Science & Technology Facilities Council

# HSL_MA97

PACKAGE SPECIFICATION HSL 2013

## 1 SUMMARY

HSL_MA97 solves one or more sets of $n \times n$ sparse **symmetric** equations $AX = B$ using a multifrontal method. The package covers the following cases:

1. $A$ is **positive definite**. HSL_MA97 computes the **sparse Cholesky factorization**

$$A = PL(PL)^{\dagger}$$

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

2. $A$ is **indefinite**. HSL_MA97 computes the sparse factorization

$$A = PLD(PL)^{\dagger}$$

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

HSL_MA97 is designed to produce **bit-compatible solutions on any number of threads** (see Sections 2.2 and 2.3). That is to say, regardless of running in serial or parallel, it will always get the same answer (on the same machine with the same binary).

An option exists to scale the matrix. In this case, the factorization of the scaled matrix $\overline{A} = SAS$ is computed, where $S$ is a diagonal scaling matrix.

For large problems where bit-compatible solutions are not required, HSL_MA86 (or HSL_MA87 for positive-definite systems) may provide significantly better parallel performance. For problems where the factors are too large to fit in memory, HSL_MA77 should be used (this allows the matrix data and computed factors to be held in files). HSL_MA77 may also be used for problems held in element form.

**ATTRIBUTES — Version:** 2.3.0 (12 February 2014) **Interfaces:** C, Fortran, MATLAB. **Types:** Real (single, double), Complex (single, double). **Uses:** MC30, HSL_MC34, HSL_MC64, HSL_MC68 (optionally using METIS), HSL_MC69, MC77, HSL_MC78, HSL_MC80, _axpy, _gemm, _gemv, _nrm2, _potrf, _swap, _syrk, _trmv, _trmm, _trsm, _trsv. **Original date:** November 2011. **Origin:** J.D. Hogg and J.A. Scott, Rutherford Appleton Laboratory. **Language:** Fortran 95, plus allocatable dummy arguments and allocatable components of derived types. **Parallelism:** OpenMP 3.0. **Remark:** The development of HSL_MA97 was supported by the EPSRC grant EP/E053351/1.

## 2 HOW TO USE THE PACKAGE

### 2.1 Calling sequences

Access to the package requires a USE statement

Single precision version
```
use hsl_ma97_single
```
Double precision version
```
use hsl_ma97_double
```
Complex version
```
use hsl_ma97_complex
```

Double complex version

```
use hsl_ma97_double_complex
```

If it is required to use more than one module at the same time then the derived types (Section 2.4) must be renamed in one of the use statements.

The following procedures are available to the user:

- ma97_analyse accepts the matrix data in compressed sparse column format and optionally checks it for duplicates and out-of-range entries. The user may supply an elimination order; otherwise one is generated. Using this elimination order, ma97_analyse analyses the sparsity pattern of the matrix and prepares the data structures for the factorization.
- ma97_analyse_coord is an alternative to ma97_analyse that may be used if the user has the matrix data held in coordinate format. Again, the user may supply an elimination order; otherwise one is generated. ma97_analyse_coord checks the matrix data for duplicates and out-of-range entries, stores it in compressed sparse column format and then proceeds in the same way as ma97_analyse.
- ma97_factor uses the data structures set up by ma97_analyse to compute a sparse factorization. More than one call to ma97_factor may follow a call to ma97_analyse (allowing more than one matrix with the same sparsity pattern but different numerical values to be factorized without multiple calls to ma97_analyse). An option exists to scale the matrix.
- ma97_factor_solve may be called in place of ma97_factor to factorize $A$ and, at the same time, solve the system $AX = B$. Multiple calls to ma97_factor_solve may follow a call to ma97_analyse.
- ma97_solve uses the computed factors generated by ma97_factor or ma97_factor_solve to solve systems $AX = B$ for one or more right-hand sides $B$. Multiple calls to ma97_solve may follow a call to ma97_factor or ma97_factor_solve. An option is available to perform a partial solution.
- ma97_finalise should be called after all other calls are complete for a problem (including after an error return that does not allow the computation to continue). It frees memory referenced by components of the derived data types.

In addition, the following routines may be called:

- ma97_free may be called to free memory associated with akeep or fkeep when a call to ma97_finalise is not appropriate (for example, if a further factorization is to be performed for a matrix with the same sparsity pattern).
- ma97_enquire_posdef may be called in the positive-definite case to obtain the pivots used.
- ma97_enquire_indef may be called in the indefinite case to obtain the pivot sequence used by the factorization and the entries of $D^{-1}$.
- ma97_alter may be called in the indefinite case to alter the entries of $D^{-1}$. Note that this means that $PLD(PL)^{\dagger}$ is no longer a factorization of $A$.
- ma97_solve_fredholm is an alternative solve routine that may be called in the indefinite case when the matrix $A$ is found to be singular. It computes the same solution $X$ as ma97_solve but, if the $j-$th system is inconsistent, it also returns $Y_j$ that satisfies $AY_j = 0$ and ${Y_j}^{\dagger}B_j \neq 0$.
- ma97_lmultiply may be called to calculate a matrix-vector or matrix-matrix product with $S^{-1}PL$ or $(S^{-1}PL)^{\dagger}$.
- ma97_sparse_fwd_solve uses the computed factors generated by ma97_factor or ma97_factor_solve to solve the triangular system $PLX = SB$ for a single sparse right-hand side $B$. Multiple calls to ma97_sparse_fwd_solve may follow a call to ma97_factor or ma97_factor_solve.

### 2.2 OpenMP

OpenMP is used by HSL_MA97 to provide parallelism for shared memory environments. To run in parallel, OpenMP must be enabled at compilation time by using the correct compiler flag (usually some variant of -openmp). The number of threads may be controlled at runtime by setting the environment variable OMP_NUM_THREADS.

### 2.3 Achieving bit-compatibility

Care has been taken to allow bit-compatibility to be achieved using this solver. However, testing has revealed that this feature is dependant on the BLAS library used.

In tests it was found that bit-compatibility was impossible to achieve with the GotoBLAS. For the Intel MKL, bit-compatibility can be achieved by setting control%solve_blas3=.true. (using dgemv rather than dgemm during the backwards solve seems to trigger some form of bug). No problems were encountered using the ACML or ATLAS BLAS libraries.

### 2.4 The derived data types

For each problem, the user must employ the derived types defined by the module to declare scalars of the types ma97_control, ma97_info, ma97_akeep, and ma97_fkeep. The following pseudo-code illustrates this.

```
use hsl_ma97_double
...
type (ma97_control) :: control
type (ma97_info) :: info
type (ma97_akeep) :: akeep
type (ma97_fkeep) :: fkeep
...
```

The components of ma97_control and ma97_info are explained in Sections 2.7.1 and 2.7.2. The components of ma97_akeep and ma97_fkeep are used to pass data between the subroutines of the package and must not be altered by the user.

### 2.5 METIS

The HSL_MA97 package optionally uses the METIS graph partitioning library available from the University of Minnesota website. If METIS is not available, the user must link with the supplied dummy subroutine METIS_NodeND. In this case, the METIS ordering option will not be available to the user and, if selected, ma97_analyse and ma97_analyse_coord will return with an error.

Note that if HSL_MA97 is to be run in parallel, it is recommended that either MeTiS is used or the user supplies an elimination computed using a nested dissection-based algorithm.

**Important:** At present, HSL_MA97 only supports MeTiS version 4, not the latest version 5 releases.

### 2.6 Argument lists and calling sequences

#### 2.6.1 Optional arguments

We use square brackets [ ] to indicate OPTIONAL arguments. In each call, optional arguments follow the argument info. Since we reserve the right to add additional optional arguments in future releases of the code, **we strongly recommend that all optional arguments be called by keyword, not by position**.

### 2.6.2 Integer, real and package types

INTEGER denotes default INTEGER and INTEGER(long) denotes INTEGER(kind=selected_int_kind(18)).

REAL denotes default real if the single precision version or the complex version is being used, and double precision real if the double precision or double precision complex version is being used.

We use the term **package type** to mean default real if the single precision version is being used, double precision real for the double precision version, default complex for the complex version and double precision complex for the double complex version.

### 2.6.3 To analyse the sparsity pattern and prepare for the factorization: CSC format

If the matrix data is held in compressed sparse column (CSC) format, the analyse phase optionally checks the user's data for out-of-range and duplicate entries. Only the lower triangular part of the matrix $A$ is required; any entries in the upper triangular part are regarded as out of range. Entries on the diagonal that are zero do not need to be entered explicitly. If checking is carried out, the cleaned matrix data (duplicates are summed during the factorization and out-of-range entries discarded) is held within the ma97_akeep derived data type and the user data ptr and row is not required by any of the remaining subroutines in the package. If the data is not checked, ptr and row must be passed unchanged to the factorization routines. Note that in this case, the presence of out-of-range or duplicates may cause this routine or any of the other routines in the package to fail in an unpredictable way.

A call of the following form should be made:

```
call ma97_analyse(check,n,ptr,row,akeep,control,info[,order,val])
```

check is an INTENT(IN) scalar of type LOGICAL. If set to .true. the matrix data is checked for errors and the cleaned matrix (duplicates are summed and out-of-range entries discarded) is stored in akeep. Otherwise, no checking of the matrix data is carried out and ptr and row must be passed unchanged to the factorization routines.

n is an INTENT(IN) scalar of type INTEGER that must hold the order of $A$. **Restriction:** n$\geq$0.

ptr is an INTENT(IN) rank-1 array of type INTEGER and size n+1. ptr(j) must be set by the user so that ptr(j) is the position in row of the first entry in column j and ptr(n+1) must be set to one more than the number of matrix entries being input by the user.

row is an INTENT(IN) rank-1 array of type INTEGER. It must hold the row indices of the entries of the lower triangular part of $A$ with the row indices for the entries in column 1 preceding those for column 2, and so on (within each column, the row indices may be in arbitrary order). If check is set to .false., row must contain no duplicates or out-of-range entries (including no entries in the upper triangular part).

akeep is an INTENT(OUT) scalar of type ma97_akeep. It is used to hold data about the problem being solved and must be passed unchanged to the other subroutines.

control is an INTENT(IN) scalar of type ma97_control (see Section 2.7.1).

info is an INTENT(OUT) scalar of type ma97_info. Its components provide information about the execution of the subroutine, as explained in Section 2.7.2.

order is an optional INTENT(INOUT) rank-1 array of type INTEGER and size n. If control%ordering=0, order must be present and order(i) must hold the position of variable $i$ in the elimination order. If the user wants to suggest a 2$\times$2 pivot involving variables $i$ and $j$, order(j) should be set to order(i)+1. If control%ordering$\geq$1, order need not be set on entry and the elimination order is computed within the analyse phase. On exit, order contains the elimination order that ma97_factor or ma97_factor_solve will be given (it is passed to these routines as part of akeep); this order may give slightly more fill-in than the user-supplied order and, in the

indefinite case, may be modified by ma97_factor or ma97_factor_solve to maintain numerical stability. Note that, if the user intends to call ma97_sparse_fwd_solve, order must be present and must be passed unchanged to ma97_sparse_fwd_solve.

val is an optional INTENT(IN) rank-1 array of package type. If present, val(k) must hold the value of the entry in row(k). val must be present if a matching-based elimination ordering is required (control%ordering=7 or 8).

#### 2.6.4 To analyse the sparsity pattern and prepare for the factorization: coordinate format

If the matrix data is held in coordinate format, entries in the upper and/or lower triangular part of $A$ may be input using a call of the following form:

```
call ma97_analyse_coord(n,ne,row,col,akeep,control,info[,order,val])
```

n is an INTENT(IN) scalar of type INTEGER that must hold the order of $A$. **Restriction:** n≥0.

ne is an INTENT(IN) scalar of type INTEGER that must hold the number of matrix entries being input by the user. **Restriction:** ne≥1.

row and col are INTENT(IN) rank-1 arrays of type INTEGER and 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.

akeep, control, info, order: see Section 2.6.3.

val is an optional INTENT(IN) rank-1 array of package type. If present, val(k) must hold the value of the entry in row(k) and col(k). val must be present if a matching-based elimination ordering is required (control%ordering=7 or 8).

#### 2.6.5 To factorize the matrix and optionally solve AX = B

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

```
call ma97_factor(matrix_type,val,akeep,fkeep,control,info[,scale,ptr,row])
```

If the user wishes to solve at the same time as factorizing the matrix, a call of the following form should be made for a single right-hand side:

```
call ma97_factor_solve(matrix_type,val,x1,akeep,fkeep,control,info[,scale,ptr,row])
```

or, for more than one right-hand side,

```
call ma97_factor_solve(matrix_type,val,nrhs,x,ldx,akeep,fkeep,control,info[,scale,ptr,row])
```

matrix_type is an INTENT(IN) scalar of type INTEGER that must be set as follows:

3 if $A$ is real, symmetric positive definite

4 if $A$ is real, symmetric indefinite

-3 if $A$ is Hermitian, positive definite

-4 if $A$ is Hermitian, indefinite

-5 if $A$ is complex, symmetric indefinite

**Restriction:** matrix_type = 3, 4 (real case), matrix_type = -3, -4, -5 (complex case).

val is an INTENT(IN) rank-1 array of package type. If ma97_analyse was called, val(k) must hold the value of the entry in row(k). Otherwise, if ma97_analyse_coord was called, val(k) must hold the value of the entry in row(k) and col(k).

akeep is an INTENT(IN) scalar of type ma97_akeep that must be unchanged since the call to ma97_analyse or ma97_analyse_coord.

control, info: see Section 2.6.3.

fkeep is an INTENT(INOUT) scalar of type ma97_fkeep. It is used to hold data about the problem being solved and must be passed unchanged to the other subroutines.

x1 is an INTENT(INOUT) rank-1 array of package type and size n. It must be set so that x1(i) holds the component of the right-hand side for variable i. On exit, x1(i) holds the solution for variable i.

nrhs is an INTENT(IN) scalar of type INTEGER that holds the number of right-hand sides. **Restriction:** nrhs$\geq$1.

x is an INTENT(INOUT) rank-2 array of package type with extents ldx and nrhs. It must be set so that x(i,j) holds the component of the right-hand side for variable i to the jth system. On exit, x(i,j) holds the solution for variable i to the jth system.

ldx is an INTENT(IN) scalar of type INTEGER that must be set to the first extent of the array x. **Restriction:** ldx$\geq$n.

scale is an optional INTENT(INOUT) rank-1 array of type REAL and size n. If control%scaling$>$0, scaling is performed and if scale is present, on exit it contains the diagonal entries of the scaling matrix $S$. If control%scaling$\leq$0 and scale is not present, no scaling is performed; if control%scaling$\leq$0 and scale is present, it must contain the diagonal entries of the scaling matrix $S$ and is unchanged on exit.

ptr and row are optional INTENT(IN) rank-1 arrays of type INTEGER. They are only accessed if ma97_analyse was called with check set to .false.. In this case, they must both be present and must be unchanged since that call.

### 2.6.6 To solve linear systems using the computed factors

After the call to ma97_factor (or ma97_factor_solve), one or more calls of the following form may be made to solve $AX = B$. Partial solutions may be performed by appropriately setting the optional parameter job. For a single right-hand side,

```
call ma97_solve(x1,akeep,fkeep,control,info[,job])
```

or, for more than one right-hand side,

```
call ma97_solve(nrhs,x,ldx,akeep,fkeep,control,info[,job])
```

x1, nrhs, x, ldx, akeep: see Section 2.6.5.

fkeep is an INTENT(IN) scalar of type ma97_fkeep that must be unchanged since the last call to ma97_factor or ma97_factor_solve.

control, info: see Section 2.6.3.

job is an optional INTENT(IN) scalar of type INTEGER. If absent, $AX = B$ is solved. In the positive-definite case, the Cholesky factorization that has been computed may be expressed in the form

$$SAS = (PL)(PL)^\dagger$$

where $P$ is a permutation matrix and $L$ is lower triangular. In the indefinite case, the factorization that has been computed may be expressed in the form

$$SAS = (PL)D(PL)^\dagger$$

where $P$ is a permutation matrix, $L$ is unit lower triangular, and $D$ is block diagonal with blocks of order 1 and 2. $S$ is a diagonal scaling matrix ($S$ is equal to the identity, if control%scaling=0 and scale is not present on the last call to ma97_factor or ma97_factor_solve). A partial solution may be computed by setting job to have one of the following values:

1 for solving $PLX = SB$

2 for solving $DX = B$ (indefinite case only)

3 for solving $(PL)^\dagger S^{-1}X = B$

4 for solving $D(PL)^\dagger S^{-1}X = B$ (indefinite case only)

**Restriction:** job = 1,2,3,4.

### 2.6.7 The finalisation and free subroutines

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_MA97 and associated with the structures akeep and/or fkeep using calls to ma97_free.

The ma97_finalise call is provided as a convenient shortcut for freeing memory associated with both akeep and fkeep.

```
call ma97_free(akeep)
call ma97_free(fkeep)
call ma97_finalise(akeep,fkeep)
```

akeep is an INTENT(INOUT) scalar of type ma97_akeep that must be passed unchanged. On exit, allocatable components will have been deallocated.

fkeep is an optional INTENT(INOUT) scalar of type ma97_fkeep that must be passed unchanged. On exit, allocatable components will have been deallocated.

### 2.6.8 To obtain information on the factorization (positive-definite case)

After a successful call to ma97_factor or to ma97_factor_solve with matrix_type=3 or -3 and prior to a call to ma97_free or ma97_finalise, information on the pivots may be obtained using a call of the form

```
call ma97_enquire_posdef(akeep,fkeep,control,info,d)
```

akeep, fkeep: see Section 2.6.6.

control, info: see Section 2.6.3.

d is an INTENT(OUT) rank-1 array of type REAL and size n. The i-th pivot will be placed in d(i), i = 1,2,...,n.

### 2.6.9 To obtain information on the factorization (indefinite case)

After a successful call to ma97_factor or to ma97_factor_solve with matrix_type=4, -4 or -5 and prior to a call to ma97_free or ma97_finalise, information on the pivot sequence and the matrix $D^{-1}$ may be obtained using a call of the form

```
call ma97_enquire_indef(akeep,fkeep,control,info[,piv_order,d])
```

akeep, fkeep: see Section 2.6.6.

control, info: see Section 2.6.3.

piv_order is an optional INTENT(OUT) rank-1 array of type INTEGER and size n. If present, then if i is used to index a variable, its position in the pivot sequence will be placed in piv_order(i), with its sign negative if it is part of a $2 \times 2$ pivot.

d is an optional INTENT(OUT) rank-2 array of package type with extents 2 and n. If present, the diagonal entries of $D^{-1}$ will be placed in d(1,i), i = 1,2,...,n, the off-diagonal entries of $D^{-1}$ will be placed in d(2,i), i = 1,2,...,n-1, and d(2,n) will be set to zero.

### 2.6.10 To alter $D^{-1}$

After a successful call to ma97_factor or to ma97_factor_solve with matrix_type=4, -4 or -5 and prior to a call to ma97_free or ma97_finalise, the matrix $D^{-1}$ may be altered using a call of the form

```
call ma97_alter(d,akeep,fkeep,control,info)
```

d is an INTENT(IN) rank-2 array of package type with extents 2 and n. The diagonal entries of $D^{-1}$ will be altered to d(1,i), i = 1,2,...,n, and the off-diagonal entries will be altered to d(2,i), i = 1,2,...,n-1 (and $PLD(PL)^{\dagger}$ will no longer be a factorization of $A$).

akeep, fkeep: see Section 2.6.6.

control, info: see Section 2.6.3.

### 2.6.11 To solve linear systems in the indefinite singular case

In the indefinite case, after the call to ma97_factor or ma97_factor_solve, one or more calls of the following form may be made. It computes the same solution $X$ as ma97_solve but, if the $j-$th system is inconsistent, it also returns the Fredholm alternative solution $Y_j$ that satisfies $AY_j = 0$ and ${Y_j}^{\dagger} B_j \neq 0$.
For one or more right-hand sides,

```
call ma97_solve_fredholm(nrhs,flag_out,x,ldx,akeep,fkeep,control,info)
```

nrhs, ldx, akeep, fkeep, control, info: see Section 2.6.6.

flag_out is an INTENT(OUT) rank-1 array of size nrhs and type LOGICAL. On exit, flag_out(j) is set to .true. if the $j-$th system is consistent and to .false. otherwise.

x is an INTENT(INOUT) rank-2 array of package type with extents ldx and 2*nrhs. It must be set so that, for j = 1,2,...,nrhs, x(i,j) holds the component of the right-hand side for variable i to the jth system. On exit, x(1:n,1:nrhs) holds the same solution as is returned by ma97_solve and, if flag_out(j)=.false., x(1:n,nrhs+j) holds the Fredholm alternative solution for the $j-$th system.

### 2.6.12 To form a matrix-vector or matrix-matrix product with $S^{-1}PL$ or $(S^{-1}PL)^{\dagger}$

In the indefinite case, after the call to ma97_factor, one or more calls of the following form may be made to calculate $Y = S^{-1}PLX$ or $Y = (S^{-1}PL)^{\dagger}X$.
To form a matrix-vector product with a triangular factor,

```
call ma97_lmultiply(trans,x1,y1,akeep,fkeep,control,info)
```

or, to form a matrix-matrix product,

```
call ma97_lmultiply(trans,k,x,ldx,y,ldy,akeep,fkeep,control,info)
```

trans is an INTENT(IN) scalar of type LOGICAL. If set to .true., the operation $Y = (S^{-1}PL)^{\dagger}X$ is performed. Otherwise, if set to .false., the operation $Y = S^{-1}PLX$ is performed.

k is an INTENT(IN) scalar of type INTEGER that holds the number columns in the matrices $X$ and $Y$. **Restriction:** k≥1.

x1 is an INTENT(IN) rank-1 array of package type and size n. On input it must contain the vector $X$.

x is an INTENT(IN) rank-2 array of package type with extents ldx and k. It must be set to contain the matrix $X$.

ldx is an INTENT(IN) scalar of type INTEGER that must be set to the first extent of the array x. **Restriction:** ldx≥n.

y1 is an INTENT(OUT) rank-1 array of package type and size n. On exit, it will be set to the requested matrix-vector product $Y$.

y is an INTENT(OUT) rank-2 array of package type with extents ldy and k. On exit, it will be set to the requested matrix-matrix product $Y$.

ldy is an INTENT(IN) scalar of type INTEGER that must be set to the first extent of the array y. **Restriction:** ldy≥n.

akeep, fkeep, control, info: see Section 2.6.6.

### 2.6.13 To solve $PLX = SB$ for sparse $B$

After the call to ma97_factor (or ma97_factor_solve), one or more calls of the following form may be made to solve $PLX = SB$ for a **single sparse right-hand side**

```
call ma97_sparse_fwd_solve(nbi,bindex,b,order,lflag,nxi,xindex,x, &
     akeep,fkeep,control,info)
```

nbi is an INTENT(IN) scalar of type INTEGER that must hold the number of nonzero entries in the right-hand side. **Restriction:** 1≤nbi≤n.

bindex is an INTENT(IN) rank-1 array of type INTEGER. The first nbi entries must hold the indices of the nonzero entries in the right-hand side.

b is an INTENT(IN) rank-1 array of package type and size n. If bindex(i)=k, b(k) must hold the k-th nonzero component of the right-hand side; other entries of b are not accessed.

order is an INTENT(IN) rank-1 array of type INTEGER and size n. It must be unchanged since the call to ma97_analyse.

lflag is an INTENT(INOUT) rank-1 array of type LOGICAL and size n. On entry, all the entries must be set by the user to .false.. On exit, if xindex(i)=k then lflag(k)=.true. and x(k) is nonzero. To reset flag(:) to .false., it is sufficient to consider only the entries xindex(1:nxi).

nxi is an INTENT(OUT) scalar of type INTEGER. On exit, nxi holds the number of nonzero entries in the solution.

xindex is an INTENT(OUT) rank-1 array of package type and size nxi (that is at most n). On exit, the first nxi entries hold the indices of the nonzero entries in the solution.

x is an INTENT(INOUT) rank-1 array of package type and size n. On entry, it must be set by the user to zero. On exit, if xindex(i)=k, x(k) holds the k-th nonzero component of the solution; all other entries of x are zero.

akeep, fkeep, control, info: see Section 2.6.6.

## 2.7 The derived types

### 2.7.1 The derived data type for holding control parameters

The derived data type ma97_control is used to hold controlling data. The components, which are automatically given default values in the definition of the type, are:

**Printing controls**

print_level is a scalar of type INTEGER that is used to controls the level of printing. The different levels are:

- $< 0$ No printing.
- $= 0$ Error and warning messages only.
- $= 1$ As 0, plus basic diagnostic printing.
- $> 1$ As 1, plus some additional diagnostic printing.

The default is print_level=0.

unit_diagnostics is a scalar of type INTEGER that holds the unit number for diagnostic printing. Printing is suppressed if unit_diagnostics$< 0$. The default is unit_diagnostics=6.

unit_error is a scalar of type INTEGER that holds the unit number for error messages. Printing of error messages is suppressed if unit_error$<0$. The default is unit_error=6.

unit_warning is a scalar of type INTEGER that holds the unit number for warning messages. Printing of warning messages is suppressed if unit_warning$<0$. The default is unit_warning=6.

**Controls used by** ma97_analyse **and** ma97_analyse_coord

ordering is a scalar of type INTEGER. If set to 0, the user must supply an elimination order in order; otherwise an elimination order will be computed by ma97_analyse or ma97_analyse_coord. The options are:

- 0 User-supplied ordering is used.
- 1 An approximate minimum degree (AMD) ordering is used.
- 2 A minimum degree ordering is used.
- 3 METIS ordering with default settings is used. Note that the user needs to supply the METIS library. If METIS is not supplied and this option is requested, the routine will return immediately with an error.
- 4 MA47 ordering for indefinite matrices is used.
- 5 A heuristic choice is made between AMD and METIS orderings assuming the factorization is to be run in **parallel**. If METIS is not available, AMD is used. The actual ordering chosen is indicated by the value of info%ordering on return from ma97_analyse or ma97_analyse_coord.

6 As 5 but assuming the factorization is to be run in **serial**.

7 A matching-based elimination ordering is computed using HSL_MC80. AMD is used on the compressed matrix. This option should only be chosen for indefinite systems. A scaling is also computed that may be passed to ma97_factor or ma97_factor_solve (see control%scaling below).

8 As 7 but METIS is used on the compressed matrix.

The default is ordering=5. **Restriction:** ordering=0, 1,..., 8.

nemin is a scalar of type INTEGER that controls node amalgamation. Two neighbours in the elimination tree are merged if they both involve fewer than nemin eliminations. The default is nemin=8. The default is used if nemin<1.

**Controls used by** ma97_factor

scaling is a scalar of type INTEGER that controls the use of scaling. The available options are:

$\leq$ 0 No scaling (scale optional argument not present), or user-supplied scaling (scale optional argument present).

= 1 Generate a scaling using a weighted bipartite matching using the package MC64.

= 2 Generate a scaling by applying the iterative method of the package MC77 for one iteration in the infinity norm and three iterations in the one norm.

= 3 A matching-based ordering has been generated during the analyse phase using control%ordering = 7 or 8. Use the scaling generated as a side-effect of this process. The scaling will be the same as that generated with control%scaling = 1 if matrix values have not changed. This option will generate an error if a matching-based ordering was not used.

$\geq$ 4 Generate a scaling by minimising the absolute sum of log values in the scaled matrix using the package MC30.

The default is scaling=0.

**Controls used by** ma97_factor **with** matrix_type=4, -4 **or** -5 **(*A* indefinite)**

action is a scalar of type default LOGICAL. If the matrix is found to be singular (has rank less than the number of non-empty rows), the computation continues after issuing a warning if action has the value .true. or terminates with an error if it has the value .false. The default is action=.true.

factor_min is a scalar of type INTEGER(long) that controls the use of parallelism within the factorization. Parallelism is only used if the predicted number of floating point operations (info%num_flops) is greater than or equal to factor_min. The default is factor_min= $2 \times 10^7$.

multiplier is a scalar of type REAL. To allow for delayed pivots, the arrays that store the factors and associated index lists are allocated to accommodate a matrix of order $s \times max(1,$multiplier$)$, where $s$ is the expected size of the factors without delays. If, during the factorization, this space is found to be too small, additional memory will be allocated dynamically. The default is multiplier= 1.1.

small is a scalar of type REAL. Any pivot whose modulus is less than small is treated as zero. The default in the double and double complex versions is small= $10^{-20}$, and in the single and single complex versions is small= $10^{-12}$.

u is a scalar of type REAL that holds the relative pivot tolerance $u$. The default in the double and double complex versions is u=0.01, and in the single and single complex versions is u=0.1. Values outside the range $[0,0.5]$ are treated as the closest value in that range.

**Controls used by** ma97_solve **and/or** ma97_solve_fredholm

consist_tol is a scalar of type REAL that holds the tolerance used to determine if a system is inconsistent in ma97_solve_fredholm. The default is consist_tol=epsilon(), the smallest quantity for the package type such that $1 + \texttt{epsilon} \neq 1$.

solve_mf is a scalar of type LOGICAL. If solve_mf=.true. a multifronal-style forward solve is used. Otherwise a supernodal-style solve is used. The supernodal solve does not use parallelism during the forward solve and typically performs best on small problems, with the multifrontal solve performing best on large problems. If the user wishes to make more than one call to ma97_solve, we recommend comparing the solve time with both solve_mf=.true. and solve_mf=.false.. The default value is solve_mf=.false..

solve_blas3 is a scalar of type LOGICAL that controls whether level 2 (solve_blas3=.false.) or level 3 (solve_blas3=.true.) BLAS are used in the case of a single right-hand side solution. On larger problems the level 3 BLAS can often out perform the level 2 BLAS. The default is solve_blas3=.false..

solve_min is a scalar of type INTEGER(long) that controls the use of parallelism within the solve. Parallelism is only used if the number of entries in $L$ (info%num_factor) is greater than or equal to solve_min. The default is solve_min=100000.

### 2.7.2 The derived data type for holding information

The derived data type ma97_info is used to hold parameters that give information about the progress and needs of the algorithm. The components of ma97_info (in alphabetical order) are:

flag is a scalar of type INTEGER that gives the exit status of the algorithm (details in Section 2.8).

flag68 is a scalar of type INTEGER. On exit from ma97_analyse or ma97_analyse_coord, it holds the exit status from HSL_MC68.

flag77 is a scalar of type INTEGER. On exit from ma97_factor or ma97_factor_solve, it holds the exit status from MC77.

matrix_dup is a scalar of type INTEGER. On exit from ma97_analyse with check set to .true. or from ma97_analyse_coord, it holds the number of duplicate entries that were found and summed.

matrix_missing_diag is a scalar of type INTEGER. On exit from ma97_analyse with check set to .true., or from ma97_analyse_coord, it holds the number of diagonal entries without an explicitly provided value.

matrix_outrange is a scalar of type INTEGER. On exit from ma97_analyse with check set to .true. or from ma97_analyse_coord, it holds the number of out-of-range entries that were found and discarded.

matrix_rank is scalar of type INTEGER. On exit from ma97_analyse and ma97_analyse_coord, it holds the structural rank of $A$, if available (otherwise, it is set to n). On exit from ma97_factor or ma97_factor_solve, it holds the computed rank of the factorized matrix.

maxdepth is a scalar of type INTEGER. On exit from ma97_analyse or ma97_analyse_coord, it holds the maximum depth of the assembly tree.

maxfront is a scalar of type INTEGER. On exit from ma97_analyse or ma97_analyse_coord, it holds the maximum front size in the positive-definite case (or in the indefinite case with the same pivot sequence). On exit from ma97_factor or ma97_factor_solve, it holds the maximum front size.

num_delay is scalar of type INTEGER. On exit from ma97_factor or ma97_factor_solve, it holds the number of eliminations that were delayed, that is, the total number of fully-summed variables that were passed to the father node because of stability considerations. If a variable is passed further up the tree, it will be counted again.

num_factor is scalar of type INTEGER(long). On exit from ma97_analyse or ma97_analyse_coord, it holds the number of entries that will be in the factor $L$ in the positive-definite case (or in the indefinite case with the same pivot sequence). On exit from ma97_factor or ma97_factor_solve, it holds the actual number of entries in the factor $L$. In the indefinite case, 2n entries of $D^{-1}$ are also held.

num_flops is scalar of type INTEGER(long). On exit from ma97_analyse or ma97_analyse_coord, it holds the number of floating-point operations that will be needed to perform the factorization in the positive-definite case (or in the indefinite case with the same pivot sequence). On exit from ma97_factor or ma97_factor_solve, it holds the number of floating-point operations performed.

num_neg is a scalar of type INTEGER. On exit from ma97_factor or ma97_factor_solve, it holds the number of negative eigenvalues of the matrix $D$.

num_sup is a scalar of type INTEGER. On exit from ma97_analyse or ma97_analyse_coord, it holds the number of supernodes in the problem.

num_two is scalar of type INTEGER. On exit from ma97_factor and ma97_factor_solve, it holds the number of $2 \times 2$ pivots used by the factorization, that is, the number of $2 \times 2$ blocks in $D$.

ordering is a scalar of type INTEGER. On exit from ma97_analyse or ma97_analyse_coord, it indicates the ordering method chosen. Values have the same meanings as in the context of control%ordering.

stat is a scalar of type INTEGER. In the event of an allocation or deallocation error, it holds the Fortran stat parameter if it is available (and is set to 0 otherwise).

### 2.8 Warning and error messages

A successful return from a subroutine in the package is indicated by info%flag having the value zero. A negative value is associated with an error message that by default will be output on unit control%unit_error.

Possible negative values are:

−1 An error has been made in the sequence of calls (this includes calling a subroutine after an error that cannot be recovered from).

−2 Returned by ma97_analyse and ma97_analyse_coord if n<0. Also returned by ma97_analyse_coord if ne<1.

−3 Returned by ma97_analyse if there is an error in ptr.

−4 Returned by ma97_analyse if all the variable indices in one or more columns are out-of-range. Also returned by ma97_analyse_coord if all entries are out-of-range.

−5 Returned by ma97_factor and ma97_factor_solve if matrix_type is out-of-range. The user may reset matrix_type and recall ma97_factor or ma97_factor_solve.

−6 Returned by ma97_factor and ma97_factor_solve if matrix_type=-3 or -4 ($A$ is Hermitian) and one or more of the diagonal entries is not real.

−7 Returned by ma97_factor and ma97_factor_solve if matrix_type=4, -4 or -5 and control%action = .false. when the matrix is found to be singular. The user may reset the matrix values in val and recall ma97_factor or ma97_factor_solve.

−8 Returned by ma97_factor and ma97_factor_solve if matrix_type=3 or -3 and the matrix is found to be not positive definite. This may be because the scaling MC64 found the matrix to be singular. The user may reset the matrix values in val and recall ma97_factor or ma97_factor_solve.

−9 Returned by ma97_factor if IEEE infinities found in the reduced matrix, probably caused by control%small or control%u having too small a value. The user may reset control%small and/or control%u or may reset the matrix values in val and recall ma97_factor or ma97_factor_solve.

−10 Returned by ma97_factor and ma97_factor_solve if ma97_analyse was called with check set to .false. but ptr and/or row is not present.

−11 Returned by ma97_analyse and ma97_analyse_coord if control%ordering is out-of-range, or control%ordering=0 and the user has either failed to provide an elimination order or an error has been found in the user-supplied elimination order (held in order).

−12 Returned by ma97_factor_solve, ma97_solve, ma97_solve_fredholm, and ma97_lmultiply if there is an error in the size of array x (that is, ldx<n or nrhs<1). The user may reset ldx and/or nrhs and recall ma97_factor_solve, , ma97_solve, ma97_solve_fredholm, or ma97_lmultiply. This error is also returned by ma97_lmultiply if there is an error in the size of array y (that is, ldy<n).

−13 Returned by ma97_solve if job is out-of-range. The user may reset job and recall ma97_solve.

−14 Returned by ma97_enquire_posdef if matrix_type=4, -4 or -5 on the last call to ma97_factor or ma97_factor_solve.

−15 Returned by ma97_enquire_indef if matrix_type=3 or -3 on the last call to ma97_factor or ma97_factor_solve.

−16 Allocation error. If available, the stat parameter is returned in info%stat. The user may wish to try the more memory conservative codes HSL_MA86 or HSL_MA77.

−17 Returned by ma97_analyse and ma97_analyse_coord if METIS ordering was requested but METIS is not available.

−18 Returned by ma97_analyse and ma97_analyse_coord if there is an unexpected error from HSL_MC68. The user is advised to ensure that if ma97_analyse was called, check was set to .true.. Further information may be provided by info%flag68.

−19 Returned by ma97_factor and ma97_factor_solve if there is an unexpected error from MC77. The user is advised to ensure that if ma97_analyse was called, check was set to .true.. Further information may be provided by info%flag77.

−20 Returned by ma97_analyse and ma97_analyse_coord if control%ordering=7 or 8 but val is not present.

−21 Returned by ma97_factorise if control%scaling=3 but a matching based ordering was not used during the call to ma97_analyse or ma97_analyse_coord (i.e. was called with control%ordering≠7 or 8).

−22 Returned by ma97_sparse_fwd_solve if nbi is out of range. The user may reset nbi and recall.

A positive value of info%flag is used to warn the user that the input matrix data may be faulty or that the subroutine cannot guarantee the solution obtained. Possible values are:

+1 Returned by ma97_analyse and ma97_analyse_coord if out-of-range variable indices found. Any such entries are ignored and the computation continues. info%matrix_outrange is set to the number of such entries.

+2 Returned by ma97_analyse and ma97_analyse_coord if duplicated indices found. Duplicates are recorded and the corresponding entries are summed. info%matrix_dup is set to the number of such entries.

+3 Returned by ma97_analyse and ma97_analyse_coord if both out-of-range and duplicated variable indices found.

+4 Returned by ma97_analyse and ma97_analyse_coord if one and more diagonal entries of $A$ is missing.

+5 Returned by ma97_analyse and ma97_analyse_coord if one and more diagonal entries of $A$ is missing and out-of-range and/or duplicated variable indices have been found.

+6 Returned by ma97_analyse and ma97_analyse_coord if $A$ is found be (structurally) singular. This will overwrite any of the above warnings.

+7 Returned by ma97_factor and ma97_factor_solve if control%action is set to .true. and the matrix is found to be (structurally or numerically) singular.

+8 Returned by ma97_factor and ma97_factor_solve if control%ordering=7 or 8 (i.e. a matching-based ordering was used) but the associated scaling was not (i.e. control%scaling$\neq$ 3).

## 3 GENERAL INFORMATION

**Workspace:** Provided automatically by the module.

**Other routines called directly:** MC30, HSL_MC34, HSL_MC64, HSL_MC68 (optionally using METIS), HSL_MC69, MC77, HSL_MC78, HSL_MC80, _axpy, _gemm, _gemv, _nrm2, _potrf, _swap, _syrk, _trmv, _trmm, _trsm, _trsv.

**Input/output:** Output is provided under the control of control%print_level. In the event of an error, diagnostic messages are printed. The output units for these messages are respectively controlled by control%unit_err, control%unit_warning and control%unit_diagnostics (see Section 2.7.1).

**Restrictions:** n$\geq$0; ne$\geq$1; nrhs$\geq$1; ldx$\geq$n; ldy$\geq$n;
control%ordering=0, 1, 2, 3, 4, 5, 6, 7, 8;
matrix_type = 3, 4, -3, -4, or -5;
job = 1, 2, 3, or 4; 1$\leq$nbi$\leq$n.

**Portability:** Fortran 95, plus allocatable dummy arguments and allocatable components of derived types. OpenMP 3.0 or above for (optional) parallel usage.

**Changes from Version** 1
Version 2 offers the option of computing a matching-based elimination ordering. This requires the user to supply the numerical values of the matrix on the call to the analyse phase. If the user wishes to factorize another matrix with the same sparsity pattern but different numerical values, it may be necessary to recall the analyse phase. A matching-based elimination ordering may be a good choice for tough indefinite systems.

## 4 METHOD

ma97_analyse and ma97_analyse_coord
If check is set to .true. on the call to ma97_analyse or if ma97_analyse_coord is called, the HSL package HSL_MC69 is used to check the matrix data. The cleaned integer matrix data (duplicates are summed and out-of-range indices discarded) is stored in akeep. The use of checking is optional on a call to ma97_analyse as it incurs both time and memory overheads. Some form of checking is recommended since the behaviour of the other routines in the

package is unpredictable if duplicates and/or out-of-range variable indices are entered. Calling the HSL_MC69 routine mc69_verify offers an alternative that can be used for debugging purposes.

If the user has supplied a pivot order it is checked for errors. Otherwise, a pivot order is generated using HSL_MC68, or if a matching-based ordering is requested, HSL_MC80. The pivot order is used to construct the assembly tree using HSL_MC78.

On exit, order is set so that order(i) holds the position at which variable i is eliminated. If a user order was supplied, this order may differ, but will be equivalent in terms of fill-in to that provided.

If a matching-based ordering is requested and scale is present, on exit, scale contains scaling factors computed by MC64. These may be passed unchanged to ma97_factor and ma97_factor_solve.

ma97_factor and ma97_factor_solve

ma97_factor and ma97_factor_solve optionally compute a scaling and then perform the numerical factorization. The user must specify whether or not the matrix is positive definite. If matrix_type is set to 3 or -3, no pivoting is performed. As a result the computation will terminate with an error if a non-positive pivot is encountered.

The factorization uses the assembly tree that was set up by the analyse phase. If running on a single thread (or if there is insufficient work available to justify running in parallel), the nodes of the tree are iterated over in a post-order.

At a node, the contributions from the children relating to those columns that are fully summed at this node are first assembled. A dense partial factorization is then performed on these columns. In the positive-definite case, LAPACK's _potrf (or _herk for Hermitian matrices) is used. In the indefinite case, an algorithm based on the same pivoting algorithm as HSL_MA64 is used.

The generated element is calculated by first forming the outer product of the fully summed columns' uneliminated rows. The contributions from the children are then added, and the stack memory used by the children is freed. As this involves copying from one stacked contribution to another, two separate stacks are used to do this.

If a pivot candidate does not pass pivot tests at a given node, it is delayed to the parent node where additional eliminations may make the pivot feasible. This results in the generation of additional fill-in and floating-point operations, and may result in additional memory allocations being required.

In parallel computation, we exploit two levels of parallelism using OpenMP tasks. In tree-level parallelism, different subtrees are factorized in independent tasks. To ensure results are bit-compatible regardless of the number of threads used, the assembly order of the children is fixed at assembly time. In node-level parallelism, the operations forming the outer-product in both the dense factorization kernel and the calculation of the generated element are broken into multiple tasks. Bit-compatibility is ensured in this case by using a data-parallel approach so each individual sum is effectively calculated in serial.

If ma97_factor_solve is called, the forward substitutions are performed as the factor entries are generated. Once the factorization is complete, the back substitutions are performed by an internal call to ma97_solve with job = 3 (positive-definite case) or job = 4 (indefinite case).

ma97_solve

Having checked the user's data, ma97_solve performs a forward substitution followed by a combined diagonal solve and back substitution (unless only one of these is requested).

In a supernodal solve updates are done directly into the right-hand side vector and do not readily admit bit-compatible parallelism in the forward substitution. In the multifrontal solve updates are passed up the tree utilising a stack. This allows parallelism to be implemented but can be slower than the supernodal solve on small problems. Regardless of whether a supernodal or multifrontal solve is chosen the same backwards solve is used that works directly on the right-hand side vectors. Due to the differing data dependencies from the forward substitution a bit-compatible parallel solve is possible.

The matrix factor must be accessed once for the forward substitution and once for the back substitution. This is independent of the number of right-hand sides so that solving for several right-hand sides at once is significantly faster

than repeatedly solving for a single right-hand side.

## References:

[1] J.D. Hogg and J.A. Scott. (2011). HSL_MA97: a bit-compatible multifrontal code for sparse symmetric systems. RAL Technical Report. RAL-TR-2011-024.

# 5 EXAMPLE OF USE

### 5.1 First example: sparse column entry

Suppose we wish to factorize the matrix

$$A = \begin{pmatrix} 2. & 1. & & & \\ 1. & 4. & 1. & & 1. \\ & 1. & 3. & 2. & \\ & & 2. & 0. & \\ & 1. & & & 2. \end{pmatrix}$$

and then solve for the right-hand side

$$B = \begin{pmatrix} 4. \\ 12. \\ 10. \\ 4. \\ 4. \end{pmatrix}$$

.

The following code may be used. Note that, in this example, it would be more efficient to pass the right-hand side to ma97_factor_solve; here our aim is to illustrate calling ma97_solve after ma97_factor.

```
! Simple code to illustrate use of hsl_ma97
program hsl_ma97ds
   use hsl_ma97_double
   implicit none

   ! Derived types
   type (ma97_akeep)   :: akeep
   type (ma97_fkeep)   :: fkeep
   type (ma97_control) :: control
   type (ma97_info)    :: info

   ! Parameters
   integer, parameter :: wp = kind(0.0d0)

   integer, dimension (:),  allocatable :: ptr
   integer, dimension (:),  allocatable :: piv_order
   integer, dimension (:),  allocatable :: row
   real(wp), dimension (:), allocatable :: val
   real(wp), dimension (:), allocatable :: x
   real(wp), dimension (:,:), allocatable :: d

   integer :: matrix_type,n,ne
```

```
   logical :: check

   ! Read in the order n of the matrix and number of entries in lower triangle
   read (*,*) n,ne

   ! Allocate arrays for matrix data and arrays for hsl_ma97
   allocate (ptr(n+1),row(ne),val(ne))
   allocate (x(n),d(2,n))

   read (*,*) ptr(1:n+1)
   read (*,*) row(1:ne)
   read (*,*) val(1:ne)

   ! Perform analyse and factorise with data checking
   check = .true.
   call ma97_analyse(check,n,ptr,row,akeep,control,info)
   if (info%flag < 0) go to 100
   matrix_type = 4 ! Real, symmetric indefinite
   call ma97_factor(matrix_type,val,akeep,fkeep,control,info)
   if (info%flag < 0) go to 100

   ! Read in the right-hand side and copy into resid.
   read (*,*) x(1:n)

   ! Solve
   call ma97_solve(x,akeep,fkeep,control,info)
   if (info%flag < 0) go to 100
   write (*,'(/a,/,(3es18.10))') ' The computed solution is:', x(1:n)

   ! Determine the pivot order used
   allocate (piv_order(1:n))
   call ma97_enquire_indef(piv_order, d, akeep, fkeep, control, info)
   write (6,*) 'piv_order', piv_order(1:n)

   100 continue
   call ma97_finalise(akeep, fkeep)

end program hsl_ma97ds
```

with the following data:

```
  5  8
  1  3  6  8  8  9
  1  2  2  5  3  4  3  5
  2. 1. 4. 1. 1. 2. 3. 2.
  4. 12. 10. 4. 4.
```

This produces the following output:

```
 The computed solution is:
    1.000     2.000     2.000     1.000     1.000
```

### 5.2 Second example: coordinate entry, refactorization, factor_solve

Suppose we wish to factorize the matrix

$$A = \begin{pmatrix} 1. & -3. & & 1. & \\ -3. & -5. & 6. & & 4. \\ & 6. & & 2. & \\ 1. & & 2. & 3. & \\ & 4. & & & 1. \end{pmatrix}$$

and then solve for the right-hand sides

$$B = \begin{pmatrix} -1. & -5. \\ 25. & -40. \\ 20. & 8. \\ 19. & -8. \\ 13. & -1. \end{pmatrix}$$

. Suppose we then wish to solve the following system with the same pattern in a single call:

$$\begin{pmatrix} 2. & 1. & & 7. & \\ 1. & 1. & 8. & & 2. \\ & 8. & & 1. & \\ 7. & & 1. & 8. & \\ & 2. & & & 8. \end{pmatrix} \mathbf{x} = \begin{pmatrix} 8. \\ 89. \\ 40. \\ 37. \\ 42. \end{pmatrix}$$

The following code may be used.

```
! Simple code to illustrate use of hsl_ma97
program hsl_ma97ds1
   use hsl_ma97_double
   implicit none

   ! Derived types
   type (ma97_akeep)   :: akeep
   type (ma97_fkeep)   :: fkeep
   type (ma97_control) :: control
   type (ma97_info)    :: info

   ! Parameters
   integer, parameter :: wp = kind(0.0d0)

   integer, dimension (:),  allocatable :: row
   integer, dimension (:),  allocatable :: col
   real(wp), dimension (:), allocatable :: val
   real(wp), dimension (:,:), allocatable :: x

   integer :: matrix_type,n,ne

   ! Read in the order n of the matrix and number of entries in lower triangle
   read (*,*) n,ne

   ! Allocate arrays for matrix data and arrays for hsl_ma97
```

```
   allocate (row(ne),col(ne),val(ne))
   allocate (x(n,2))

   read (*,*) row(1:ne)
   read (*,*) col(1:ne)
   read (*,*) val(1:ne)

   ! Perform analyse and factorise with coordinate input
   call ma97_analyse_coord(n,ne,row,col,akeep,control,info)
   if (info%flag < 0) go to 100
   matrix_type = 4 ! Real, symmetric indefinite
   call ma97_factor(matrix_type,val,akeep,fkeep,control,info)
   if (info%flag < 0) go to 100

   ! Read in the right-hand side
   read (*,*) x(1:n,1:2)

   ! Solve
   call ma97_solve(2,x,n,akeep,fkeep,control,info)
   if (info%flag < 0) go to 100
   write (*,'(/a,/,(3es18.10))') ' The computed solution is:', x(1:n,1)
   write (*,'(3es18.10)') x(1:n,2)

   ! Read values of second matrix with same pattern
   read (*,*) val(1:ne)

   ! Read another right hand side
   read (*,*) x(1:n,1)

   ! Perform combined factor and solve
   call ma97_factor_solve(matrix_type,val,x(1:n,1),akeep,fkeep,control,info)
   write (*,'(/a,/,(3es18.10))') ' Next solution is:', x(1:n,1)

   100 continue
   call ma97_finalise(akeep, fkeep)

end program hsl_ma97ds1
```

with the following data:

```
5 9
1   2   1   2   3   5   4   4   5
1   1   4   2   2   2   4   3   5
1. -3.  1. -5.  6.  4.  3.  2.  1.
-1.  25.  20.  19.  13.
-5. -40.   8.  -8.  -1.
2.  1.  7.  1.  8.  2.  8.  1.  8.
16.5 89.0 40.5 41.0 42.0
```

This produces the following output:

```
 Warning from ma97_analyse_coord. Warning flag =   4
 one or more diagonal entries is missing

 The computed solution is:
  1.0000000000E+00  2.0000000000E+00  3.0000000000E+00
  4.0000000000E+00  5.0000000000E+00
 -3.0000000000E+00  1.0000000000E+00 -4.0000000000E+00
  1.0000000000E+00 -5.0000000000E+00

 Next solution is:
  4.0000000000E+00  5.0000000000E+00  9.0000000000E+00
  5.0000000000E-01  4.0000000000E+00
```

Note that the warning is entirely innocous and is merely due to the absence of a non-zero in the diagonal $(3,3)$ position.