Science and Technology Facilities Council

HSL_MA74

PACKAGE SPECIFICATION HSL

1 SUMMARY

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

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

$$PAQ = \begin{pmatrix} L_1 & 0 \\ L_2 & I \end{pmatrix} \begin{pmatrix} D_1 & 0 \\ 0 & A_2 \end{pmatrix} \begin{pmatrix} U_1 & U_2 \\ 0 & I \end{pmatrix},$$

where P and Q are permutation matrices, L_1 and U_1 are unit lower and unit upper triangular matrices of order q, and D_1 is diagonal of order q. The permutation matrices P and Q are of the form

$$P = \begin{pmatrix} P_1 & 0 \\ 0 & I \end{pmatrix}, \quad Q = \begin{pmatrix} Q_1 & 0 \\ 0 & I \end{pmatrix},$$

where P_1 and Q_1 are of order p.

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

$$\begin{pmatrix} L_1 & 0 \\ L_2 & I \end{pmatrix} X = B, \quad \begin{pmatrix} D_1 & 0 \\ 0 & I \end{pmatrix} X = B, \quad \begin{pmatrix} D_1 & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} U_1 & U_2 \\ 0 & I \end{pmatrix} X = B, \quad \text{and} \quad \begin{pmatrix} U_1 & U_2 \\ 0 & I \end{pmatrix} X = B,$$

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

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

$$\begin{pmatrix} U_1^T & 0 \\ U_2^T & I \end{pmatrix} X = B, \quad \begin{pmatrix} D_1 & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} L_1^T & L_2^T \\ 0 & I \end{pmatrix} X = B, \quad \text{and} \quad \begin{pmatrix} L_1^T & L_2^T \\ 0 & I \end{pmatrix} X = B,$$

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

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

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

ATTRIBUTES — Version: 1.5.0 (3 January 2012) **Types:** Real (single, double). **Calls:** _GEMM, _GEMV, _GER, I_AMAX, _SWAP, _TRSM, _TRSV. **Original date:** May 2007. **Origin:** J.K. Reid and J.A. Scott, Rutherford Appleton Laboratory. **Language:** Fortran 95. **Remark:** The development of HSL_MA74 was supported by EPSRC grant GR/S42170.

2 HOW TO USE THE PACKAGE

2.1 Introduction

Access to the package requires a USE statement

Single precision version

```
USE HSL_MA74_single
```

Double precision version

```
USE HSL_MA74_double
```

If it is required to use more than one module at the same time, the derived types (Section 2.2) and any MA74 procedures invoked must be renamed in one of the USE statements.

In HSL_MA74_single, all reals are default reals. In HSL_MA74_double, all reals are double precision reals. All integers are default integers.

The following subroutines are available to the user:

- MA74_factor partially factorizes an unsymmetric matrix.
- MA74_solveL1 solves $\begin{pmatrix} L_1 & 0 \\ L_2 & I \end{pmatrix} x = b$ and MA74_solveL2 solves $\begin{pmatrix} L_1 & 0 \\ L_2 & I \end{pmatrix} X = B$.
- MA74_solveD1 solves $\begin{pmatrix} D_1 & 0 \\ 0 & I \end{pmatrix} x = b$ and MA74_solveD2 solves $\begin{pmatrix} D_1 & 0 \\ 0 & I \end{pmatrix} X = B$.
- MA74_solveDU1 solves $\begin{pmatrix} D_1 & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} U_1 & U_2 \\ 0 & I \end{pmatrix} x = b$ and
  MA74_solveDU2 solves $\begin{pmatrix} D_1 & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} U_1 & U_2 \\ 0 & I \end{pmatrix} X = B$.
- MA74_solveU1 solves $\begin{pmatrix} U_1 & U_2 \\ 0 & I \end{pmatrix} x = b$ and MA74_solveU2 solves $\begin{pmatrix} U_1 & U_2 \\ 0 & I \end{pmatrix} X = B$.
- MA74_solveUT1 solves $\begin{pmatrix} U_1^T & 0 \\ U_2^T & I \end{pmatrix} x = b$ and MA74_solveUT2 solves $\begin{pmatrix} U_1^T & 0 \\ U_2^T & I \end{pmatrix} X = B$.
- MA74_solveDL1 solves $\begin{pmatrix} D_1 & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} L_1^T & L_2^T \\ 0 & I \end{pmatrix} x = b$ and
  MA74_solveDL2 solves $\begin{pmatrix} D_1 & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} L_1^T & L_2^T \\ 0 & I \end{pmatrix} X = B$.
- MA74_solveLT1 solves $\begin{pmatrix} L_1^T & L_2^T \\ 0 & I \end{pmatrix} x = b$ and MA74_solveU2 solves $\begin{pmatrix} L_1^T & L_2^T \\ 0 & I \end{pmatrix} X = B$.

2.2 The derived data types

For each problem, the user must employ the derived types defined by the module to declare scalars of the types MA74_control and MA74_info. The following pseudocode illustrates this.

```
use HSL_MA74_double
...
type (MA74_control) :: control
type (MA74_info) :: info
...
```

The components of MA74_control and MA74_info are explained in Sections 2.4 and 2.5.

2.3 Argument lists and calling sequences

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**.

2.3.1 Partially factorize a matrix

```
call MA74_factor(n,p,nb,ap,ldap,q,pperm,qperm,control,info[,s])
```

n is a scalar INTENT(IN) argument of type INTEGER that must be set to the matrix order. **Restriction:** n $\geq$ 0.

p is a scalar INTENT(IN) argument of type INTEGER that must be set to the number of leading rows and columns on which eliminations are allowed. **Restriction:** 0 $\leq$ p $\leq$ n.

nb is a scalar INTENT(IN) argument of type INTEGER that must be set to the block size used during the factorization. Whenever nb pivots have been chosen, a Schur complement update is performed. Section 2.7 contains a discussion of suitable values. **Restriction:** nb $\geq$ 1.

ap is a rank-2 array of INTENT(INOUT) and type REAL. Its first extent is ldap and its second extent is at least n. It holds the matrix A on entry and is overwritten by the factorized matrix on return.

ldap is a scalar INTENT(IN) argument of type INTEGER that must be set to the leading dimension of ap. **Restriction:** ldap $\geq$ n.

q is a scalar INTENT(OUT) argument of type INTEGER. On successful exit, q is the number of pivots chosen.

pperm is an array of shape p, INTENT(OUT) and type INTEGER. On successful exit, pperm(i) holds the index of the row of A that is permuted to row i, i = 1,..., p.

qperm is an array of shape p, INTENT(OUT) and type INTEGER. On successful exit, qperm(j) holds the index of the column of A that is permuted to column j, j = 1,..., p.

control is a scalar INTENT(OUT) argument of type MA74_control. Its components control the actions of the subroutine, as explained in Section 2.4.

info is a scalar INTENT(OUT) argument of type MA74_info. Its components provide information about the execution of the subroutine, as explained in Section 2.5.

s is an optional scalar INTENT(IN) argument of type INTEGER. If present and 0 $<$ s $<$ p, the search for a pivot is started in column s+1. Otherwise, the search starts from column 1.

2.3.2 One partial solution

```
call MA74_solveL1(n,q,nb,ap,ldap,b,flag)
call MA74_solveD1(n,q,nb,ap,ldap,b,flag)
call MA74_solveDU1(n,q,nb,ap,ldap,b,flag)
call MA74_solveU1(n,q,nb,ap,ldap,b,flag)
call MA74_solveUT1(n,q,nb,ap,ldap,b,flag)
call MA74_solveDLT1(n,q,nb,ap,ldap,b,flag)
call MA74_solveLT1(n,q,nb,ap,ldap,b,flag)
```

n,q,nb are scalars of INTENT(IN) and type INTEGER whose values must be unchanged since the call to MA74_factor.

ap is an array of shape (n,n), INTENT(IN) and type REAL

ap is arank-2 array of INTENT(IN) and type REAL. Its first extent is ldap and its second extent is at least n. It must hold the factorized matrix, as returned by MA74_factor.

ldap is a scalar of INTENT(IN) and type INTEGER whose value must be unchanged since the call to MA74_factor.

b is an array of shape n, INTENT(INOUT) and type REAL. It holds the vector b on entry and is overwritten by the vector x on return.

flag is a scalar INTENT(OUT) argument of INTEGER. On successful return, it has the value zero. After an unsuccessful return, it has a negative value; see Section 2.6.

2.3.3 One or more partial solutions

```
call MA74_solveL2(n,q,nb,nrhs,ap,ldap,b,ldb,flag)
call MA74_solveD2(n,q,nb,nrhs,ap,ldap,b,ldb,flag)
call MA74_solveDU2(n,q,nb,nrhs,ap,ldap,b,ldb,flag)
call MA74_solveU2(n,q,nb,nrhs,ap,ldap,b,ldb,flag)
call MA74_solveUT2(n,q,nb,nrhs,ap,ldap,b,ldb,flag)
call MA74_solveDLT2(n,q,nb,nrhs,ap,ldap,b,ldb,flag)
call MA74_solveLT2(n,q,nb,nrhs,ap,ldap,b,ldb,flag)
```

n,q,nb are scalars of INTENT(IN) and type INTEGER whose values must be unchanged since the call to MA74_factor.

nrhs is a scalar of INTENT(IN) and type INTEGER that must hold the number of right-hand sides. **Restriction:** nrhs $\geq$ 0.

ap is arank-2 array of INTENT(IN) and type REAL. Its first extent is ldap and its second extent is at least n. It must hold the factorized matrix, as returned by MA74_factor.

ldap is a scalar of INTENT(IN) and type INTEGER whose value must be unchanged since the call to MA74_factor.

b is arank-2 array of INTENT(INOUT) and type REAL. Its first extent is ldb and its second extent is at least nrhs. It holds the matrix B on entry and is overwritten by the matrix X on return.

ldb is a scalar of INTENT(IN) and type INTEGER that must hold the first extent of the array b. **Restriction:** ldb $\geq$ n.

flag is a scalar INTENT(OUT) argument of INTEGER. On successful return, it has the value zero. After an unsuccessful return, it has a negative value; see Section 2.6.

2.4 The derived data type for holding control parameters

The derived data type MA74_control is used to hold controlling data for MA74_factor. The components, which are automatically given default values in the definition of the type, are:

pivoting is a scalar of type INTEGER with default value 1 that controls the pivoting. Possible values are:

1 : threshold partial pivoting.

2 : threshold diagonal pivoting.

3 : threshold rook pivoting.

Restriction: pivoting = 1, 2 or 3.

small is a scalar of type REAL with default value 1×10^{-20}. If, during the factorization, all the entries in a column (or row) of the reduced matrix are of modulus less than or equal to small, all the entries in the column (or row) are replaced by zero. Every pivot (nonzero entry of D) must also be of absolute value greater than the absolute value of small.

static is a scalar of type REAL with default value 0.0. If static is positive, pivots that do not satisfy the threshold criteria may be selected and small pivots may be replaced by static until all p eliminations are completed; in this case, the factorization may be inaccurate. See Section 4 for details. **Restriction:** static = 0.0 or static $\geq$ abs(small).

u is a scalar of type REAL and default value 0.01 that holds the pivoting threshold parameter. Values of u that are less than 0.0 are treated as 0.0 and values greater than 1.0 are treated as 1.0. An entry in the leading p rows and columns is normally only considered suitable for use as a pivot if it is of absolute value at least as large as u times the entry of largest absolute value in its column (and its row, if control%pivoting = 3). However, if static is positive and fewer than p pivots can be chosen that satisfy this threshold criteria, pivots are chosen that come closest to satisfying it.

2.5 The derived data type for holding information

The derived data type MA74_info is used to hold parameters that give information about the factorization. The components of MA74_info that are set by MA74_factor are:

detlog is a scalar of type REAL. On successful exit, detlog holds the natural logarithm of the absolute value of the determinant of D_1 or zero if the determinant is zero.

detsign is a scalar of type INTEGER. On successful exit, it holds the product of the sign of the determinant of P, the sign of the determinant of D, and the sign of the determinant of Q, or is set to zero if the determinant of D is zero.

flag is a scalar of type INTEGER that gives the exit status. An unsuccessful call is flagged with a negative value as follows:

- -1 n $<$ 0.
- -2 p $<$ 0.
- -3 p $>$ n.
- -4 nb $<$ 1.
- -10 control%static $<$ abs(control%small) and control%static $\neq$ 0.0.
- -11 Value of control%pivoting is not valid.
- -12 ldap $<$ n.
- -13 control%pivoting $=$ 2, control%u $\leq$ 0.0, control%static $=$ 0.0 and at least one pivot candidate is of absolute value less than control%small.
- -14 IEEE infinities found in the reduced matrix, probably caused by control%small or control%u having too small a value.

num_diag is a scalar of type INTEGER. On successful exit, it holds the number of pivots that were chosen from the diagonal.

num_nothresh is a scalar of type INTEGER. On successful exit, it holds the number of pivots that did not satisfy the threshold criteria based on the user-supplied value of control%u.

num_perturbed is a scalar of type INTEGER. On successful exit, it holds the number of pivots that were replaced by control%static.

num_zero is scalar of type INTEGER. On successful exit, it holds the number of zeros on the diagonal of D_1.

usmall is a scalar of type REAL. On successful exit with $q = p$ and num_perturbed = 0, usmall holds the threshold parameter that was used. On successful exit with $q < p$, usmall holds the value of the threshold parameter control%u that would have allowed another pivot to have been chosen (control%pivoting = 1 or 2 only).

2.6 Errors from solve procedures

A successful return from one of the solve procedures is indicated by flag having the value zero. A negative value is associated with an error as follows:

-1 n $<$ 0.
-4 nb $<$ 1.
-5 nrhs $<$ 0.
-6 ldb $<$ n.
-8 q $<$ 0.
-9 q $>$ n.
-12 ldap $<$ n.

2.7 Choice of the block size

The choice of the best block size nb is machine dependent. The value that allows a full matrix of size nb to fit in the level-1 cache is usually good, but a larger value may give better performance if n is large because of the influence of the level-2 cache. For Cholesky factorizations, Andersen, Gunnels, Gustavson, Reid, and Wasniewski (ACM Trans. on Math. Software, **31**, 2005, 201–227) experimented with the values 40, 72, 100, 200 and found that the best value rose with n but that the following single values were adequate: 40 for the Intel Pentium III; 100 for the IBM Power4 and SGI Origin 200; and 200 for the Alpha EV6, SUN Ultra III, and HP Itanium 2. If performance is critical, we recommend that different values of nb be tried.

3 GENERAL INFORMATION

Other routines called directly: BLAS routines _GEMM, _GEMV, _GER, I_AMAX, _SWAP, _TRSM, _TRSV.

Restrictions: n≥0; 0≤p≤n;
0≤q≤n (solve routines); nb≥1;
nrhs≥1; ldap≥n;
ldb≥n;
control%pivoting = 1, 2, or 3; control%static = 0.0 or control%static ≥ abs(control%small).

4 METHOD

The input parameters are first checked for errors and the entries of info are initialised. If s is present and less than p, each column i is swapped with column p-i+1 ($1 \leq \mathtt{i} \leq min(\mathtt{s}, \mathtt{p}-\mathtt{s})$). Columns 1 to p are then searched cyclically for a pivot. The number of updates that have been applied to each column is held and, if q is the number of pivots chosen so far and the column to be searched has had $\mathtt{k} < \mathtt{q}$ updates, the column is updated with the last $\mathtt{q}-\mathtt{k}$ pivots to have been chosen before it was searched. If threshold rook pivoting (control%pivoting = 3) is being used, any row that is to be searched is also fully updated before it is searched. Each time a pivot is chosen, q is incremented by one. Once either p or nb pivots have been chosen, the remaining $\mathtt{n}-\mathtt{q}$ rows and columns of the matrix are updated using the BLAS kernels _trsm and _gemm. If $\mathtt{q} < \mathtt{p}$, the search of the columns restarts from column $\mathtt{q}+1$.

For threshold partial pivoting (control%pivoting = 1), an entry a_{km} of the reduced matrix is chosen as a pivot if it satisfies

$$|a_{km}| \geq \max(\texttt{control\%u} * \max_l |a_{lm}|, \texttt{control\%small}). \tag{4.1}$$

For threshold diagonal pivoting (control%pivoting = 2), pivots are initially chosen from the diagonal and must satisfy the threshold criteria (4.1) with $k = m$. If $p = n$ and static pivoting is not being used (that is, control%static = 0.0) and k < n pivots can be chosen from the diagonal that satisfy (4.1), the code switches to choosing off-diagonal pivots (so that the final n − k pivots may be off-diagonal entries). The number of pivots chosen from the diagonal is returned in info%num_diag. Note that diagonal pivoting is only recommended if the user knows the matrix has large entries on the diagonal. Diagonal pivoting makes no check on off-diagonal entries for IEEE infinities. For threshold rook pivoting (control%pivoting = 3), a_{km} is chosen as a pivot if it satisfies (4.1) and, additionally,

$$|a_{km}| \geq \texttt{control\%u} * \max_l |a_{kl}|. \tag{4.2}$$

Static pivoting may be used if control%static is positive. In this case, if fewer than p pivots can be chosen that satisfy the threshold criteria, the pivot candidate that came closest to satisfying these criteria is chosen. Thus for control%pivoting = 1, the pivot candidate a_{km} for which

$$ratio = |a_{km}| / \max_l |a_{lm}| \tag{4.3}$$

is the largest is chosen; for control%pivoting = 2, the candidate for which

$$ratio = |a_{mm}| / \max_l |a_{lm}| \tag{4.4}$$

is the largest is chosen (if $p = n$ and a_{mm} is zero for all m, the code switches to choosing off-diagonal pivots); for control%pivoting = 3, the candidate for which

$$ratio = min(|a_{km}| / \max_l |a_{lm}|, |a_{km}| / \max_l |a_{kl}|) \tag{4.5}$$

is the largest is chosen. If the absolute value of the chosen pivot a_{km} is greater than control%static, info%num_nothresh is incremented by one and info%usmall is set to the minimum of info%usmall and *ratio* (info%usmall is initialised to control%u). Otherwise, the pivot is given the value that has the same sign as a_{km} but absolute value control%static, info%usmall is set to zero and info%num_perturbed is incremented by one. Static pivoting ensures q = p pivots are chosen but the factorization may be inaccurate.

5 EXAMPLE OF USE

To illustrate the use of HSL_MA74, the following code performs a factorization and solves $Ax = b$ by doing two partial factorizations and two partial solves.

```
program example

  use hsl_ma74_double
  implicit none
  integer, parameter :: wp = kind(1.0d0)
  type(ma74_control) :: control
  type(ma74_info) :: info
  integer :: flag, i, j, ldap, n, nb, p, q, q1
```

```
  real(wp), allocatable :: ap(:,:), b(:), blocal(:), x(:)
  integer, allocatable :: pperm(:), qperm(:)
  integer, allocatable :: pperm1(:), qperm1(:), irow(:), jcol(:)

! Read the matrix order and number of eliminations for partial factorization
  read(*,*) n,p
  nb = min(p,100)
  ldap = n

! Allocate arrays for partial factorization
  allocate(ap(n,n), pperm(n), qperm(n))

! Read the matrix by columns
  do j = 1,n
    read(*,*) ap(1:n,j)
  end do

! Perform partial factorization
  call MA74_factor(n,p,nb,ap,ldap,q,pperm,qperm,control,info)
  if (info%flag /= 0) call terminate("ma74_factor")

  if (n > q) then
! Complete the factorization by performing a second partial factorization.
! Set pperm(p+1:n) and qperm(p+1:n)
    do i = p+1,n
      pperm(i) = i
      qperm(i) = i
    end do

! Allocate arrays to complete factorization
    allocate(pperm1(n-q), qperm1(n-q), irow(n-q), jcol(n-q))

! Let irow/jcol hold the row/column indices of the variables that remain
! to be eliminated
    do i = 1,n-q
      irow(i) = pperm(q+i)
      jcol(i) = qperm(q+i)
    end do

    call MA74_factor(n-q,n-q,nb,ap(q+1,q+1),ldap,q1, &
         pperm1,qperm1,control,info)
    if (info%flag /= 0) call terminate("ma74_factor")

! Permute the row and column indices
    irow(1:n-q) = irow(pperm1(1:n-q))
    jcol(1:n-q) = jcol(qperm1(1:n-q))

  end if
```

```
! Allocate arrays for the solve
  allocate(b(n), x(n), blocal(n))

! Read the right-hand side
  read(*,*) b(1:n)

  if (q > 0) then
! permute right-hand side b
    blocal(1:n) = b(pperm(1:n))

! Perform partial forward substitution
    call MA74_solveL1(n,q,nb,ap,ldap,blocal,flag)
    if (flag /= 0) call terminate("ma74_solvel1")

! Permute back into b
    b(pperm(1:n)) = blocal(1:n)
  end if

  if (n > q) then
! Load required part of right-hand sides b into a local array
    blocal(1:n-q) = b(irow(1:n-q))

! Forward substitution
     call MA74_solveL1(n-q,n-q,nb,ap(q+1,q+1),ldap,blocal,flag)
     if (flag /= 0) call terminate("ma74_solvel1")

! Back substitution
     call MA74_solveDU1(n-q,n-q,nb,ap(q+1,q+1),ldap,blocal,flag)
     if (flag /= 0) call terminate("ma74_solvedu1")
! Permute blocal into solution vector x
     x(jcol(1:n-q)) = blocal(1:n-q)
   end if

   if (q > 0) then
! Load partial solution into a local array blocal
     blocal(1:q) = b(pperm(1:q))

! and load required part of computed solution into blocal
     blocal(q+1:n) = x(qperm(q+1:n))

! Forward substitution
     call MA74_solveDU1(n,q,nb,ap,ldap,blocal,flag)
     if (flag /= 0) call terminate("ma74_solvedu1")
! Copy from blocal into x
     x(qperm(1:q)) = blocal(1:q)
   end if


  write(*,'(a,8f10.3)') ' Computed solution ',x(1:n)
```

```
  deallocate(ap, b, x, blocal, pperm, qperm)
  if (n > q) deallocate(pperm1, qperm1, irow, jcol)

contains

  subroutine terminate(name)
    character(*) name
    write(*,*) "Stopping after failure in ",name," with flag = ", flag
    stop
  end subroutine terminate

end program example
```

Given the data

```
   4   2
   3   1   1   4
   2  -1  -5   1
   1   2   2   0
  -1   0   1  -1
   5   2  -1   4
```

this produces the output:

```
 Computed solution      1.000     1.000     1.000     1.000
```