1 SUMMARY

HSL_MA57 **solves a sparse symmetric system of linear equations**. Given a sparse symmetric matrix $A = \{a_{ij}\}_{n\times n}$ and an n-vector b or a matrix $B = \{b_{ij}\}_{n\times r}$, this package solves the system $Ax = b$ or the system $AX = B$. The matrix 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 and is discussed further by Duff and Reid, ACM Trans. Math. Software **9** (1983), 302-325. A detailed discussion on the HSL_MA57 strategy and performance is given by Duff, ACM Trans. Math. Software **30** (2004), 118-144. Relevant work on pivoting and scaling strategies is given by Duff and Pralet, SIAM Journal Matrix Analysis and Applications **27** (2005), 313-340. More recent work on static pivoting is given by Duff and Pralet, SIAM Journal Matrix Analysis and Applications **29** (2007), 1007-1024.

The HSL_MA57 package has a range of options including several sparsity orderings, multiple right-hand sides, partial solutions, error analysis, scaling, a matrix modification facility, the efficient factorization of highly rank deficient systems, and a stop and restart facility. Although the default settings should work well in general, there are several parameters available to enable the user to tune the code for his or her problem class or computer architecture.

In the Fortran 95 version, which this package interfaces to, there are added facilities for automatic restarts when storage limits are exceeded, the return of information on pivots, permutations, scaling, modifications, and the possibility to alter the pivots a posteriori. More recent additions exploit sparsity in the right-hand side, compute the Fredholm alternative, multiply vectors by the triangular factor, and return the factors in standard format.

ATTRIBUTES — Version: 5.3.2 (13 July 2022) **Interfaces:** C, Fortran. **Types:** Real (single, double). **Uses:** MA57 (optionally using METIS version 4.x), _GEMM, _TPMV, _TPSV, MC71, HSL_ZD11. **Original date:** September 2000. **Origin:** I. S. Duff, Rutherford Appleton Laboratory. **Language:** Fortran 2003 subset (Fortran 95 + TR 15581 (allocatable members) + C interoperability). **Parallelism:** None. **Remark:** HSL_MA57 is a Fortran 95 encapsulation of MA57 and offers some additional facilities to the Fortran 77 version.

2 HOW TO USE THE PACKAGE

2.1 C interface to Fortran code

This package is written in Fortran and a wrapper is provided for C programmers. This wrapper may only implement a subset of the full functionality described in the Fortran user documentation.

The wrapper will automatically convert between 0-based (C) and 1-based (Fortran) sparse matrix indexing, so may be used transparently from C. This conversion involves both time and memory overheads that may be avoided by supplying sparse matrix index data that is already stored using 1-based indexing. The conversion may be disabled by setting the control parameter control.f_arrays=1 and supplying all sparse matrix index data using 1-based indexing. With 0-based indexing, a sparse matrix is treated as having rows and columns $0, 1, \ldots n-1$. **In this document, we assume 0-based sparse matrix indexing**.

The wrapper uses the Fortran 2003 interoperability features. **Matching C and Fortran compilers must be used**, for example, gcc and gfortran, or icc and ifort. If the Fortran compiler is not used to link the user's program, additional Fortran compiler libraries may need to be linked explicitly.

2.2 Calling sequences

Access to the package requires inclusion of the header file

Single precision version
```
#include "hsl_ma57s.h"
```
Double precision version
```
#include "hsl_ma57d.h"
```

In hsl_ma57s, all reals are default C floats. In hsl_ma57d, all reals are C double precision reals. In both interfaces, all integers are default integers. It is not possible to use more than one version at the same time.

There are five principal functions for user calls (see Section 2.6 for further features):

1. The function ma57_init_factors must be called to initialize the memory for the factors.

2. ma57_default_control must be called to set default values for members of the ma57_control structure needed by other functions. If non-default values are wanted for any of the control members, the corresponding members should be altered after the call to ma57_default_control.

3. ma57_analyse accepts the pattern of A and chooses pivots for Gaussian elimination using a selection criterion to preserve sparsity. It subsequently constructs subsidiary information for the actual factorization by ma57_factorize. The user may provide the pivot sequence, in which case only the necessary information for ma57_factorize will be generated.

4. ma57_factorize factorizes a matrix A using the information from a previous call to ma57_analyse. The actual pivot sequence used may differ from that of ma57_analyse if A is not definite.

5. ma57_solve uses the factors generated by ma57_factorize to solve a system of equations with one ($Ax = b$) or several ($AX = B$) right-hand sides, or to improve a given solution or set of solutions by iterative refinement.

6. ma57_finalize frees the memory allocated by a previous call to ma57_init_factors. It should be called when all the systems involving its matrix have been solved unless the memory is about to be used for the factors of another matrix.

2.3 The derived data types

For each problem, the user must employ the derived type structs defined in the header file to declare structures for holding the matrix factors, controlling the factorization, and providing information. The following pseudo-code illustrates this.

```
#include "hsl_ma57(s|d).h"
...
struct ma57_control control;
void *factors;
struct ma57_ainfo   ainfo;
struct ma57_finfo   finfo;
struct ma57_sinfo   sinfo;
...
```

The members of ma57_control are explained in Section 2.7.1; members of the info structures ma57_ainfo, ma57_finfo and ma57_sinfo for the analyse, solve and factorize phases, respectively, are explained in Sections 2.7.2, 2.7.3 and 2.7.4. The void *factors pointer is used to pass the matrix factors between the functions of the package and must not be altered by the user.

2.4 METIS

The Fortran HSL_MA57 package uses the METIS graph partitioning library available from the University of Minnesota website. If METIS is not available then the user must compile with the supplied replacement function METIS_NodeND.

Important: At present, HSL_MA57 only supports METIS version 4, not the latest version 5 releases.

2.5 Argument lists and calling sequences

2.5.1 Package types

We use the following type definitions in the different versions of the package:

Single precision version

```
typedef float pkgtype
```

Double precision version

```
typedef double pkgtype
```

Elsewhere, for *single* version replace double with float.

2.5.2 The initialization function

The ma57_init_factors function must be called for each structure used to hold the factors. ma57_default_control must also be called for the control structure used to control the subsequent functions.

```
void ma57_default_control(struct ma57_control *control)

void ma57_init_factors(void **factors)
```

control on exit, its members will have been given the default values specified in Section 2.7.1.

factors is the C handle of the Fortran ma57_factors derived type. It must be initialized with this function and passed unalterated to the subsequent functions.

ma57_init_factors dynamically allocates memory on the Fortran side that must always be freed using the function ma57_finalize explained in Section 2.5.6. Trying to free it using C features could lead to undefined behaviour.

2.5.3 To analyse the sparsity pattern

```
void ma57_analyse(int n, int ne, const int row[], const int col[],
      void **factors, const struct ma57_control *control,
      struct ma57_ainfo *ainfo, const int perm[])
```

n the order of the matrix. **Restriction:** n≥1

ne the number of non-zero entries in the matrix. **Restriction:** ne≥0

row and col are arrays of size ne. Each diagonal entry a_{ii} of A must be represented by row[k]=i and col[k]=i and each pair of off-diagonal entries a_{ij} and a_{ji} must be represented by row[k]=i and col[k]=j or by row[k]=j and col[k]=i. Duplicated entries are summed and out-of-range entries are discarded.

factors the factors handle; it must have been initialized by a call to ma57_init_factors or have been used for a previous calculation. In the latter case, the previous data will be lost but the allocatable arrays will not be reallocated unless they are found to be too small.

control the control structure; its members control the action, as explained in Section 2.7.1.

ainfo the info structure; its members provide information about the execution, as explained in Section 2.7.2.

perm is an array of size n that may be NULL. If non-NULL, perm[i], i = 0,1,...,n-1 should be set to the position of variable i in the pivotal sequence.

2.5.4 To perform a factorization

To factorize the matrix, a call of the following form should be made:

```
void ma57_factorize(int n, int ne, const int row[], const int col[],
      const pkgtype val[], void **factors,
      const struct ma57_control *control, struct ma57_finfo *finfo)
```

n the order of the matrix. **Restriction:** n≥1

ne the number of non-zero entries in the matrix. **Restriction:** ne≥0

row and col are arrays of size ne. Each diagonal entry a_{ii} of A must be represented by row[k]=i and col[k]=i and each pair of off-diagonal entries a_{ij} and a_{ji} must be represented by row[k]=i and col[k]=j or by row[k]=j and col[k]=i. Duplicated entries are summed and out-of-range entries are discarded.

val is an array of size ne and val[k] must hold the value of the entry in row[k] and col[k].

factors the factors handle. It must be unaltered since the call to ma57_analyse or a subsequent call to ma57_factorize.

control the control structure; its members control the action, as explained in Section 2.7.1.

finfo the info structure; its members provide information about the execution, as explained in Section 2.7.3.

2.5.5 To solve a set of equations

```
void ma57_solve(int n, int ne, const int row[], const int col[],
      const pkgtype val[], void **factors, int nrhs, pkgtype x[],
      const struct ma57_control *control, struct ma57_sinfo *sinfo,
      const pkgtype rhs[], int iter, int cond)
```

n the order of the matrix. **Restriction:** n≥1

ne the number of non-zero entries in the matrix. **Restriction:** ne≥0

row and col are arrays of size ne. Each diagonal entry a_{ii} of A must be represented by row[k]=i and col[k]=i and each pair of off-diagonal entries a_{ij} and a_{ji} must be represented by row[k]=i and col[k]=j or by row[k]=j and col[k]=i. Duplicated entries are summed and out-of-range entries are discarded.

val is an array of size ne and val[k] must hold the value of the entry in row[k] and col[k].

factors the factors handle. It must be unaltered since the call to ma57_factorize and is not altered by the function.

nrhs the number of right-hand sides. **Restriction:** nrhs≥1

x is a rank-1 array with size x[n] or a rank-2 array with size x[nrhs][n]. If rhs is NULL, x must be set by the user so that either x holds b (for a single right-hand side) or (for multiple right-hand sides) so that x[j][i] holds the component of the right-hand side for variable i to the jth system. On return, either x holds the solution x (for a single right-hand side) or (for multiple right-hand sides) x[j][i] holds the solution for variable i to the jth system. If rhs is non-NULL, x must be set by the user to an approximate solution and on return it holds an improved solution, obtained by one cycle of iterative refinement without any use of arithmetic with additional precision.

control the control structure. Its members control the action, as explained in Section 2.7.1.

sinfo the info structure. Its members provide information about the execution, as explained in Section 2.7.4.

rhs is an array of the same shape as x, that may be NULL. If non-NULL, it must be set by the user so that either rhs holds b (for a single right-hand side) or (for multiple right-hand sides) so that rhs[j][i] holds the component of the right-hand side for variable i to the jth system.

iter if rhs is NULL, this argument is ignored; otherwise, if different from 0, iterative refinement is performed (see Section 2.6.8).

cond if iter is 0, this argument is ignored; otherwise, if different from 0, the condition number of the matrix is computed and is returned in sinfo.cond and sinfo.cond2, the backward error in sinfo.berr and sinfo.berr2, and an estimate of the error in sinfo.error (see Section 2.6.8).

2.5.6 The finalization function

Once all other calls are complete for a problem or after an error return that does not allow the computation to continue, a call should be made to free memory allocated by HSL_MA57 and associated with the factors structure using calls to ma57_finalize

```
void ma57_finalize(void **factors, struct ma57_control *control, int *info)
```

factors the factors hadle. On exit, its allocatable array members will be deallocated. Without such finalization, the storage occupied is unavailable for other purposes.

control the control structure. Its members control the action, as explained in Section 2.7.1.

info on return, the value 0 indicates success. Any other value is the stat value of a Fortran deallocate statement that has failed.

2.6 Further features

In this section, we describe features for enquiring about and manipulating the parts of the factorization constructed. These features will not be needed by a user who wants simply to solve systems of equations with matrix A.

The algorithm produces an LDL^T factorization of a permutation of a scaled version of A, where L is a unit lower triangular matrix and D is a block diagonal matrix with blocks of order 1 and 2. It is convenient to write this factorization in the form

$$PSASP^T = LDL^T$$

where P is a permutation matrix and S the diagonal scaling matrix. The following functions are provided:

1. ma57_enquire functions return P, D, or S.
2. ma57_alter_d alters D. Note that this means that we no longer have a factorization of the given matrix A.

3. ma57_part_solve solves one of the systems of equations $LSx = PSb$, $S^{-1}DS^{-1}x = b$, or $SL^TP^TS^{-1}x = b$, for one or more right-hand sides (b or B, respectively).

4. ma57_sparse_lsolve performs the forward solution using $LD^{1/2}$ and can be used only if the matrix is positive-definite. It exploits sparsity in the right-hand side.

5. ma57_fredholm_alternative solves $Ax = b$. If the system is inconsistent, it also returns a vector that is in the null-space of A and has a non-zero scalar product with the right-hand side b.

6. ma57_lmultiply computes the product of L or L^T with a vector.

7. ma57_get_factors returns the factors L and D for the matrix in the standard Compressed Sparse Column (CSC) format, and the permutation P such that $PAP^T = LDL^T$. It also returns the scaling factors.

2.6.1 To return information on the analysis and factorization

```
void ma57_enquire_perm(const struct ma57_control *control, void **factors, int perm[])
void ma57_enquire_pivots(const struct ma57_control *control, void **factors, int pivots[])
void ma57_enquire_d(void **factors, pkgtype d[])
void ma57_enquire_perturbation(void **factors, pkgtype perturbation[])
void ma57_enquire_scaling(void **factors, pkgtype scaling[])
```

control the control structure. Its members control the action, as explained in Section 2.7.1.

factors the factors handle. It must be unaltered since the call to ma57_factorize or a subsequent call to ma57_alter_d.

perm is an array of size n that will be set to the pivot permutation selected by ma57_analyse.

pivots is an array of size n. On return, the index of pivot i will be placed in pivots[i], i = 0,1,...,n-1 with its sign negative if it is the index of a 2 x 2 block.

d is an array of size d[n][2]. On return, the diagonal entries of D^{-1} will be placed in d[i][0], i = 0,1,...,n-1 and the off-diagonal entries of D^{-1} will be placed in d[i][1], i = 0,1,...,n-2.

perturbation is an array of size n. On return, if ma57_control.pivoting is set to 4, perturbation will be set to the perturbation to the diagonal of the matrix (see Section 2.7.1). If ma57_control.pivoting is not set to 4, perturbation will be set to zero.

scaling is an array of size n. On return, if ma57_control.scaling is set to 1, scaling will be set to the values of the diagonal scaling matrix S. If ma57_control.scaling is not set to 1, scaling will be set to the vector with all entries equal to one.

2.6.2 To alter D

```
void ma57_alter_d(void **factors, const pkgtype d[], int *info)
```

factors the factors handle. It must be unaltered since the call to ma57_factorize or a subsequent call to ma57_alter_d.

d is an array of size d[n][2]. On return, the diagonal entries of D^{-1} will be altered to d[i][0], i = 0,1,...,n-1 and the off-diagonal entries of D^{-1} will be altered to d[i][1], i = 0,1,...,n-2.

info a scalar. On return, the value 0 indicates success and the value i > 0 indicates that d[i-1][1] is nonzero, but is not part of a block of order 2 of D.

2.6.3 To perform a partial solution

```
void ma57_part_solve(void **factors, const struct ma57_control *control,
      char part, int nrhs, pkgtype x[], int *info)
```

`factors` the factors handle. It must be unaltered since the call to `ma57_factorize` or a subsequent call to `ma57_alter_d`.

`control` the control structure. Its members control the action, as explained in Section 2.7.1.

`part` is scalar of type `char`. It must have one of the values

- `L` for solving $LSx = PSb$ or $LSX = PSB$
- `D` for solving $S^{-1}DS^{-1}x = b$ or $S^{-1}DS^{-1}X = B$, or
- `U` for solving $SL^TP^TS^{-1}x = b$ or $SL^TP^TS^{-1}X = B$.

`nrhs` the number of right-hand sides. **Restriction:** `nrhs`$\geq$`1`

`x` is a rank-1 array with size `x[n]` or a rank-2 array with size `x[nrhs][n]`. It must be set by the user so that either `x` holds b (for a single right-hand side) or (for multiple right-hand sides) so that `x[j][i]` holds the component of the right-hand side for variable `i` to the `j`th system. On return, either `x` holds the solution x (for a single right-hand side) or (for multiple right-hand sides) `x[j][i]` holds the solution for variable `i` to the `j`th system.

`info` on return, the value `0` indicates success. Any other value is the `stat` value of a Fortran `deallocate` statement that has failed.

2.6.4 To perform forward elimination exploiting sparsity in right-hand side

```
void ma57_sparse_lsolve(void **factors, const struct ma57_control *control,
      int nzrhs, const int irhs[], int *nzsoln, int isoln[],
      pkgtype x[], struct ma57_sinfo *sinfo)
```

`factors` the factors handle. It must be unaltered since the call to `ma57_factorize` or a subsequent call to `ma57_alter_d`.

`control` the control structure. Its members control the action, as explained in Section 2.7.1.

`nzrhs` must be set by the user to the number of nonzero entries in the right-hand side.

`irhs` an array of size `nzrhs`. It must be set by the user to a list of the components of the right-hand side that are nonzero.

`nzsoln` will be set by the routine to the number of nonzero entries in the solution.

`isoln` an array of size `nzsoln`. The entries `isoln[i]`, `i = 0,...,nzsoln-1` will be set by the routine to a list of the components of the solution that are nonzero.

`x` is an array of size `n`. It must be set by the user to the vector b and on return it holds the solution x.

`sinfo` is a struct of type `ma57_sinfo`. If `sinfo.flag` is equal to `0`, then the execution was successful; if it is equal to `-3`, there was an error in a Fortran `allocate` or `deallocate` call and the `stat` value is returned in `sinfo.stat`. If `sinfo.flag` is equal to `-4`, then the matrix is not positive-definite.

2.6.5 To solve an indefinite potentially singular set of equations

This function computes the same solution of $Ax = b$ as `ma57_solve` but, if the system is deemed to be inconsistent (see parameter `control.consist`), then it will also return a Fredholm alternative vector y satisfying $Ay = 0$ and $y^T b \neq 0$.

```
void ma57_fredholm_alternative(void **factors, const struct ma57_control *control,
      pkgtype x[], pkgtype fredx[], struct ma57_sinfo *sinfo)
```

`factors` the factors handle. It must be unaltered since the call to `ma57_factorize` or a subsequent call to `ma57_alter_d`.

`control` the control structure. Its members control the action, as explained in Section 2.7.1. For this call, it is advised to set `control.tolerance` to 10^{-12} in the prior call to `ma57_factorize`. `control.consist` should be set to the value of the residual at which the equations may be considered consistent.

`x` is an array of size `n`. It must be set by the user to the vector b and on return it holds a solution x of the set of equations. When the system is inconsistent this will be the minimizer of $\|Ax-b\|_{(LL^T)^{-1}}$ of minimum norm.

`fredx` is an array of size `n`. If the system is inconsistent, on return it will hold the vector y from the Fredholm alternative. If the system is consistent, it will not be accessed by the function.

`sinfo` is a struct of type `ma57_sinfo`. If `sinfo.flag` is equal to `0`, then the execution was successful; if it is equal to `-3`, there was an error in a Fortran `allocate` or `deallocate` call and the `stat` value is returned in `sinfo.stat`. If `sinfo.flag` is equal to `1` then the system is inconsistent and a Fredholm alternative vector, y, will be returned in `fredx`.

2.6.6 To form a matrix-vector product using L

This function computes $y = S^{-1}P^T LSx$ or $y = SL^T P^T S^{-1}x$ for a given vector x.

```
void ma57_lmultiply(void **factors, const struct ma57_control *control,
      char trans, pkgtype x[], pkgtype y[], struct ma57_sinfo *sinfo)
```

`factors` the factors handle. It must be unaltered since the call to `ma57_factorize` or a subsequent call to `ma57_alter_d`.

`control` the control structure. Its members control the action, as explained in Section 2.7.1.

`trans` is scalar of type `char`. If it is set to `N` (or `n`), then the vector $S^{-1}P^T LSx$ is returned in y, otherwise $SL^T P^T S^{-1}x$ is returned in y.

`x` is an array of size `n`. It must be set by the user to the vector x.

`y` is an array of size `n`. It will hold the product vector y.

`sinfo` is a struct of type `ma57_sinfo`. If `sinfo.flag` is equal to `0`, then the execution was successful; if it is equal to `-3`, there was an error in a Fortran `allocate` or `deallocate` call and the `stat` value is returned in `sinfo.stat`.

2.6.7 To return the factors in a standard form

```
void ma57_get_factors(void **factors, const struct ma57_control *control,
      int *nzl, int iptrl[], int lrows[], pkgtype lvals[], int *nzd,
      int iptrd[], int drows[], pkgtype dvals[], int perm[],
      int invperm[], pkgtype scale[], struct ma57_sinfo *sinfo)
```

factors the factors handle. It must be unaltered since the call to ma57_factorize or a subsequent call to ma57_alter_d.

control the control structure. Its members control the action, as explained in Section 2.7.1.

nzl is a scalar. On return it will hold the number of entries in the factor L. This will be equal to finfo.nebdu on exit from ma57_factorize.

iptrl is an array of size n+1. iptrl[i] will be set to the position in lrows and lvals of the first entry in column i of L.

lrows is an array of size nzl. It will be set to row indices in the columns of L.

lvals is an array of size nzl. It will be set to the values of entries in the columns of L. Thus, entries in column j of L are in rows lrows[k], k = iptrl[j], iptrl[j+1]-1 and have values lvals[k], k = iptrl[j], iptrl[j+1]-1.

nzd is a scalar. On return it will hold the number of entries in the factor D. This will be equal to 2*finfo.ntwo + n on exit from ma57_factorize. This value is always less than or equal to 2*n.

iptrd is an array of size n+1. iptrd[i] will be set to the position in drows and dvals of the first entry in column i of D.

drows is an array of size nzd. It will be set to row indices in the columns of D.

dvals is an array of size nzd. It will be set to the values of entries in the columns of D. Thus, entries in column j of D are in rows drows[k], k = iptrd[j], iptrd[j+1]-1 and have values dvals[k], k = iptrd[j], iptrd[j+1]-1. Note that the whole 2 by 2 pivot is held. Zeros are held explicitly in the case of singular matrices.

perm is an array of size n. perm will be set to the pivot permutation selected by ma57_factorize.

invperm is an array of size n. invperm will be set to the inverse of perm.

scale is an array of size n. It will be set to the values of the scaling factors.

sinfo is a struct of type ma57_sinfo. If sinfo.flag is equal to 0, then the execution was successful; if it is equal to -3, there was an error in a Fortran allocate or deallocate call and the stat value is returned in sinfo.stat.

2.6.8 Iterative refinement, condition number, and error estimates

If the parameter iter is different from 0 in a call to ma57_solve, then iterative refinement is invoked. If cond is also different from 0, information is returned on the condition number and errors. We use the theory developed by Arioli, Demmel, and Duff (see below). We use the notation $\bar{x}$ for the computed solution and a modulus sign on a vector or matrix to indicate the vector or matrix obtained by replacing all entries by their moduli. We define two scaled residuals:

$$\omega_1 = \max_i \left(\frac{|b - A\bar{x}|_i}{(|b| + |A|\,|\bar{x}|)_i} \right)$$

where the max is over all equations except those for which the numerator is nonzero and the denominator is small. For the exceptional equations, we define

$$\omega_2 = \max_i \left(\frac{|b - A\bar{x}|_i}{(|A|\,|\bar{x}|)_i + \|A_i\|_\infty \|\bar{x}\|_\infty} \right).$$

ω_1 and ω_2 are returned in sinfo.berr and sinfo.berr2, respectively.

The computed solution $\bar{x}$ is the exact solution of the equation

$$(A + \delta A)x = (b + \delta b)$$

where $\delta A_{ij} \leq \max(\texttt{berr}, \texttt{berr2})\,|A|_{ij}$ and $\delta b_i \leq \max(\texttt{berr}|b|_i, \texttt{berr2}\|A_i\|_\infty \|\bar{x}\|_\infty)$. Note that δA respects the sparsity in A. An upper bound for the forward error is returned in `error` that is computed as `berr*cond + berr2*cond2` where `cond` and `cond2` are condition numbers corresponding to κ_{ω_1} and κ_{ω_2}, respectively as defined in equation (15) of Arioli, Demmel, and Duff (1989).

Reference

Arioli, M. Demmel, J. W., and Duff, I. S. (1989). Solving sparse linear systems with sparse backward error. *SIAM J. Matrix Anal. and Applics.* **10**, 165-190.

2.7 The derived types

2.7.1 Derived data type for control of the functions

The derived type `ma57_control` is used to hold controlling data. The members, which are given default values through a call to `ma57_default_control`, are:

`int lp` is used by the functions as the output stream for error messages. If it is negative, these messages will be suppressed. The default value is `6`.

`int wp` is used by the functions as the output stream for warning messages. If it is negative, these messages will be suppressed. The default value is `6`.

`int mp` is used by the functions as the output stream for diagnostic printing. If it is negative, these messages will be suppressed. The default value is `6`.

`int ldiag` is used by the functions to control diagnostic printing. If `ldiag` is less than `1`, no messages will be output. If the value is `1`, only error messages will be printed. If the value is `2`, then error and warning messages will be printed. If the value is `3`, scalar data and a few entries of array data on entry and exit from each function will be printed. If the value is greater than `3`, all data will be printed on entry and exit. The default value is `2`.

`int la` is used by `ma57_factorize`. If `la` $\geq$ `nrlnec` (see Section 2.7.2), the real array that holds data for the factors is reallocated to have size `la`. Otherwise, the array is not reallocated unless its size is less than `nrlnec`, in which case it is reallocated with size `nrltot` (see Section 2.7.2). The default value is `0`.

`int liw` is used by `ma57_factorize`. If `liw` $\geq$ `nirnec` (see Section 2.7.2), the integer array that holds data for the factors is reallocated to have size `liw`. Otherwise, the array is not reallocated unless its size is less than `nirnec`, in which case it is reallocated with size `nirtot` (see Section 2.7.2). The default value is `0`.

`int maxla` is used by `ma57_factorize`. An error return occurs if the real array that holds data for the factors is too small and reallocating it to have size changed by the factor `multiplier` would make its size greater than `maxla`. The default value is `INT_MAX`.

`int maxliw` is used by `ma57_factorize`. An error return occurs if the integer array that holds data for the factors is too small and reallocating it to have size changed by the factor `multiplier` would make its size greater than `maxliw`. The default value is `INT_MAX`.

`pkgtype multiplier` is used by `ma57_factorize` when a real or integer array that holds data for the factors is too small. the array is reallocated with its size changed by the factor `multiplier`. The default value is `2.0`.

`pkgtype reduce` reduces the size of previously allocated internal workspace arrays if they are larger than currently required by a factor of `reduce` or more. The default value is `2.0`.

`int nemin` is used by `ma57_analyse` for the minimum number of eliminations in a step that is automatically accepted. if two adjacent steps can be combined and each has fewer eliminations, they are combined. The default value is `16`.

int thresh is used by ma57_analyse to identify dense rows during pivot selection when the minimum degree algorithm from MA27 is being used (ordering = 3). it is the percentage density for a row to be regarded as dense. The default value is 100.

int pivoting is used to control numerical pivoting by ma57_factorize. it must have one of the following values:

- 1 Numerical pivoting will be performed, with relative pivot tolerance given by the member u.
- 2 No pivoting will be performed and an error exit will occur immediately a sign change is detected among the pivots. This is suitable for cases when A is thought to be definite and is likely to decrease the factorization time while still providing a stable decomposition.
- 3 No pivoting will be performed and an error exit will occur if a zero pivot is detected. This is likely to decrease the factorization time, but may be unstable if there is a sign change among the pivots.
- 4 No pivoting will be performed but the matrix will be altered if a non-positive pivot is encountered.

The default value is 1.

pkgtype u is used by ma57_factorize when the member pivoting has the value 1 to hold the relative pivot tolerance. The default value is 0.01. For problems requiring greater than average numerical care a higher value than the default would be advisable. Values greater than 0.5 are treated as 0.5 and less than 0.0 as 0.0.

pkgtype tolerance is used by ma57_factorize. Any entry of modulus less than or equal to tolerance is treated as zero. The default value is 10^{-20}. If rank_deficient = 1, then blocks of entries less than tolerance can be discarded during the factorization and the corresponding pivots are placed at the end of the ordering. In this case or when the fredholm alternative entry is to be used, a normal value for tolerance could be 10^{-12}.

pkgtype convergence is used by ma57_solve. Iterative refinement will stop if the ratio of the norm of the scaled residual in successive iterations is greater than convergence. The default value is 0.5.

int factorblocking is used by ma57_factorize to determine the block size used for the Level 3 BLAS. The default value is 16.

int solveblocking is used by ma57_solve to determine when to use Level 2 and Level 3 BLAS. The default value is 10.

int ordering is used by ma57_analyse to determine the sparsity ordering that is used. The default value is 5. Possible values are:

- 0 AMD ordering using MC47 (without the detection of dense rows).
- 2 AMD ordering using MC47.
- 3 Minimum degree ordering as generated by the MA27 code.
- 4 METIS ordering is used. Note that the user needs to supply the METIS library (http://www-users.cs.umn.edu/~karypis/metis/metis/download.html).
  If it is not supplied and this option is requested, the routine will return immediately with ainfo.flag set to -10.
- 5 If METIS is available, the routine will make an automatic selection of the ordering choosing either METIS or MC47. If METIS is not available, then action is taken as if ordering were set to 2.
- $\leq$6 Currently is equivalent to setting pivoting = 5, but may be used for alternative orderings in later releases of hsl_ma57.

int scaling is used by ma57_factorize to control the scaling. This has default value 1 and indicates that the matrix will be scaled using a symmetrized version of the HSL code MC64. Setting scaling to any other value will suppress this option although this parameter may be used to call other scalings in future releases.

pkgtype static_tolerance is used by ma57_factorize to control the static pivoting (see Section 4). It has default value 0.0. If the value is positive, we will only accept delayed pivots to a level defined by static_level and small pivots will be increased so that the factorization could be inaccurate. When static pivots are chosen, ma57_solve will automatically use iterative refinement.

pkgtype static_level is used by ma57_factorize. Static pivoting is only invoked if static_tolerance is greater than zero and the accumulated number of delayed pivots exceeds static_level*n. The default value is zero.

int rank_deficient is used by ma57_factorize that has default value 0. If rank_deficient is set to 1, then when small entries (defined by tolerance) are detected, they are removed and the corresponding pivots placed at the end of the factorization. This can be particularly efficient if the matrix is highly rank deficient.

pkgtype consist is used by ma57_fredholm_alternative to determine whether the system being solved is consistent. It has default value 10^{-20}.

C only controls

int f_arrays indicates whether to use C or Fortran sparse matrix indexing. If f_arrays!=0 (i.e. evaluates to true) then 1-based indexing of the arrays ptr, row and order is assumed. Otherwise, if f_arrays=0 (i.e. evaluates to false), then these arrays are copied and converted to 1-based indexing in the wrapper function. This documentation assumes that f_arrays=0. The default is f_arrays=0 (false).

2.7.2 Derived data type for information from ma57_analyse

The interface contains a derived type called ma57_ainfo that is initialized to its default values inside the Fortran interface and, following a call to ma57_analyse, gives the following information:

int flag gives the exit status of the function. The value zero indicates that the function has performed successfully. For nonzero values, see Section 2.8.1.

int more provides further information in the case of an error, see Section 2.8.1.

int oor is set to the number of entries with one or both indices out of range.

int dup is set to the number of duplicate off-diagonal entries.

int stat is set, in the case of the failure of an allocate or deallocate statement, to the stat value.

int nsteps is set to the number of nodes in the assembly tree (number of major steps in the factorization).

int maxfrt holds the largest front size.

pkgtype opsa is set to the number of floating-point additions required by the assembly of frontal matrices if no pivoting is performed. Numerical pivoting may increase the number of operations.

pkgtype opse is set to the number of floating-point operations required by the factorization if no pivoting is performed. Numerical pivoting may increase the number of operations.

int nrltot and int nirtot give the total amount of floating point and integer words respectively required for a successful factorization without the need for data compression, provided no numerical pivoting is performed.

int nrlnec and int nirnec give the total amount of floating point and integer words required respectively for successful factorization allowing data compression, provided no numerical pivoting is performed.

int nrladu and int niradu give the number of floating point and integer words required respectively to hold the matrix factors if no numerical pivoting is performed.

int ncmpa holds the number of compresses of the internal data structure performed by ma57_analyse.

int ordering records the actual ordering used by ma57_analyse using the same numbering as for control.ordering.

2.7.3 Derived data type for information from ma57_factorize

The interface contains a derived type called ma57_finfo that is initialized to its default values inside the Fortran interface and, following a call to ma57_factorize, gives the following information:

int flag gives the exit status of the function. The value of zero indicates that the function has performed successfully. For nonzero values, see Section 2.8.2.

int more provides further information in the case of an error, see Section 2.8.2.

int stat is set, in the case of the failure of an allocate or deallocate statement, to the stat value.

int maxfrt holds the largest front size.

pkgtype opsa is set to the number of floating-point additions performed during assembly.

pkgtype opse is set to the number of floating-point operations performed during factorization.

pkgtype opsb is set to the number of additional floating-point operations performed during factorization because of use of the BLAS.

int nrltot and int nirtot give the total amount of floating point and integer words respectively required for a successful factorization without the need for data compression, provided the same pivots are used.

int nrlnec and int nirnec give the amount of floating point and integer words required respectively for successful factorization allowing data compression, provided the same pivots are used.

int nebdu gives the total number of entries in the factorization.

int nrlbdu and int nirbdu give the amount of floating point and integer words used respectively to hold the factorization.

int ncmpbr and int ncmpbi hold the number of compresses of the real and integer data structure respectively required by the factorization.

int ntwo holds the number of 2 x 2 pivots used during the factorization.

int neig holds the number of negative eigenvalues of A.

int rank holds the rank of the original factorization. Note that, if static pivoting is used (see control.static_tolerance in Section 2.7.1), then the rank of the factorized matrix will always be n.

int delay holds the number of pivots passed up the tree because of numerical pivoting considerations.

int signc holds the number of sign changes of pivot when ma57_control.pivoting is set to 3.

int modstep holds the pivot step at which matrix modification is first performed when ma57_control.pivoting is set to 4. If no matrix modification is performed, modstep is set to 0.

pkgtype maxchange is set to the value of the largest change made to a pivot when ma57_ainfo.modstep is positive.

pkgtype smax is set to the value of the largest scaling factor.

pkgtype smin is set to the value of the smallest scaling factor.

int static_ is set to 1 if static pivots have been chosen and is otherwise set to 0. The corresponding Fortran component is static, but as this clashes with the reserved C keyword, an underscore suffix has been added.

2.7.4 Derived data type for information from ma57_solve

The interface contains a derived type called ma57_sinfo that is initialized to its default values inside the Fortran interface. ma57_sinfo is also used by the functions in Section 2.6. Following a call to ma57_analyse, this structure gives the following information:

int flag gives the exit status of the function. The value of zero indicates that the function has performed successfully. For nonzero values, see Section 2.8.3.

pkgtype cond and pkgtype cond2 hold the condition number of the matrix if the optional parameter cond was included in the call to ma57_solve (see Sections 2.5.5 and 2.6.8).

pkgtype berr and pkgtype berr2 hold the backward error (scaled residual) for the matrix if the optional parameter cond was included in the call to ma57_solve (see Sections 2.5.5 and 2.6.8).

pkgtype error holds an estimate of the error in the solution if the optional parameter cond was included in the call to ma57_solve (see Sections 2.5.5 and 2.6.8).

int stat is set, in the case of the failure of an allocate or deallocate statement, to the stat value.

2.8 Warning and error messages

2.8.1 When analysing the sparsity pattern

A successful return from ma57_analyse is indicated by ainfo.flag having the value zero. A negative value is associated with an error message which will be output on unit control.lp. Possible negative values are:

-1 Value of n out of range. n<1. ainfo.more is set to value of n.

-2 Value of ne out of range. ne<0. ainfo.more is set to value of ne.

-3 Failure of an allocate or deallocate statement. ainfo.stat is set to the stat value.

-9 The array perm does not hold a permutation. ainfo.more holds first member at which an error was detected.

-10 The ordering from METIS was requested but the package was not available.

A positive flag value is associated with a warning message which will be output on unit ainfo.mp. Possible positive values are:

+1 Index (in row or col) out of range. Action taken by function is to ignore any such entries and continue. ainfo.oor is set to the number of such entries. Details of the first ten are printed on unit control.mp.

+2 Duplicate indices. Action taken by function is to keep the duplicates and then to sum corresponding reals when ma57_factorize is called. ainfo.dup is set to the number of faulty entries. Details of the first ten are printed on unit control.mp.

+3 Both out-of-range indices and duplicates exist.

2.8.2 When factorizing the matrix

A successful return from ma57_factorize is indicated by finfo.flag having the value zero. A negative value is associated with an error message which will be output on unit control.lp. In this case, no factorization will have been calculated. Possible negative values are:

-1 Value of n differs from the ma57_analyse value. finfo.more holds value of n.

-2 Value of ne out of range. ne<0. finfo.more holds value of ne.

-3 Failure of an allocate or deallocate statement. finfo.stat is set to the stat value.

-5 Zero pivot detected (control.pivoting has the value 2 or 3). finfo.more is set to the pivot step at which this was detected.

-6 A change of sign of pivots has been detected (control.pivoting has the value 2). finfo.more is set to the pivot step at which this was detected.

-7 The real array that holds data for the factors needs to be bigger than control.maxla.

-8 The integer array that holds data for the factors needs to be bigger than control.maxliw.

A positive flag value is associated with a warning message which will be output on unit control.mp. In this case, a factorization will have been calculated.

+4 Matrix is rank deficient. In this case, finfo.rank will be set to the rank of the original factorization, but the factorization is altered by changing all the zero pivots to one. This will enable the subsequent solution of consistent sets of equations.

+5 Pivots have different signs when control.pivoting has the value 3. finfo.neig is set to the number of negative eigenvalues. Details of the first ten are printed on unit control.mp. finfo.more is set to the number of sign changes.

2.8.3 When solving the system

A successful return from ma57_solve is indicated by sinfo.flag having the value zero. A negative value is associated with an error message which will be output on unit control.lp. In this case, no solution will have been calculated. Possible negative values are:

-3 Failure of an allocate or deallocate statement. sinfo.stat is set to the stat value.

-11 Iterative refinement has failed to converge.

A positive flag value can only be obtained when calling ma57_fredholm_alternative

+1 System is singular and inconsistent. A Fredholm alternative vector is returned as well as a minimum norm solution.

3 GENERAL INFORMATION

Use of common: None

Workspace: Provided automatically by the underlying module.

Other routines called directly: ma57A/AD, ma57B/BD, ma57C/CD, ma57D/DD, ma57E/ED, ma57I/ID, MC71A/AD.

Other modules used directly: HSL_ZD11_single/double.

Restrictions: n≥1; ne≥0; nrhs≥1.

Portability: Fortran 2003 subset (Fortran 95 + TR 15581 + C interoperability).

Changes from Version 1.0.0

Version 2.0.0 incorporates several additional features to those of Version 1.0.0. We give information on these in this section and indicate where default values of control parameters have changed from the earlier version.

Built-in scaling is now available through a symmetrized version of MC64. This is controlled by the new control parameter control.scaling whose default is set so that scaling is performed. The matrix is explicitly scaled internally to the package as are the right-hand side and the solution so that the user need not be concerned with this. Iterative refinement, if requested, is based on the original unscaled matrix. The minimum and maximum scale factors are returned in finfo.smin and finfo.smax, respectively.

The user can now use an ordering from the METIS package. This option is invoked by setting control.ordering equal to 4. Since the user must load the METIS library separately from HSL, a dummy routine has been included that will return the error finfo.flag = -10 if the METIS library is not available.

Static pivoting is now used so that the factorization can be performed using the same storage as predicted by the analysis even if the matrix is not positive definite. This is invoked by setting control.static_tolerance to a nonzero value. A certain amount of of additional storage can be allowed by setting control.static_level to a nonzero value.

Further testing on large problems has determined that a better value for control.nemin is 16 so that this value has now been set as the default for this parameter.

Changes from Version 2.0.0

An additional option has been added to the choice of ordering strategies. If control.ordering is set to 5 (now the default), then ma57_analyse will choose automatically between using METIS or AMD (avoiding problems with dense rows); first based on matrix order and density and then, if necessary, by running both ordering strategies. The actual ordering used is returned in ainfo.ordering.

Changes from Version 3.0.0

Pointer arrays have been replaced by allocatables in derived data types. An added control parameter rank_deficient allows the option of dropping blocks of small entries during the factorization so that the factorization will be much more efficient if the matrix is highly rank deficient.

Changes from Version 4.0.0

Several new facilities were introduced in Version 5.0.0:

1. Triangular solve exploiting sparsity in right-hand side.

2. Computation of Fredholm alternative vector when system is inconsistent.
3. Matrix-vector product using factor L.
4. Return of factors in standard compressed sparse column (CSC) format.

4 METHOD

A version of sparse Gaussian elimination is used.

The ma57_analyse entry (with control.ordering $\neq 1$) chooses a pivot ordering based on either nested dissection or minimum degree using a generalized element model of the elimination to avoid storing the filled-in pattern explicitly. The elimination is represented as an assembly tree with the order of elimination determined by a depth-first search of the tree.

The ma57_factorize entry factorizes the matrix by using the assembly and elimination ordering generated by ma57_analyse. By default, the input matrix is first scaled using a symmetrized version of the HSL code MC64. At each stage in the multifrontal approach, pivoting and elimination are performed on full submatrices and, when diagonal 1×1 pivots would be numerically unstable, 2×2 diagonal blocks are used. The operations on the full submatrices are performed using the Level 3 BLAS. ma57_factorize can thus be used to factor indefinite systems and will perform well on machines with caches or levels of memory hierarchy.

If control.static_tolerance is set to a value greater than zero, static pivoting is invoked. This means that if, at any stage of the multifrontal elimination, there are fully summed rows and columns from which it is not possible to choose pivots because of the threshold criterion defined by control.u, then we first try to get pivots using a weaker threshold by successively trying values one tenth of the previous until a threshold of

$$\sqrt{\texttt{control.u} * \texttt{control.static\_tolerance}}$$

is reached. If there are still fully summed rows and columns left, then the diagonal entries are replaced by control.static_tolerance times the largest modulus of an entry in the scaled matrix and are used as 1×1 pivots in the factorization. If, furthermore, control.static_level is greater than zero, then control.static_tolerance is treated as 0 (uneliminated variables are delayed) until control.static_level*n fully-summed variables have been delayed.

The ma57_solve entry uses the factors from ma57_factorize to solve systems of equations either by loading the appropriate parts of the vectors into an array of the current front-size and using full matrix code employing the Level 2 and Level 3 BLAS or by indirect addressing at each stage, depending on the value of control.solveblocking and the size of the frontal matrix. If static pivots were chosen, then iterative refinement is automatically performed.

A fuller account of this method is given by Duff and Reid (AERE-R.10533, 1982) and Duff and Reid, ACM Trans. Math. Software **9** (1983), 302-325. A description of Version 1.0.0 of the HSL_MA57 package is given by Duff, ACM Trans. Math. Software **30** (2004), 118-144.

Some of the features that are new to Version 2.0.0 are described in "Strategies for scaling and pivoting for sparse symmetric indefinite problems" by Duff and Pralet (SIAM Journal Matrix Analysis and Applications **27** (2005), 313-340). A fuller discussion of our static pivoting strategy is given in the report RAL-TR-2005-007 "Towards a stable static pivoting strategy for the sequential and parallel solution of sparse symmetric indefinite systems" by Duff and Pralet. This report has been accepted for publication in the SIAM Journal on Matrix Analysis and Applications and can be obtained from the web site

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

5 EXAMPLE OF USE

5.1 First example: sparse column entry

We illustrate the use of the package on the solution of the single set of equations

$$A = \begin{pmatrix} 2 & 3 & & & \\ 3 & & 4 & & 6 \\ & 4 & 1 & 5 & \\ & & 5 & 0 & \\ & 6 & & & 1 \end{pmatrix} \quad x = \begin{pmatrix} 8 \\ 45 \\ 31 \\ 15 \\ 17 \end{pmatrix}$$

Note that this example does not illustrate all the facilities.

Program

```
/* hsl_ma57ds.c */
/* Simple example of use of the HSL_MA57 C interface*/

#include <stdio.h>
#include <stdlib.h>
#include "hsl_ma57d.h"

int main(void) {
  typedef double pkgtype;

  struct ma57_control control;
  void *factors;
  struct ma57_ainfo   ainfo;
  struct ma57_finfo   finfo;
  struct ma57_sinfo   sinfo;

  pkgtype *x, *b;
  int i, info;
  /* Members of HSL zd11_type */
  int n, ne, *row, *col;
  pkgtype *val;

  /* Read in the order n of the matrix and number of entries */
  scanf("%d %d", &n, &ne);

  /* Allocate arrays for matrix data and arrays for hsl_ma57 */
  col = (int *)     malloc(ne * sizeof(int));
  row = (int *)     malloc(ne * sizeof(int));
  val = (pkgtype *) malloc(ne * sizeof(pkgtype));
  x   = (pkgtype *) malloc(n * sizeof(pkgtype));
  b   = (pkgtype *) malloc(n * sizeof(pkgtype));

  /* Using a single right-hand side */
  int nrhs = 1;

  /* Read matrix and right-hand side */
```

```
for(i=0; i<ne; i++) {
  scanf("%d %d %lf", &(row[i]), &(col[i]), &(val[i]));
}
for(i=0; i<n; i++) {
  scanf("%lf", &(b[i]));
}

/* Initialize the structures */
ma57_default_control(&control);
ma57_init_factors(&factors);

/* Analyse */
ma57_analyse(n, ne, row, col, &factors, &control, &ainfo, NULL);
if (ainfo.flag < 0) {
  free(col); free(row); free(val); free(x); free(b);
  return 1;
}

/* Factorize */
ma57_factorize(n, ne, row, col, val, &factors, &control, &finfo);
if (finfo.flag < 0) {
  free(col); free(row); free(val); free(x); free(b);
  return 1;
}

/* Solve without refinement */
for(i=0; i<n; i++) {
  x[i] = b[i];
}

ma57_solve(n, ne, row, col, val, &factors, nrhs, x, &control, &sinfo, NULL, 0, 0);
if (sinfo.flag == 0) {
  printf(" Solution without refinement is\n");
  for (i=0; i<n; i++)
    printf("%20.16f  ", x[i]);
}

/* Perform one refinement */
ma57_solve(n, ne, row, col, val, &factors, nrhs, x, &control, &sinfo, b, 0, 0);
if (sinfo.flag == 0) {
  printf("\n Solution after one refinement is\n");
  for (i=0; i<n; i++)
    printf("%20.16f  ", x[i]);
}

/* Clean up */
free(col); free(row); free(val); free(x); free(b);
ma57_finalize(&factors, &control, &info);
```

```
  return 0;
}
```

Data

```
 5 7
 0 0 2.0
 0 1 3.0
 1 2 4.0
 1 4 6.0
 2 2 1.0
 2 3 5.0
 4 4 1.0
 8. 45. 31. 15. 17.
```

Output

```
 Solution without refinement is
  0.9999999999999997    2.0000000000000000    2.9999999999999996    \
  4.0000000000000000    5.0000000000000027
 Solution after one refinement is
  1.0000000000000000    2.0000000000000000    3.0000000000000000    \
  4.0000000000000000    5.0000000000000000
```