. **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

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

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

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

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 90 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

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

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

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

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 90.

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 Kernel code for HSL_MA78

New in this release.

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

$$\mathbf{PAQ} = \begin{pmatrix} \mathbf{L}_1 & \mathbf{0} \\ \mathbf{L}_2 & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{D}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_2 \end{pmatrix} \begin{pmatrix} \mathbf{U}_1 & \mathbf{U}_2 \\ \mathbf{0} & \mathbf{I} \end{pmatrix},$$

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

$$\mathbf{P} = \begin{pmatrix} \mathbf{P}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix}, \quad \mathbf{Q} = \begin{pmatrix} \mathbf{Q}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix},$$

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

$$\begin{pmatrix} \mathbf{L}_1 & \mathbf{0} \\ \mathbf{L}_2 & \mathbf{I} \end{pmatrix} \mathbf{X} = \mathbf{B}, \quad \begin{pmatrix} \mathbf{D}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \mathbf{X} = \mathbf{B}, \quad \begin{pmatrix} \mathbf{D}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{U}_1 & \mathbf{U}_2 \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \mathbf{X} = \mathbf{B}, \quad \text{and} \quad \begin{pmatrix} \mathbf{U}_1 & \mathbf{U}_2 \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \mathbf{X} = \mathbf{B},$$

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

$$\begin{pmatrix} \mathbf{U}_1^T & \mathbf{0} \\ \mathbf{U}_2^T & \mathbf{I} \end{pmatrix} \mathbf{X} = \mathbf{B}, \quad \begin{pmatrix} \mathbf{D}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{L}_1^T & \mathbf{L}_2^T \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \mathbf{X} = \mathbf{B}, \quad \text{and} \quad \begin{pmatrix} \mathbf{L}_1^T & \mathbf{L}_2^T \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \mathbf{X} = \mathbf{B},$$

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:** 1.0.0. **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 HSL_MA74 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 $A = \{a_{ij}\}$ 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

New in this release, but initial release does not support the indefinite case.

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. 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 positive-definite or singular.

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.

All the data for a problem are held in a structure `keep` and the files that it accesses. It is therefore possible to have more than one problem active at the same time. For each problem, it is permitted to change the real data, in which case a new call of `MA77_factor` is needed. Any change to the integer data, however, must be treated as creating a new problem to be input afresh.

For a very large problem, several direct-access files are used. The actual input/output is performed through the package `HSL_OF01`. This automatically shares the available memory in units called *pages*, whose size and number are under the user's control. If a file become full, `HSL_OF01` opens secondary files and treats the primary file and all its secondaries as a single superfile. To allow the secondary files to reside on different devices, the user may supply an array of path names; the full name of a file is the concatenation of a path name with the file name.

If the problem is not very large, the superfiles may be replaced by arrays in memory. Storage is measured in Fortran storage units, with one unit for default reals and integers, and two units for double precision reals and long integers.

At the heart of the subroutines `MA77_factor` and `MA77_solve` there are calls to the packages `HSL_MA54` and `HSL_MA64` for the efficient partial factorization and partial solution of full sets of symmetric positive definite and symmetric indefinite equations, respectively. These blocks the matrix to reduce caching overheads.

ATTRIBUTES — **Version:** 1.0.0 **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 system: multifrontal out of core

New in this release.

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.

All the data for a problem are held in a structure *keep* and the files that it accesses. It is therefore possible to have more than one problem active at the same time. For each problem, it is permitted to change the real data, in which case a new call of MA78_factor is needed. Any change to the integer data, however, must be treated as creating a new problem to be input afresh.

For a very large problem, several direct-access files are used. The actual input/output is performed through the package HSL_OF01. This automatically shares the available memory in units called *pages*, whose size and number are under the user's control. If a file becomes full, HSL_OF01 opens secondary files and treats the primary file and all its secondaries as a single superfile. To allow the secondary files to reside on different devices, the user may supply an array of path names; the full name of a file is the concatenation of a path name with the file name.

If the problem is not very large, the superfiles may be replaced by arrays in memory. Storage is measured in Fortran storage units, with one unit for default reals and integers, and two units for double precision reals and long integers.

At the heart of the subroutines MA78_factor and MA78_solve there are calls to the package HSL_MA74 for the efficient partial factorization and partial solution of full sets of unsymmetric equations.

ATTRIBUTES — **Version:** 1.0.0. **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.

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 $\tilde{\mathbf{A}} = \mathbf{PAQ}$.

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 .

Described in I.S. Duff, Harwell report R.8730 (1977).

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 $\tilde{\mathbf{A}} = \mathbf{PAQ}$ is block lower triangular and it returns $\tilde{\mathbf{A}}$ 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 MC23 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:** 1.0.0. **Types:** Real (single, double). **Calls:** None. **Language:** Fortran 77. **Date:** February 1987. **Origin:** I.S. Duff, Harwell.

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

Improved in this release.

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:** 2.0.0. **Types:** Real (single, double). **Calls:** MC34, MC59. **Language:** Fortran 77. **Date:** March 1995, revised August 2001 and July 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:** 1.0.0. **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. **Licence:** A third-party licence for this package is available without charge.

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:** 1.0.0. **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. **Origin:** I.S. Duff, 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

New, but not available in initial release.

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:** 1.0.0. **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. **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:** 1.4.0. **Types:** Real (single, double). **Calls:** FD15, MC21. **Language:** Fortran 77. **Date:** July 1999, revised July 2004. **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

New in this release.

Given a sparse unsymmetric or rectangular matrix $\mathbf{A} = \{a_{ij}\}_{m \times n}$, $m \geq n$, 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

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

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 permutation** is 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. It is planned to include further enhancements in future releases.

ATTRIBUTES — **Version:** 1.0.0. **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 **Origin:** I.S. Duff, Rutherford Appleton Laboratory and J. Koster, Trondheim.

HSL_MC65 Construct and manipulate matrices in compressed sparse row format

Improved in this release.

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:** 1.0.0. **Types:** Real (single, double). **Calls:** HSL_ZD01. **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

Improved in this release.

To order an unsymmetric matrix \mathbf{A} into singly bordered blocked diagonal (SBBD) 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 SBBD form:

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

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:** 1.0.0. **Types:** Real (single, double). **Calls:** HSL_FA14, HSL_MC65, HSL_ZD01, HSL_ZD01_char. **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

New in this release.

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:** 1.0.0 **Types:** Real (single, double). **Calls:** HSL_ZB01, HSL_ZD11, 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.

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

Improved in this release.

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 algorithm of Hu and Scott [1] (referred to here as 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:** 2.0.0. **Types:** Real (single, double). **Precision:** At least 8-byte arithmetic is recommended. **Calls:** HSL_MC65, HSL_ZD01, 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, $\|\mathbf{A}^{-1}\|_\infty \|\mathbf{A}\|_\infty$, 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:** 1.0.0. **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:** 1.0.0. **Types:** Real (single, double), Complex (single, double). **Calls:** None. **Language:** Fortran 77. **Date:** July 2004. **Origin:** Daniel Ruiz, E.N.S.E.E.I.H.T., Toulouse (France).

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 $\tilde{\mathbf{A}} = \mathbf{PAQ}$.

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

Improved, but not available in initial release.

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:** 1.1.0 **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.

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

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

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

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

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:** 1.0.0. **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 $A = \{a_{ij}\}_{m \times n}$. 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 $A = \{a_{ij}\}_{n \times n}$. 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

New in this release.

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 — **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:** 1.0.0. **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

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

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:** 1.0.0. **Types:** Real (single, double). **Calls:** _LAEV2. **Language:** Fortran 90. **Date:** August 1996. **Origin:** N.I.M. Gould, Rutherford Appleton Laboratory. **Language:** Fortran 90.

MI11 Unsymmetric system: incomplete LU factorization

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

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

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 \mathbf{M} of an $n \times n$ sparse unsymmetric matrix \mathbf{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 \mathbf{A} and then finding approximate inverses to the resulting diagonal blocks.

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

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

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

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

New in this release.

Given a **block symmetric matrix**

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

where \mathbf{H} has n rows and columns and \mathbf{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 \mathbf{G} is a suitably chosen approximation to \mathbf{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:** 1.0.0 **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

New in this release.

This routine uses the ‘Flexible Generalized Minimal Residual’ method with restarts every 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

$$\bar{\mathbf{A}}\mathbf{x}=\bar{\mathbf{b}} \quad (1, 1)$$

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, 1). 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

New in this release.

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, as described in [1]. 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.

References

[1] K. Stüben. *An introduction to algebraic multigrid*. In U. Trottenberg, C. Oosterlee, A. Schüller, eds, *Multigrid*, Academic Press, 2001, pp 413-532.

ATTRIBUTES — **Version:** 1.0.0 **Types:** Real (single, double). **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

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

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

ATTRIBUTES — **Version:** 1.2.0. **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 $\mathbf{A}\mathbf{x} = \mathbf{b}$, optionally using preconditioning. If \mathbf{P}_L , \mathbf{P}_R are the preconditioning matrices, the routine actually solves the preconditioned system

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

with $\overline{\mathbf{A}} = \mathbf{P}_L\mathbf{A}\mathbf{P}_R$ and $\overline{\mathbf{b}} = \mathbf{P}_L\mathbf{b}$ and recovers the solution $\mathbf{x} = \mathbf{P}_R\overline{\mathbf{x}}$. 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 $\mathbf{A}\mathbf{z}$.

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 $\mathbf{A}\mathbf{x} = \mathbf{b}$, optionally using preconditioning. If \mathbf{P}_L , \mathbf{P}_R are left and right preconditioning matrices, the routine actually solves the preconditioned system

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

with $\overline{\mathbf{A}} = \mathbf{P}_L\mathbf{A}\mathbf{P}_R$ and $\overline{\mathbf{b}} = \mathbf{P}_L\mathbf{b}$. The solution may be recovered as $\mathbf{x} = \mathbf{P}_R\overline{\mathbf{x}}$. 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 $\mathbf{A}\mathbf{z}$.

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 $\mathbf{A}\mathbf{x} = \mathbf{b}$, optionally using preconditioning. If \mathbf{P}_L , \mathbf{P}_R are the preconditioning matrices, the routine actually solves the preconditioned system

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

with $\overline{\mathbf{A}}=\mathbf{P}_L\mathbf{A}\mathbf{P}_R$ and $\overline{\mathbf{b}}=\mathbf{P}_L\mathbf{b}$ and recovers the solution $\mathbf{x}=\mathbf{P}_R\overline{\mathbf{x}}$. 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

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

with $\overline{\mathbf{A}}=\mathbf{P}_L\mathbf{A}\mathbf{P}_R$ and $\overline{\mathbf{b}}=\mathbf{P}_L\mathbf{b}$ and recovers the solution $\mathbf{x}=\mathbf{P}_R\overline{\mathbf{x}}$. 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_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

$$\mathbf{Au}=\mathbf{b},$$

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

$$\overline{\mathbf{A}}\mathbf{y}=\overline{\mathbf{b}},$$

with $\overline{\mathbf{A}}=\mathbf{U}^{-T}\mathbf{A}\mathbf{U}^{-1}$ and $\overline{\mathbf{b}}=\mathbf{U}^{-T}\mathbf{b}$ 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:** 1.0.2. **Types:** Real (single, double). **Calls:** FD15, _DOT, _NRM2, _SCAL, _AXPY. **Language:** Fortran 90. **Date:** July 2004. **Origin:** M. Arioli, Rutherford Appleton Laboratory, and Gianmarco Manzini, IMATI-CNR, Pavia, Italy.

MP Packages dependent on MPI

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 $\mathbf{AX}=\mathbf{B}$ 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

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

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

$$\mathbf{B} = \sum_{k=1}^m \mathbf{B}^{(k)} \tag{2}$$

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 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

$$\mathbf{FY} = \mathbf{C}, \tag{3}$$

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 (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:** 1.1.0. **Types:** Real (single, double). **Calls:** KB08, MA42, MA52, MC53, MC63. **Language:** Fortran 90. **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 $\mathbf{Ax}=\mathbf{b}$ (or $\mathbf{AX}=\mathbf{B}$) where \mathbf{A} has been preordered to singly-bordered block-diagonal form

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

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, L and 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:** 1.1.0. **Types:** Real (single, double). **Remark:** If \mathbf{A} is a sum of finite elements, use HSL_MP42 or HSL_MP62. **Calls:** KB08, MA42, MA52, MC62. **Language:** Fortran 90. **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

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

MPI is used for message passing.

A partial 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, L and 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:** 1.1.0. **Types:** Real (single, double). **Calls:** 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 90. **Date:** March 2003. **Origin:** I.S. Duff and J.A. Scott, Rutherford Appleton Laboratory.

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

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

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

$$\mathbf{B} = \sum_{k=1}^m \mathbf{B}^{(k)} \quad (2)$$

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

$$\mathbf{FY} = \mathbf{C}, \quad (3)$$

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 (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:** 1.1.0. **Types:** Real (single, double). **Calls:** KB08, MA62, MA72, MC53, MC63. **Language:** Fortran 90. **Date:** September 1999. **Origin:** J.A. Scott, Rutherford Appleton Laboratory.

N – NONLINEAR EQUATIONS

NS Solution of systems of nonlinear equations in several unknowns

NS12 Sparse nonlinear equations: Powell dog-leg algorithm

This subroutine solves a system of n nonlinear equations

$$f_i(x_1, x_2, \dots, x_n) = 0, \quad i=1, 2, \dots, n$$

given an initial approximation. It is especially designed for the case where the Jacobian matrix $\mathbf{J} = (\partial f_i / \partial x_j)$ is sparse. It uses a modified version of the ‘dog-leg’ algorithm of Powell. The problem should be reasonably well-scaled, that is the magnitudes of the components x_i of the solution should not vary widely, nor should the magnitudes of the functions f_i .

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double). **Calls:** FD15. **Language:** Fortran 77. **Date:** 1983, revised September 2001. **Remark:** NS12 is a threadsafe version of NS02. **Origin:** N. Munksgaard, CE-DATA, Denmark and J.K. Reid, Harwell.

NS23 Sparse nonlinear over-determined equations: Marquardt method

To solve a system of m nonlinear equations in n unknowns, $\mathbf{x} = x_1, x_2, \dots, x_n$ of the form

$$r_i(\mathbf{x}) = f_i(\mathbf{x}) + \sum_{j=1}^n a_{ij} x_j = 0 \quad i=1, 2, \dots, m \quad m \geq n$$

where the matrices $\mathbf{A} = \{a_{ij}\}_{m \times n}$ and $\mathbf{J} = \{\partial f_i / \partial x_j\}_{m \times n}$ are sparse.

The over-determined case, $m > n$, is handled by taking the solution to be that which minimizes the sum of squares

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

This makes the routine suitable for the nonlinear data fitting problem and parameter variances and covariances can be generated. Derivatives are optional and when not provided by the user are estimated using a definite difference approximation.

The algorithm is based on Fletcher’s version of the Marquardt method.

The user must supply the matrix \mathbf{A} in a condensed form, an initial estimate of \mathbf{x} , and code to calculate $f_i(\mathbf{x})$ and optionally the matrix \mathbf{J} .

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double). **Calls:** MA57, _DOT, FD15 and TD22. **Language:** Fortran 77. **Date:** 1990, revised June 2001. **Remark:** NS23 is a threadsafe version of NS13. **Origin:** J.K. Reid, Rutherford Appleton Laboratory.

O – INPUT/OUTPUT

OF File management

HSL_OF01 Fortran virtual memory

New in this release.

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:** 1.0.0. **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.

T – INTERPOLATION AND APPROXIMATION

TD Estimation of derivatives by finite differences

TD22 Approximate Jacobian matrix using finite differences

Given functions $f_i(x_1, x_2, \dots, x_n, t)$ $i=1, 2, \dots, m$ evaluates an approximation to the Jacobian matrix $\mathbf{J} = \{\partial f_i / \partial x_j\}$ using finite differences. The subroutine is intended for the case when \mathbf{J} is sparse or band structured and has additional entries which given the functions f_i construct the sparsity pattern for \mathbf{J} .

Method references: A.R. Curtis, M.J.D. Powell and J.K. Reid, AERE TP.476, A.R. Curtis and J.K. Reid, AERE TP.477 and J.K. Reid, Harwell report R.7293 (1972).

Derivatives are estimated by

$$\partial f_i / \partial x_j = \frac{1}{2h_j} \{f_i(x_1, \dots, x_j + h_j, \dots, x_n, t) - f_i(x_1, \dots, x_j - h_j, \dots, x_n, t)\}$$

where the steplengths h_j are automatically chosen by the subroutine within bounds specified by the user.

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double). **Precision:** At least 8-byte arithmetic is recommended. **Calls:** FD15. **Language:** Fortran 77. **Date:** 1972, revised June 2001. **Remark:** TD22 is a threadsafe version of TD12. **Origin:** J.K. Reid, Harwell.

V – OPTIMIZATION AND NONLINEAR DATA FITTING

VA Minimization of general functions and sums of squares of functions of several variables

VA08 Minimize a function of several variables: Fletcher-Reeves method

To find the minimum of a general function $f(\mathbf{x})$ of several variables $\mathbf{x} = x_1, x_2, \dots, x_n$ given that values of the derivatives $\partial f / \partial x_i$ can be calculated. The subroutine should be used on large problems when storage space is at a minimum.

The method of conjugate gradients is used.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Precision:** At least 8-byte arithmetic is recommended. **Calls:** `_DOT`. **Language:** Fortran 77. **Date:** January 1972. **Origin:** R. Fletcher, Harwell.

VA34 Minimize a function of a huge number of variables: conjugate gradients

To calculate the minimum of a general function of many variables when values of the derivatives with respect to the variables can be provided, that is, find $\mathbf{x} = \{x_j\}_n$ to minimize a function $F(\mathbf{x})$ given $\partial F / \partial x_j, j=1, 2, \dots, n$.

The subroutine is intended for problems that are so large that an $n \times n$ matrix cannot be stored conveniently.

A conjugate gradient method is used that includes an automatic re-start procedure.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** None. **Language:** Fortran 77. **Date:** 1975, revised July 2001. **Remark:** VA34 is a threadsafe version of VA14. **Origin:** M.J.D. Powell, Harwell. **Licence:** A third-party licence for this package is available without charge.

VA35 Minimize a function of many variables: limited-memory BFGS method

This subroutine solves the unconstrained minimization problem

$$\text{minimize } F(\mathbf{x}), \quad \mathbf{x} = (x_1, x_2, \dots, x_n).$$

The subroutine is especially effective on problems involving a large number of variables and does not require knowledge about the sparsity structure of the Hessian matrix. It uses the limited memory BFGS method, as described by Liu and Nocedal (On The Limited Memory BFGS Method For Large Scale Optimization, Technical Report

NA-03, Department of Electrical Engineering and Computer Science, Northwestern University, 1988). In a typical iteration of this method an approximation H_k to the inverse of the Hessian

$$\left(\frac{\partial^2 F}{\partial x_i \partial x_j} \right)$$

is obtained by applying m BFGS updates to a diagonal matrix H_k^0 , using information from the previous m steps. The user specifies the number m , which also determines the amount of storage required by the subroutine.

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double). **Calls:** `_AXPY`, `_DOT`, `_COPY`, `FD15`. **Language:** Fortran 77. **Date:** 1989, revised September 2001. **Remark:** VA35 is a threadsafe version of VA15. **Origin:** Jorge Nocedal, Northwestern University, Illinois. **Licence:** A third-party licence for this package is available without charge.

VE Minimization of a general function subject to linear constraints

VE08 Minimize a sum of finite-element functions

To minimize an objective function consisting of a sum of ‘finite element’ functions, each of which involves only a few variables or whose second derivative matrix has a low rank for other reasons. Bounds on the variables and known values may be specified. The subroutine is especially effective on problems involving a large number of variables.

The objective function has the form

$$F(\mathbf{x}) = \sum_{k=1}^n f_k(\mathbf{x})$$

and the user is required to write a subroutine that calculates each function f_k and (optionally) its gradient

$$g_k = \left(\frac{\partial f_k}{\partial x_i} \right).$$

When code for the gradient is supplied (and this usually results in faster and more reliable execution), there are facilities for automatically checking it during the first iteration.

There are flexible re-start facilities for problems that do not differ too substantially from a problem previously solved. Using these can sometimes lead to dramatically reduced execution times.

If there is a linear transformation of variables such that one or more of the element functions depend on fewer transformed variables than the number of original variables x_i they involve, this may be specified. This too can lead to substantially improved computing time.

There are facilities for the user: to specify the termination criterion, to influence the

step size in the line-search procedure, and to choose whether to calculate the first approximations to the Hessian matrices

$$H_k = \left(\frac{\partial f_k}{\partial x_i \partial x_j} \right).$$

by finite differences.

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double). **Precision:** At least 8-byte arithmetic is recommended. **Calls:** FD15. **Language:** Fortran 77. **Date:** December 1983. **Origin:** Ph.L. Toint, FUNDP, Namur, Belgium. **Licence:** A third-party licence for this package is available without charge.

VE10 Minimize a sum of squares of element functions

To minimize an objective function consisting of a sum of squares of 'element' functions, each of which involves only a few variables or whose second derivative matrix has a low rank for other reasons. Bounds on the variables and known values may be specified. The subroutine is especially effective on problems involving a large number of variables.

The objective function has the form

$$F(\mathbf{x}) = \sum_{k=1}^{n_s} f_k^2(\mathbf{x}_k) \quad \mathbf{x} = (x_1, x_2, \dots, x_n)$$

where the \mathbf{x}_k are small subsets of \mathbf{x} . The user is required to write a subroutine that calculates each function f_k and (optionally) its gradient

$$g_k = \left(\frac{\partial f_k}{\partial x_i} \right).$$

When code for the gradient is supplied (and this usually results in faster and more reliable execution), there are facilities for automatically checking it during the first iteration.

There are flexible re-start facilities for problems that do not differ too substantially from a problem previously solved. Using these can sometimes lead to dramatically reduced execution times.

If there is a linear transformation of variables such that one or more of the element functions depend on fewer transformed variables than the number of original variables x_i they involve, this may be specified. This too can lead to substantially improved computing time.

There are facilities for the user: to specify the termination criterion, and to choose whether to calculate the approximations to the Hessian matrices

$$H_k = \left(\frac{\partial^2 f_k}{\partial x_i \partial x_j} \right).$$

by finite differences at the starting point.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Precision:** At least 8-byte arithmetic is recommended. **Calls:** VE08. **Language:** Fortran 77. **Date:** March 1987. **Origin:** Ph.L. Toint, FUNDP, Namur, Belgium. **Licence:** A third-party licence for this package is available without charge.

HSL_VE12 Quadratic programming problem: interior-point trust-region method

Improved in this release.

This package uses a primal-dual interior-point trust-region method to solve the quadratic programming problem

$$\text{minimize } \frac{1}{2} \mathbf{x}^T \mathbf{H} \mathbf{x} + \mathbf{g}^T \mathbf{x} + f$$

subject to the general linear constraints

$$c_i^l \leq \mathbf{a}_i^T \mathbf{x} \leq c_i^u, \quad i=1, 2, \dots, m,$$

and the simple bound constraints

$$x_j^l \leq x_j \leq x_j^u, \quad j=1, 2, \dots, n,$$

where the $n \times n$ symmetric matrix \mathbf{H} and the vectors \mathbf{g} , \mathbf{a}_i , \mathbf{x}^l , \mathbf{x}^u , \mathbf{c}^l , and \mathbf{c}^u are given. Full advantage is taken of any zero coefficients in the matrix \mathbf{H} or the vectors \mathbf{a}_i . Any of the constraint bounds x_j^l , x_j^u , c_i^l and c_i^u may be infinite.

If the matrix \mathbf{H} is positive semi-definite, a global solution is found. However, if \mathbf{H} is indefinite, the procedure may find a local solution which is not the global solution to the problem.

ATTRIBUTES — **Version:** 2.0.0. **Types:** Real (single, double). **Calls:** HSL_VE13, HSL_VE15, HSL_VF05, HSL_ZD01, HSL_MA57, ZA12. **Language:** Fortran 95 + TR 15581 (allocatable components). **Date:** September 2000. **Origin:** N.I.M. Gould, Rutherford Appleton Laboratory and Ph.L. Toint, University of Namur, Belgium.

HSL_VE13 Constrained least distance problem: interior-point trust-region method

Improved in this release.

This package uses an primal-dual interior-point trust-region method to solve the constrained least distance problem

$$\text{minimize } \sqrt{\sum_{i=1}^n w_i^2 (x_i - x_i^0)^2}$$

subject to the general linear constraints

$$c_i^l \leq \mathbf{a}_i^T \mathbf{x} \leq c_i^u, \quad i=1, 2, \dots, m,$$

and the simple bound constraints

$$x_j^l \leq x_j \leq x_j^u, \quad j=1, 2, \dots, n,$$

where the vectors \mathbf{w} , \mathbf{x}^0 , \mathbf{a}_i , \mathbf{x}^l , \mathbf{x}^u , \mathbf{c}^l , and \mathbf{c}^u are given.

Full advantage is taken of any zero coefficients in the vectors \mathbf{a}_i . Any of the constraint

bounds x_j^l , x_j^u , c_i^l and c_i^u may be infinite. In the special case where $\mathbf{w}=0$, the so-called analytic centre of the feasible set will be found.

ATTRIBUTES — **Version:** 2.0.0. **Types:** Real (single, double). **Calls:** HSL_VE15, HSL_ZD01, HSL_MA57, ZA12. **Language:** Fortran 95 + TR 15581 (allocatable components). **Date:** September 2000. **Origin:** N.I.M. Gould, Rutherford Appleton Laboratory and Ph.L. Toint, University of Namur, Belgium.

HSL_VE15 Quadratic programming problem: reorder the problem

This package reorders to a standard form the variables and constraints for the quadratic programming problem

$$\text{minimize } \frac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x} + \mathbf{g}^T\mathbf{x} + f$$

subject to the general linear constraints

$$c_i^l \leq \mathbf{a}_i^T \mathbf{x} \leq c_i^u, \quad i=1, 2, \dots, m,$$

and the simple bound constraints

$$x_j^l \leq x_j \leq x_j^u, \quad j=1, 2, \dots, n,$$

where the $n \times n$ symmetric matrix \mathbf{H} and the vectors \mathbf{g} , \mathbf{a}_i , \mathbf{x}^l , \mathbf{x}^u , \mathbf{c}^l , and \mathbf{c}^u are given. Full advantage is taken of any zero coefficients in the matrix \mathbf{H} and the vectors \mathbf{a}_i . Any of the constraint bounds x_j^l , x_j^u , c_i^l and c_i^u may be infinite.

The variables are reordered so that any free variables (i.e., those without bounds) occur first, followed respectively by non-negativities (i.e., those for which the only bounds are that $x_j \geq 0$), lower-bounded variables (i.e., those for which the only bounds are that $x_j \geq x_j^l \neq 0$), range-bounded variables (i.e., those for which the bounds satisfy $-\infty < x_j^l < x_j^u < \infty$) upper-bounded variables (i.e., those for which the only bounds are that $x_j \leq x_j^u \neq 0$), and finally non-positivities (i.e., those for which the only bounds are that $x_j \leq 0$). Fixed variables will be removed. Within each of the above categories, the variables are further ordered so that those with non-zero diagonal Hessian entries occur before the remainder.

The constraints are reordered so that equality constraints (i.e., those for which $c_i^l = c_i^u$) occur first, followed respectively by those which are lower-bounded (i.e., those for which the only bounds are that $\mathbf{a}_i^T \mathbf{x} \geq c_i^l$), those which have ranges (i.e., those for which the bounds satisfy $-\infty < c_j^l < c_j^u < \infty$), and finally those which are upper-bounded (i.e., those for which the only bounds are that $\mathbf{a}_i^T \mathbf{x} \leq c_i^u$). Free constraints, that is those for which $c_i^l = -\infty$ and $c_i^u = \infty$, are removed. Procedures are provided to determine the required ordering, to reorder the problem to standard form, and to recover the problem, or perhaps just the values of the original variables, once it has been converted to standard form. It is anticipated that this module will principally be used as a pre- and post-processing tool for other HSL packages.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** None. **Language:** Fortran 90. **Date:** December 1999. **Origin:** N.I.M. Gould, Rutherford Appleton Laboratory and Ph.L. Toint, University of Namur, Belgium.

HSL_VE19 Quadratic programming problem: working-set method

Improved in this release.

This package uses a working-set method to solve the ℓ_1 quadratic programming problem

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad q(\mathbf{x}) + \rho_g v_g(\mathbf{x}) + \rho_b v_b(\mathbf{x}) \quad (1.1)$$

involving the quadratic objective

$$q(\mathbf{x}) = \frac{1}{2} \mathbf{x}^T \mathbf{H} \mathbf{x} + \mathbf{g}^T \mathbf{x} + f$$

and the infeasibilities

$$v_g(\mathbf{x}) = \sum_{i=1}^m \max(c_i^l - \mathbf{a}_i^T \mathbf{x}, 0) + \sum_{i=1}^m \max(\mathbf{a}_i^T \mathbf{x} - c_i^u, 0)$$

and

$$v_b(\mathbf{x}) = \sum_{j=1}^n \max(x_j^l - x_j, 0) + \sum_{j=1}^n \max(x_j - x_j^u, 0),$$

where the n by n symmetric matrix \mathbf{H} , the vectors \mathbf{g} , \mathbf{a}_i , \mathbf{c}^l , \mathbf{c}^u , \mathbf{x}^l , \mathbf{x}^u , and the scalars f , ρ_g , and ρ_b are given. Full advantage is taken of any zero coefficients in the matrix \mathbf{H} or the vectors \mathbf{a}_i . Any of the constraint bounds c_i^l , c_i^u , x_j^l and x_j^u may be infinite.

The package may also be used to solve the quadratic programming problem

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad q(\mathbf{x}), \quad (1.2)$$

subject to the general linear constraints

$$c_i^l \leq \mathbf{a}_i^T \mathbf{x} \leq c_i^u, \quad i = 1, \dots, m, \quad (1.3)$$

and the simple bound constraints

$$x_j^l \leq x_j \leq x_j^u, \quad j = 1, \dots, n, \quad (1.4)$$

by automatically adjusting the parameters ρ_g and ρ_b in (1.1). Similarly, the package is capable of solving the bound-constrained ℓ_1 quadratic programming problem

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad q(\mathbf{x}) + \rho_g v_g(\mathbf{x}), \quad (1.5)$$

subject to the simple bound constraints (1.4), by automatically adjusting ρ_b in (1.1).

If the matrix \mathbf{H} is positive semi-definite, a global solution is found. However, if \mathbf{H} is indefinite, the procedure may find a (weak second-order) critical point that is not the global solution to the given problem.

ATTRIBUTES — **Version:** 2.1.0. **Types:** Real (single, double). **Calls:** HSL_ZD01, HSL_ZD02, HSL_MA57, HSL_MA69, HSL_KB22, HSL_VE15, HSL_FA14. **Language:** Fortran 95 + TR 15581 (allocatable components). **Date:** October 2001. **Origin:** N.I.M. Gould, Rutherford Appleton Laboratory, and Ph. L. Toint, University of Namur, Belgium.

VE24 Quadratic programming problem within a region defined by simple bounds

These subroutines are for the solution of the general, large, quadratic programming problem within a feasible region defined by simple bound constraints.

ATTRIBUTES — **Version:** 1.1.0. **Types:** Real (single, double). **Calls:** `_AXPY`, `_COPY`, `_DOT`, `MA57`, `FD15`. **Language:** Fortran 77. **Date:** 1992, revised July 2001. **Remark:** VE24 is a threadsafe version of VE14. **Origin:** N.I.M. Gould, Rutherford Appleton Laboratory.

VF Minimization of a general function subject to nonlinear constraints

HSL_VF05 Approximate minimization of a sparse quadratic function with norm constraint

Given real $n \times n$ symmetric matrices \mathbf{H} and \mathbf{M} (with \mathbf{M} positive definite), a real n vector \mathbf{c} and a positive scalar Δ , this package finds an approximate minimizer of the quadratic objective function $\frac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x} + \mathbf{c}^T\mathbf{x}$, where the vector \mathbf{x} is required to satisfy the constraint $\|\mathbf{x}\|_{\mathbf{M}} \leq \Delta$, and where the \mathbf{M} -norm of \mathbf{x} is $\|\mathbf{x}\|_{\mathbf{M}} = \sqrt{\mathbf{x}^T\mathbf{M}\mathbf{x}}$. This problem commonly occurs as a trust-region subproblem in nonlinear optimization calculations. The method may be suitable for large n as no factorization of \mathbf{H} is required. Reverse communication is used to obtain matrix-vector products of the form $\mathbf{H}\mathbf{z}$ and $\mathbf{M}^{-1}\mathbf{z}$.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** `_PTTRF`, `_NRM2`. **Language:** Fortran 90. **Date:** April 1997. **Origin:** N.I.M. Gould, Rutherford Appleton Laboratory.

HSL_VF06 Global minimization of a sparse quadratic function with norm constraint

Improved in this release.

Given a real $n \times n$ symmetric matrix \mathbf{H} , a real n vector \mathbf{c} and a positive radius Δ , this package finds a global minimizer of the quadratic objective function $\frac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x} + \mathbf{c}^T\mathbf{x}$, where the vector \mathbf{x} is required to satisfy the constraint $\|\mathbf{x}\|_{\mathbf{M}} \leq \Delta$, and where the \mathbf{M} -norm of \mathbf{x} is $\|\mathbf{x}\|_{\mathbf{M}} = \sqrt{\mathbf{x}^T\mathbf{M}\mathbf{x}}$. The symmetric positive-definite matrix \mathbf{M} is constructed from an appropriate factorization of \mathbf{H} , and is chosen so that the problem is easy to solve. Such problems commonly occurs as a trust-region subproblems in nonlinear optimization calculations, and it is envisaged that this will be the primary use for this package.

ATTRIBUTES — **Version:** 2.0.0. **Types:** Real (single, double). **Calls:** `HSL_ZD01`, `HSL_MA57`, `ZA12`. **Language:** Fortran 95 + TR 15581 (allocatable components). **Date:** September 2000. **Origin:** N.I.M. Gould, Rutherford Appleton Laboratory.

VH Minimization of functions of integer variables

HSL_VH01 Genetic algorithm for the smallest value of a function of binary variables

This package uses the genetic algorithm to find the smallest value of an objective function of n binary (zero-one) variables. Each *string* of binary variables is stored in a logical array. A *population* of p such strings is maintained along with their associated objective function values. The population evolves in a sequence of iterations. The best features of the population at iteration k are passed to the population at iteration $k+1$ by means of *mutation* and *crossover*.

The package obtains function values and checks for termination by reverse communication.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** HSL_FA14, HSL_ZA03. **Language:** Fortran 90. **Date:** September 1995, revised August 2001. **Origin:** N.I.M. Gould, 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

ZA Timing, machine constants, etc

HSL_ZA03 Kind values for 1- and 2-byte LOGICAL variables

This package provides kind values for 1- and 2-byte Fortran 90 LOGICAL variables. If a particular kind is not supported, a kind offering at least as much storage is substituted.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Integer. **Calls:** None. **Language:** Fortran 90. **Date:** September 2000. **Origin:** N.I.M. Gould, Rutherford Appleton Laboratory.

ZA12 CPU time

Provides the Fortran programmer with the cpu time.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Calls:** CPU_TIME (ISO Fortran 90), ETIME (generic UNIX), MCLOCK (AIX), LIB\$GETJPI (Vax), CPUTIME (IBM mainframe), SECOND (Cray), CLOCK@ (Salford with Intel). **Date:** March 2005. **Licence:** A third-party licence for this package is available without charge.

ZB Array allocation

HSL_ZB01 Reallocate an array

New in this release.

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:** 1.0.0. **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_ZD02 Derived type for quadratic programming

This package defines a derived type capable of supporting a variety of quadratic programming problem storage schemes. Quadratic programming aims to minimize or maximize either a general objective function

$$\frac{1}{2} \mathbf{x}^T \mathbf{H} \mathbf{x} + \mathbf{g}^T \mathbf{x} + \mathbf{f},$$

or sometimes a squared-least-distance objective function,

$$\frac{1}{2} \sum_{i=1}^n w_i^2 (x_i - x_i^0)^2,$$

subject to the general linear constraints

$$c_i^l \leq \mathbf{a}_i^T \mathbf{x} \leq c_i^u, \quad i=1, 2, \dots, m,$$

and the simple bound constraints

$$x_j^l \leq x_j \leq x_j^u, \quad j=1, 2, \dots, n,$$

where the $n \times n$ symmetric matrix \mathbf{H} , the vectors \mathbf{g} , \mathbf{w} , \mathbf{x}^0 , \mathbf{a}_i , \mathbf{c}^l , \mathbf{c}^u , \mathbf{x}^l , and \mathbf{x}^u , and the scalar f are given. Full advantage may be taken of any zero coefficients in the matrix \mathbf{H} or the vectors \mathbf{a}_i . Any of the constraint bounds c_i^l , c_i^u , x_j^l and x_j^u may be infinite.

The principal use of the package is to allow exchange of data between HSL subprograms and other codes.

ATTRIBUTES — **Version:** 1.0.0. **Types:** Real (single, double). **Remark:** This package is also included in the HSL Archive. **Calls:** None. **Date:** July 2000. **Origin:** N.I.M. Gould, Rutherford Appleton Laboratory, and Ph. L. Toint, University of Namur, Belgium. **Language:** Fortran 90. **Licence:** A third-party licence for this package is available without charge.

HSL_ZD11 Derived type for sparse matrix storage schemes

New in this release.

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). **Remark:** This package is also included in the HSL Archive. **Calls:** None. **Language:** Fortran 95 + TR 15581 (allocatable components). **Date:** February 2006. **Origin:** N.I.M. Gould and J.K. Reid, Rutherford Appleton Laboratory. **Licence:** A third-party licence for this package is available without charge.

INDEX

- active-set method** HSL_VE19
- algebraic multigrid** HSL_MI20
- Allocatable array**
 - reallocate an array HSL_ZB01
- analytic centre** HSL_VE13
- approximate minimum degree ordering**
 - HSL_MC68, MC47
- approximate-inverse preconditioner** MI12
- Arnoldi's method** EB13
- assemble finite-element matrix** MC57
- augmented system** HSL_MA69, MA69
- automatic differentiation** HSL_AD02
 - real matrix MC71
- conjugate gradient squared method** MI23
- conjugate gradients** HSL_MI31, MI21,
 - VA08, VA34, VA35
 - preconditioned MA61
- constants**
 - real machine constants FD15
- constrained least-distance problem**
 - HSL_VE13
- constraints**
 - equality
 - least squares
 - full MA44
 - simple bounds
 - finite elements VE08
 - least squares VE10
 - quadratic programming VE24
- covariance matrix** MA75
- cpu time** ZA12
- Cuthill-McKee** MC60, MC61

- data fitting**
 - linear MA44
 - nonlinear NS23
- derived type for sparse matrices**
 - HSL_MC65, HSL_ZD11
- determinant, real sparse matrix**
 - unsymmetric HSL_MA48, MA51
- differential algebraic equations**
 - initial-value problems HSL_DC05
- differential equations, ordinary**
 - boundary value
 - 1st order systems DD14
- differentiation**
 - automatic HSL_AD02
- direct-access i/o facilities** HSL_OF01

- eigenvalues**
 - complex
 - Hermitian full matrix
 - Jacobi's method EC23
 - real
 - symmetric full
 - Jacobi's method EA23
 - symmetric sparse
 - generalised eigenvalue problem
 - EA16, HSL_EA19

- backward error estimate based on iterative refinement** MA60
- balancing a matrix.** see scaling factors for matrices
- banded matrix**
 - approximate Jacobian TD22
 - complex unsymmetric
 - row-by-row frontal ME42, ME43
 - real symmetric positive definite
 - HSL_MA55
 - real unsymmetric MA65
 - row-by-row frontal HSL_MA42,
 - HSL_MP43, MA42, MA43
- bandwidth reduction** HSL_MC73, MC60,
 - MC61
- BFGS**
 - limited memory method VA35
- BiConjugate gradient method** MI25
- BiConjugate gradient stabilized method**
 - MI26
- block triangular form,** see permutation
- bordered block triangular form** MC33
- boundary-value problems**
 - ordinary differential equations
 - 1st order DD14

- Cholesky factorization**
 - banded HSL_MA55
- clock – computer** ZA12
- condition number estimate**
 - classical MC75
 - Skeel's MC75
 - 1-norm or infinity-norm
 - complex matrix MF71

Lanczos EA16, EA25, EP25
 simultaneous iteration EA22
 subspace iteration HSL_EA19
 unsymmetric sparse
 Arnoldi's method EB13
 subspace iteration EB22

eigenvectors, see eigenvalues

element ordering MC63

equations
 nonlinear, see nonlinear equations

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, complex sparse matrix
 Hermitian
 multifrontal ME57
 rectangular ME48, ME50
 symmetric
 multifrontal ME57
 unsymmetric ME48, ME50
 frontal HSL_MA42_ELEMENT, ME42
 multifrontal ME38
 row-by-row frontal ME43

factorization, incomplete
 real unsymmetric MI11

factorization, partial
 real symmetric indefinite HSL_MA64
 real symmetric positive definite HSL_MA54
 real unsymmetric HSL_MA74

factorization, real sparse matrix
 approximate minimum degree ordering HSL_MC68, MC47
 nested bisection ordering HSL_MC68
 rectangular HSL_MA48, MA48, MA50, MA51
 find rank MC58
 symmetric
 banded HSL_MA55
 frontal MA62
 multifrontal HSL_MA77, HSL_MA57, MA57
 out of core HSL_MA77
 multiple front HSL_MP62, MA72
 zeros on diagonal MA67
 symmetric indefinite HSL_MA77, HSL_MA57, MA57
 symmetric positive definite incomplete MA61
 unsymmetric HSL_MA48, LA15, MA48, MA50, MA51
 banded MA65
 find rank MC58
 finite element
 HSL_MA42_ELEMENT, HSL_MA42, MA42, MA46
 frontal HSL_MA42_ELEMENT, HSL_MA42, MA42
 multifrontal HSL_MA78, MA38, MA41, MA46
 out of core HSL_MA78
 multiple front HSL_MP42, HSL_MP43, MA52, MC77
 parallel HSL_MP42, HSL_MP43, HSL_MP48, MA41
 row-by-row frontal MA43
 unsymmetric indefinite HSL_MA78
 update following rank-one change HSL_MA69, LA15, MA69

feasible point problem HSL_VE13

Fiedler vector HSL_MC73

file management HSL_OF01

finite differences
 approximation to derivatives TD22

finite element functions VE08

finite elements
 assemble matrix MC57
 complex
 Hermitian
 frontal ME62
 symmetric
 frontal ME62
 unsymmetric
 frontal HSL_MA42_ELEMENT, ME42
 find connectivity graph MC44
 find supervariables MC44
 generate assembly order MC63
 real
 symmetric
 frontal MA62
 multifrontal HSL_MA77
 multiple front MA72
 parallel HSL_MP62
 unsymmetric
 frontal HSL_MA42_ELEMENT, HSL_MA42, MA42
 multifrontal HSL_MA78, MA46

- multiple front MA52
 - parallel HSL_MP42
- wavefront reduction MC53
- finite-element matrix**
- represent as MC37
- Fletcher-Reeves method** VA08
- frontal method**
- complex
 - Hermitian ME62
 - symmetric ME62
 - unsymmetric
 - HSL_MA42_ELEMENT, ME42
 - row-by-row ME43
- generate element assembly order MC63
- ordering for row-by-row MC62
- real
 - parallel HSL_MP42, HSL_MP43, HSL_MP62
 - symmetric MA62
 - unsymmetric
 - HSL_MA42_ELEMENT, HSL_MA42, MA42
 - row-by-row MA43

Gaussian elimination, see factorization, matrix

Gear's method HSL_DC05

generalised eigenvalue problem EA16, HSL_EA19

generalized minimal residual method MI24

genetic algorithm HSL_VH01

global minimization

quadratic function HSL_VF06

GMRES method MI15

graph

column connectivity HSL_MC65

element connectivity MC44

row connectivity HSL_MC65

supervariable connectivity MC44

Hager's exchange algorithms MC67

heapsort HSL_KB22

Householder reduction

over-determined systems MA44, MA49

incomplete LU factorization MI11

incomplete LDL^T factorization MA61

initial value problems, see differential equations

initial-value problems

differential algebraic equations

HSL_DC05

ordinary differential equations

HSL_DC05

interior-point method HSL_VE12,

HSL_VE13

iterative method

with preconditioning HSL_MI20,

HSL_MI31, HSL_MI02,

HSL_VF05, MI15, MI21, MI23,

MI24, MI25, MI26

iterative refinement MA60

Jacobian

estimate of TD22

Jacobi's method, full matrix

complex Hermitian EC23

real symmetric EA23

kernel linear solvers HSL_MA54,

HSL_MA64, HSL_MA74

kind values

short logical variables HSL_ZA03

Lanczos algorithm EA16, EA25, EP25, HSL_EA19

least squares

full

linear equality constraints MA44

nonlinear

bounds on variables VE10

sparse

linear MA49

nonlinear

unconstrained, Marquardt method NS23

weighted MA75

Lentini and Pereyra's method DD14

linear constraints HSL_VE13,

HSL_ZD02

quadratic programming HSL_VE12, HSL_VE19

linear equations, complex sparse

Hermitian

frontal ME62

multifrontal ME57

symmetric

frontal ME62

multifrontal ME57

unsymmetric ME48, ME50

- frontal HSL_MA42_ELEMENT, ME42
- multifrontal ME38
- row-by-row frontal ME43
- linear equations, real full**
 - least squares MA44
 - partial factorization HSL_MA54, HSL_MA64, HSL_MA74
- linear equations, real sparse**
 - augmented system HSL_MA69, MA69
 - least squares MA49
 - symmetric
 - banded HSL_MA55
 - frontal MA62
 - incomplete factorization MA61
 - iterative HSL_MI31, HSL_MI02, MI15, MI21
 - multifrontal HSL_MA77, HSL_MA57, MA57
 - out of core HSL_MA77
 - multiple front HSL_MP62, MA72
 - zeros on diagonal MA67
 - unsymmetric HSL_MA48, LA15, MA48, MA50, MA51
 - banded MA65
 - frontal HSL_MA42_ELEMENT, HSL_MA42, MA42
 - iterative HSL_MI20, MI11, MI23, MI24, MI25, MI26
 - multifrontal HSL_MA78, MA38, MA41, MA46
 - out of core HSL_MA78
 - multiple front HSL_MP42, HSL_MP43, MA52
 - parallel HSL_MP42, HSL_MP43, HSL_MP48, MA41
 - preconditioning MI12
 - row-by-row frontal MA43
 - weighted least squares MA75
- linear equations, saddle point systems**
 - preconditioning HSL_MI13
- linear least squares**, see least squares
- linear programming**
 - revised simplex method LA04
 - steepest edge LA04
 - update basis LA15
- machine constants**
 - real FD15
- Marquardt method** NS23
- matrix multiplication**
 - sparse HSL_MC65
- maximum transversal** MC21

- minimization of a function**
 - bounds on variables
 - general objective, 1st derivatives, large problems VE08
 - sum of squares, 1st derivatives, large problems VE10
 - combinatorial problems HSL_VH01
 - unconstrained
 - general objective
 - 1st derivatives, large problems VA08, VA34, VA35
 - zero-one variables HSL_VH01
- minimum degree ordering**
 - approximate HSL_MC68, MC47
- MONET algorithm** HSL_MC66
- MPI** EP25, HSL_MP42, HSL_MP43, HSL_MP48, HSL_MP62
- multifrontal method**
 - finite elements HSL_MA77, HSL_MA78, MA46
 - parallel MA41
 - unsymmetric MA38, MA41, ME38
- multiple front method**
 - ordering HSL_MC66, MC53
- real
 - symmetric MA72
 - parallel HSL_MP62
 - unsymmetric MA52
 - parallel HSL_MP42, HSL_MP43
- nonlinear equations, sparse**
 - over-determined, Marquardt method NS23
 - Powell dog-leg NS12
- nonlinear least squares**, see least squares
- non-convex quadratic programming**
 - HSL_VE12, HSL_VE19
- norm**
 - estimate of 1-norm or infinity-norm MC71, MF71
- normal equations** MA75
- optimization**, see minimization of a function
- ordering**
 - approximate minimum degree HSL_MC68, MC47
 - finite-element matrices
 - frontal method MC63
 - multiple front MC53
 - large entries on the diagonal HSL_MC64, MC64, MF64
 - nested bisection HSL_MC68

numbers
 ascending order HSL_KB22, KB05
 with index array KB07
 descending order KB06
 with index array KB08
 profile and wavefront reduction
 HSL_MC73, MC60, MC61, MC67
 row-by-row frontal MC62
 singly bordered block diagonal form
 HSL_MC66
 spectral HSL_MC73
ordinary differential equations
 initial-value problems HSL_DC05
out-of-core factorization
 complex
 Hermitian
 frontal ME62
 symmetric
 frontal ME62
 unsymmetric
 frontal HSL_MA42_ELEMENT,
 ME42
 real
 symmetric
 frontal MA62
 multifrontal HSL_MA77
 multiple front HSL_MP62
 unsymmetric
 frontal HSL_MA42_ELEMENT,
 HSL_MA42, MA42
 multifrontal HSL_MA78
 multiple front HSL_MP42,
 HSL_MP43
over-determined system
 full
 least squares MA44
 sparse MA49
 weighted least squares MA75

parallel
 eigenvalues EP25
 solution of linear equations HSL_MP42,
 HSL_MP43, HSL_MP48,
 HSL_MP62, MA41
permutations
 row, to force diagonal nonzeros for real
 matrix MC21
 symmetric, to block triangular form
 real matrix MC13
 symmetric, to reduce profile MC67
 symmetric, to reduce profile and wavefront
 HSL_MC73, MC60, MC61
 unsymmetric
 complex matrix MC22
 real matrix MC22
 unsymmetric, to block triangular form
 complex matrix MC25, ME22
 real matrix MC25
polynomial (complex coefficients)
 zeros, complex and real PA16
polynomial (real coefficients)
 zeros, complex and real PA17
Powell dog-leg NS12
preconditioning
 approximate-inverse MI12
 incomplete LU MI11
 incomplete LDL^T MA61
 saddle point systems HSL_MI13
profile reduction HSL_MC73, MC60,
 MC61, MC67
pseudo-random numbers, see random
 numbers

quadratic constraint HSL_VF05
quadratic objective function HSL_VF06
quadratic programming HSL_VE13
 active-set method HSL_VE19
 derived type HSL_ZD02
 interior-point method HSL_VE12
 non-convex HSL_VE12, HSL_VE19
 reorder the problem HSL_VE15
 simple bound constraints VE24
Quicksort
 numbers
 ascending order KB05
 with index array KB07
 descending order KB06
 with index array KB08

random matrix YM11
random numbers FA14, HSL_FA14
rank, real sparse matrix
 unsymmetric MC58
reallocate an array HSL_ZB01
Reverse Cuthill-McKee MC60, MC61
roots
 polynomial (complex coefficients) PA16
 polynomial (real coefficients) PA17
Rouche's theorem PA16, PA17
row ordering
 symmetric HSL_MC73, MC60, MC61,
 MC67
 unsymmetric MC62
Rutherford-Boeing format MC54, MC55,
 MC56, YM11

saddle point systems

preconditioner HSL_MI13

scaling factors for matrices

complex

full unsymmetric MC72, MC77

sparse Hermitian MF30

sparse symmetric MC77, MF30

sparse unsymmetric MC77, MF29,
MF64

real

full unsymmetric MC72, MC77

sparse rectangular HSL_MC64

sparse symmetric MC30, MC77

sparse unsymmetric HSL_MC64,
MC29, MC64, MC77

Schur complement HSL_MA69, MA69

simple bound constraints HSL_VE13,
HSL_ZD02

quadratic programming VE24

simplex method LA04

simultaneous iteration

symmetric matrix EA22

unsymmetric matrix EB22

singly bordered block diagonal form

HSL_MC66

Sloan HSL_MC73

solution of linear equations, see linear equations

solution of nonlinear equations, see nonlinear equations

sorting

create structure for sparse symmetric storage
MC34

create structure for sparse symmetric storage
of $A^T A$ MC26

nonzeros of sparse matrix HSL_MC65,
MC46, MC59

numbers

ascending order HSL_KB22, KB05

with index array KB07

descending order KB06

with index array KB08

quadratic programming HSL_VE15

sparse

eigenvalue problems, see eigenvalues

least squares, see least squares

linear equations, see linear equations

matrix

basic operations HSL_MC65

factorization, see factorization, matrix
scaling factors, see scaling factors for
matrices

sorting nonzeros, see sorting

transpose, see transpose sparse matrix

over-determined system, see
over-determined system

test matrices, see Rutherford-Boeing test
matrices

spectral ordering HSL_MC73

subspace iteration

symmetric matrix EA22

unsymmetric matrix EB22

SYMMBK method HSL_MI02

symmetrize sparse matrix HSL_MC65

Tarjan's method MC13

tearing MC33

test matrices

random YM11

timing programs ZA12

transpose sparse matrix HSL_MC65,
MC38, MC46

trust-region method HSL_VE12,
HSL_VE13

trust-region subproblem HSL_VF05,
HSL_VF06

unconstrained minimization VA34, VA35

virtual memory HSL_OF01

wavefront reduction HSL_MC73, MC53,
MC60, MC61, MC63

weighted least squares MA75

zeros of

polynomial (complex coefficients) PA16

polynomial (real coefficients) PA17