NAG Toolbox

nag_sparse_real_gen_precon_bdilu (f11df)

1 Purpose

nag_sparse_real_gen_precon_bdilu (f11df) computes a block diagonal incomplete LU factorization of a real sparse nonsymmetric matrix, represented in coordinate storage format. The diagonal blocks may be composed of arbitrary rows and the corresponding columns, and may overlap. This factorization can be used to provide a block Jacobi or additive Schwarz preconditioner, for use in combination with nag_sparse_real_gen_basic_solver (f11be) or nag_sparse_real_gen_solve_bdilu (f11dg).

2 Syntax

```
[a, irow, icol, ipivp, ipivq, istr, idiag, nnzc, npivm, ifail] =
nag_sparse_real_gen_precon_bdilu(n, nnz, a, irow, icol, istb, indb, lfill, dtol,
milu, ipivp, ipivq, 'la', la, 'nb', nb, 'lindb', lindb, 'pstrat', pstrat)
```

```
[a, irow, icol, ipivp, ipivq, istr, idiag, nnzc, npivm, ifail] = f11df(n, nnz,
a, irow, icol, istb, indb, lfill, dtol, milu, ipivp, ipivq, 'la', la, 'nb', nb,
'lindb', lindb, 'pstrat', pstrat)
```

3 Description

nag_sparse_real_gen_precon_bdilu (f11df) computes an incomplete LU factorization (see Meijerink and Van der Vorst (1977) and Meijerink and Van der Vorst (1981)) of the (possibly overlapping) diagonal blocks A_b, for $b = 1, 2, \ldots,$ **nb**, of a real sparse nonsymmetric n by n matrix A. The factorization is intended primarily for use as a block Jacobi or additive Schwarz preconditioner (see Saad (1996)), with one of the iterative solvers nag_sparse_real_gen_basic_solver (f11be) and nag_sparse_real_gen_solve_bdilu (f11dg).

The **nb** diagonal blocks need not consist of consecutive rows and columns of A, but may be composed of arbitrarily indexed rows, and the corresponding columns, as defined in the arguments **indb** and **istb**. Any given row or column index may appear in more than one diagonal block, resulting in overlap. Each diagonal block A_b, for $b = 1, 2, \ldots,$ **nb**, is factorized as:

$$A_b = M_b + R_b$$

where

$$M_b = P_b L_b D_b U_b Q_b$$

and L_b is lower triangular with unit diagonal elements, D_b is diagonal, U_b is upper triangular with unit diagonals, P_b and Q_b are permutation matrices, and R_b is a remainder matrix.

The amount of fill-in occurring in the factorization of block b can vary from zero to complete fill, and can be controlled by specifying either the maximum level of fill **lfill**(b), or the drop tolerance **dtol**(b).

The parameter **pstrat**(b) defines the pivoting strategy to be used in block b. The options currently available are no pivoting, user-defined pivoting, partial pivoting by columns for stability, and complete pivoting by rows for sparsity and by columns for stability. The factorization may optionally be modified to preserve the row-sums of the original block matrix.

The sparse matrix A is represented in coordinate storage (CS) format (see Section 2.1.1 in the F11 Chapter Introduction). The array **a** stores all the nonzero elements of the matrix A, while arrays **irow** and **icol** store the corresponding row and column indices respectively. Multiple nonzero elements may not be specified for the same row and column index.

The preconditioning matrices M_b, for $b = 1, 2, \ldots,$ **nb**, are returned in terms of the CS representations of the matrices

$$C_b = L_b + {D^{-1}}_b + U_b - 2I.$$

4 References

Meijerink J and Van der Vorst H (1977) An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix *Math. Comput.* **31** 148–162

Meijerink J and Van der Vorst H (1981) Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems *J. Comput. Phys.* **44** 134–155

Saad Y (1996) *Iterative Methods for Sparse Linear Systems* PWS Publishing Company, Boston, MA

5 Parameters

5.1 Compulsory Input Parameters

1: **n** – INTEGER

n, the order of the matrix A.

Constraint: $\mathbf{n} \geq 1$.

2: **nz** – INTEGER

The number of nonzero elements in the matrix A.

Constraint: $1 \leq \mathbf{nnz} \leq \mathbf{n}^2$.

3: **a**(**la**) – REAL (KIND=nag_wp) array

The nonzero elements in the matrix A, ordered by increasing row index, and by increasing column index within each row. Multiple entries for the same row and column indices are not permitted. The function nag_sparse_real_gen_sort (f11za) may be used to order the elements in this way.

4: **irow**(**la**) – INTEGER array

5: **icol**(**la**) – INTEGER array

The row and column indices of the nonzero elements supplied in **a**.

Constraints:

irow and **icol** must satisfy these constraints (which may be imposed by a call to nag_sparse_real_gen_sort (f11za)):

$1 \leq \mathbf{irow}(i) \leq \mathbf{n}$ and $1 \leq \mathbf{icol}(i) \leq \mathbf{n}$, for $i = 1, 2, \ldots, \mathbf{nnz}$;
either $\mathbf{irow}(i-1) < \mathbf{irow}(i)$ or both $\mathbf{irow}(i-1) = \mathbf{irow}(i)$ and $\mathbf{icol}(i-1) < \mathbf{icol}(i)$, for $i = 2, 3, \ldots, \mathbf{nnz}$.

6: **istb**(**nb** + **1**) – INTEGER array

istb(b), for $b = 1, 2, \ldots, \mathbf{nb}$, holds the indices in arrays **indb**, **ipivp**, **ipivq** and **idiag** that, on successful exit from this function, define block b. **istb**$(\mathbf{nb}+1)$ holds the sum of the number of rows in all blocks plus **istb**(1).

Constraint: $\mathbf{istb}(1) \geq 1, \mathbf{istb}(b) < \mathbf{istb}(b+1)$, for $b = 1, 2, \ldots, \mathbf{nb}$.

7: **indb**(**lindb**) – INTEGER array

indb must hold the row indices appearing in each diagonal block, stored consecutively. Thus the elements **indb**(**istb**(b)) to **indb**(**istb**$(b+1)-1$) are the row indices in the bth block, for $b = 1, 2, \ldots, \mathbf{nb}$.

Constraint: $1 \leq \mathbf{indb}(m) \leq \mathbf{n}$, for $m = 1, 2, \ldots, \mathbf{istb}(\mathbf{nb}+1) - 1$.

8: **lfill**(**nb**) – INTEGER array

If **lfill**$(b) \geq 0$ its value is the maximum level of fill allowed in the decomposition of the block (see Section 9.2 in nag_sparse_real_gen_precon_ilu (f11da)). A negative value of **lfill**(b) indicates that **dtol**(b) will be used to control the fill in the block instead.

9: **dtol**(**nb**) – REAL (KIND=nag_wp) array

If **lfill**$(b) < 0$ then **dtol**(b) is used as a drop tolerance in the block to control the fill-in (see Section 9.2 in nag_sparse_real_gen_precon_ilu (f11da)); otherwise **dtol**(b) is not referenced.

Constraint: if **lfill**$(b) < 0$, **dtol**$(b) \geq 0.0$, for $b = 1, 2, \ldots,$ **nb**.

10: **milu**(**nb**) – CHARACTER(1) array

milu(b), for $b = 1, 2, \ldots,$ **nb**, indicates whether or not the factorization in the block should be modified to preserve row-sums (see Section 9.4 in nag_sparse_real_gen_precon_ilu (f11da)).

milu$(b) =$ 'M'
The factorization is modified.

milu$(b) =$ 'N'
The factorization is not modified.

Constraint: **milu**$(b) =$ 'M' or 'N', for $b = 1, 2, \ldots,$ **nb**.

11: **ipivp**(**lindb**) – INTEGER array
12: **ipivq**(**lindb**) – INTEGER array

If **pstrat**$(b) =$ 'U', then **ipivp**(**istb**$(b) + k - 1$) and **ipivq**(**istb**$(b) + k - 1$) must specify the row and column indices of the element used as a pivot at elimination stage k of the factorization of the block. Otherwise **ipivp** and **ipivq** need not be initialized.

Constraint: if **pstrat**$(b) =$ 'U', the elements **istb**(b) to **istb**$(b+1) - 1$ of **ipivp** and **ipivq** must both hold valid permutations of the integers on $[1, \textbf{istb}(b+1) - \textbf{istb}(b)]$.

5.2 Optional Input Parameters

1: **la** – INTEGER

Default: the dimension of the arrays **a**, **irow**, **icol**. (An error is raised if these dimensions are not equal.)

The dimension of the arrays **a**, **irow** and **icol**. these arrays must be of sufficient size to store both A (**nnz** elements) and C (**nnzc** elements).

Note: the minimum value for **la** is only appropriate if **lfill** and **dtol** are set such that minimal fill-in occurs. If this is not the case then we recommend that **la** is set much larger than the minimum value indicated in the constraint.

Constraint: **la** $\geq 2 \times$ **nnz**.

2: **nb** – INTEGER

Default: the dimension of the arrays **lfill**, **dtol**, **pstrat**, **milu**. (An error is raised if these dimensions are not equal.)

The number of diagonal blocks to factorize.

Constraint: $1 \leq$ **nb** $\leq$ **n**.

3: **lindb** – INTEGER

Default: the dimension of the arrays **indb**, **ipivp**, **ipivq**. (An error is raised if these dimensions are not equal.)

The dimension of the arrays **indb**, **ipivp**, **ipivq** and **idiag**.

Constraint: **lindb** $\geq$ **istb**(**nb** + 1) − 1.

4: **pstrat**(**nb**) – CHARACTER(1) array

Suggested value: **pstrat**(b) = 'C', for $b = 1, 2, \ldots,$ **nb**.

Default: 'C'

pstrat(b), for $b = 1, 2, \ldots,$ **nb**, specifies the pivoting strategy to be adopted in the block as follows:

pstrat(b) = 'N'
No pivoting is carried out.

pstrat(b) = 'U'
Pivoting is carried out according to the user-defined input values of **ipivp** and **ipivq**.

pstrat(b) = 'P'
Partial pivoting by columns for stability is carried out.

pstrat(b) = 'C'
Complete pivoting by rows for sparsity, and by columns for stability, is carried out.

Constraint: **pstrat**(b) = 'N', 'U', 'P' or 'C', for $b = 1, 2, \ldots,$ **nb**.

5.3 Output Parameters

1: **a**(**la**) – REAL (KIND=nag_wp) array

The first **nnz** entries of **a** contain the nonzero elements of A and the next **nnzc** entries contain the elements of the matrices C_b, for $b = 1, 2, \ldots,$ **nb** stored consecutively. Within each block the matrix elements are ordered by increasing row index, and by increasing column index within each row.

2: **irow**(**la**) – INTEGER array
3: **icol**(**la**) – INTEGER array

The row and column indices of the nonzero elements returned in **a**.

4: **ipivp**(**lindb**) – INTEGER array
5: **ipivq**(**lindb**) – INTEGER array

The row and column indices of the pivot elements, arranged consecutively for each block, as for **indb**. If **ipivp**(**istb**(b) + k − 1) = i and **ipivq**(**istb**(b) + k − 1) = j, then the element in row i and column j of A_b was used as the pivot at elimination stage k.

6: **istr**(**lindb** + **1**) – INTEGER array

istr(**istb**(b) + k − 1), gives the index in the arrays **a**, **irow** and **icol** of row k of the matrix C_b, for $b = 1, 2, \ldots,$ **nb** and $k = 1, 2, \ldots,$ **istb**(b + 1) − **istb**(b).

istr(**istb**(**nb** + 1)) contains **nnz** + **nnzc** + 1.

7: **idiag**(**lindb**) – INTEGER array

idiag(**istb**(b) + k − 1), gives the index in the arrays **a**, **irow** and **icol** of the diagonal element in row k of the matrix C_b, for $b = 1, 2, \ldots,$ **nb** and $k = 1, 2, \ldots,$ **istb**($b + 1$) − **istb**(b).

8: **nnzc** – INTEGER

The sum total number of nonzero elements in the matrices C_b, for $b = 1, 2, \ldots,$ **nb**.

9: **npivm**(**nb**) – INTEGER array

If **npivm**$(b) > 0$ it gives the number of pivots which were modified during the factorization to ensure that M_b exists.

If **npivm**$(b) = -1$ no pivot modifications were required, but a local restart occurred (see Section 9.3 in nag_sparse_real_gen_precon_ilu (f11da)). The quality of the preconditioner will generally depend on the returned values of **npivm**(b), for $b = 1, 2, \ldots,$ **nb**.

If **npivm**(b) is large, for some block, the preconditioner may not be satisfactory. In this case it may be advantageous to call nag_sparse_real_gen_precon_bdilu (f11df) again with an increased value of **lfill**(b), a reduced value of **dtol**(b), or **pstrat**$(b) =$ 'C'.

10: **ifail** – INTEGER

ifail $= 0$ unless the function detects an error (see Section 5).

6 Error Indicators and Warnings

Errors or warnings detected by the function:

ifail $= 1$

Constraint: $1 \leq$ **nb** $\leq$ **n**.

Constraint: **dtol**$(b) \geq 0.0$, for $b = 1, 2, \ldots,$ **nb**.

Constraint: **istb**$(b + 1) >$ **istb**(b), for $b = 1, 2, \ldots,$ **nb**.

Constraint: **istb**$(1) \geq 1$.

Constraint: **la** $\geq 2 \times$ **nnz**.

Constraint: **lindb** $\geq$ **istb**(**nb** $+ 1) - 1$.

Constraint: $1 \leq$ **indb**$(m) \leq$ **n**, for $m = 1, 2, \ldots,$ **istb**(**nb** $+ 1) - 1$

Constraint: **milu**$(b) =$ 'M' or 'N' for all b.

Constraint: **nnz** $\leq$ **n**^{2}.

Constraint: **nnz** ≥ 1.

Constraint: **n** ≥ 1.

Constraint: **pstrat**$(b) =$ 'N', 'U', 'P' or 'C' for all b.

liwork is too small.

ifail $= 2$

Constraint: $1 \leq$ **irow**$(i) \leq$ **n**, for $i = 1, 2, \ldots,$ **nnz**.

Constraint: $1 \leq$ **icol**$(j) \leq$ **n**, for $j = 1, 2, \ldots,$ **nnz**.

On entry, element $\langle$*value*$\rangle$ of **a** was out of order.

On entry, location $\langle$*value*$\rangle$ of (**irow**, **icol**) was a duplicate.

ifail $= 3$

On entry, the user-supplied value of **ipivp** for block $\langle$*value*$\rangle$ lies outside its range.

On entry, the user-supplied value of **ipivp** for block $\langle$*value*$\rangle$ was repeated.

On entry, the user-supplied value of **ipivq** for block $\langle$*value*$\rangle$ lies outside its range.

On entry, the user-supplied value of **ipivq** for block $\langle$*value*$\rangle$ was repeated.

ifail $= 4$

The number of nonzero entries in the decomposition is too large.
The decomposition has been terminated before completion.
Either increase **la**, or reduce the fill by reducing **lfill**, or increasing **dtol**.

ifail $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

ifail $= -399$

Your licence key may have expired or may not have been installed correctly.

ifail $= -999$

Dynamic memory allocation failed.

7 Accuracy

The accuracy of the factorization of each block A_b will be determined by the size of the elements that are dropped and the size of any modifications made to the pivot elements. If these sizes are small then the computed factors will correspond to a matrix close to A_b. The factorization can generally be made more accurate by increasing the level of fill **lfill**(b), or by reducing the drop tolerance **dtol**(b) with **lfill**$(b) < 0$.

If nag_sparse_real_gen_precon_bdilu (f11df) is used in combination with nag_sparse_real_gen_basic_solver (f11be) or nag_sparse_real_gen_solve_bdilu (f11dg), the more accurate the factorization the fewer iterations will be required. However, the cost of the decomposition will also generally increase.

8 Further Comments

nag_sparse_real_gen_precon_bdilu (f11df) calls nag_sparse_real_gen_precon_ilu (f11da) internally for each block A_b. The comments and advice provided in Section 9 in nag_sparse_real_gen_precon_ilu (f11da) on timing, control of fill, algorithmic details, and choice of parameters, are all therefore relevant to nag_sparse_real_gen_precon_bdilu (f11df), if interpreted blockwise.

9 Example

This example program reads in a sparse matrix A and then defines a block partitioning of the row indices with a user-supplied overlap and computes an overlapping incomplete LU factorization suitable for use as an additive Schwarz preconditioner. Such a factorization is used for this purpose in the example program of nag_sparse_real_gen_solve_bdilu (f11dg).

9.1 Program Text

```
function f11df_example

fprintf('f11df example results\n\n');

% Sparse matrix A
n    = nag_int(9);
nz   = nag_int(33);
a    = zeros(20*nz, 1);
irow = zeros(20*nz, 1, nag_int_name);
icol = zeros(20*nz, 1, nag_int_name);
a(1:nz)    = [64; -20; -20; -12;  64; -20; -20; -12;  64; -20; -12;
               64; -20; -20; -12; -12;  64; -20; -20; -12; -12;  64;
              -20; -12;  64; -20; -12; -12;  64; -20; -12; -12;  64];
irow(1:nz) = [ 1;   1;   1;   2;   2;   2;   2;   3;   3;   3;   4;
                 4;   4;   4;   5;   5;   5;   5;   5;   6;   6;   6;
                 6;   7;   7;   7;   8;   8;   8;   8;   9;   9;   9];
icol(1:nz) = [ 1;   2;   4;   1;   2;   3;   5;   2;   3;   6;   1;
                 4;   5;   7;   2;   4;   5;   6;   8;   3;   5;   6;
```

```
              9;   4;   7;   8;   5;   7;   8;   9;   6;   8;   9];

% 3 Blocks
nb     = nag_int(3);
nover  = 1;
lfill  = [nag_int(0); 0; 0];
dtol   = [  0;   0;   0];
pstrat = {'n'; 'n'; 'n'};
milu   = {'n'; 'n'; 'n'};

% Define diagonal block indices.
% In this example use blocks of mb consecutive rows and initialise
% assuming no overlap.
mb    = idivide(n+nb-1, nb);
istb  = zeros(nb+1, 1, nag_int_name);
indb  = zeros(3*n, 1, nag_int_name);
ipivp = zeros(3*n, 1, nag_int_name);
ipivq = zeros(3*n, 1, nag_int_name);
istb(1:nb) = [1:mb:nb*mb];
istb(nb+1) = n+1;
indb(1:n)  = [1:n];

% Modify indb and istb to account for overlap.
[istb, indb, ifail] = f11df_overlap(n, nz, irow, icol, nb, ...
                                    istb, indb, 3*n, nover);
if (ifail == -999)
  error('indb is too small, size of indb = %d', numel(indb));
end

% Output matrix and blocking details
fprintf('\nOriginal matrix\n');
fprintf(' n  = %d\n', n);
fprintf(' nz = %d\n', nz);
fprintf(' nb = %d\n', nb);
for k=1:nb
  fprintf(' Block %d: order = %d, start row = %d\n', k, istb(k+1)-istb(k), ...
          min(indb(istb(k):istb(k+1)-1)));
end

% Calculate Factorisation
[a, irow, icol, ipivp, ipivq, istr, idiag, nnzc, npivm, ifail] = ...
f11df( ...
       n, nz, a, irow, icol, istb, indb, ...
       lfill, dtol, milu, ipivp, ipivq, 'pstrat', pstrat);

% Output details of the factorization
fprintf('\nFactorization\n');
fprintf(' nnzc = %d\n\n', nnzc);
fprintf(' Elements of factorization\n\n');
fprintf('        i   j        c(i,j)     index\n');
for k=1:nb
  fprintf(' C_%d   ---------------------------------\n', k);
  % Elements of the k-th block
  for i = istr(istb(k)):istr(istb(k+1))-1
    fprintf('     %4d%4d%16e%8d\n', irow(i), icol(i), a(i), i);
  end
end

fprintf('\n Details of factorized blocks\n\n');
if max(npivm) > 0
  % Including pivoting details.
  fprintf('  k   i      istr(i)  idiag(i)   indb(i)  ipivp(i)  ipivq(i)\n');
  for k=1:nb
    i = istb(k);
    fprintf(' %4d%4d%10d%10d%10d%10d%10d\n', k, i, istr(i), idiag(i), ...
            indb(i), ipivp(i), ipivq(i));
    for i = istb(k)+1:istb(k+1)-1
      fprintf(' %7d%10d%10d%10d%10d%10d\n', i, istr(i), idiag(i), ...
              indb(i), ipivp(i), ipivq(i));
    end
    fprintf(' -----------------------------------\n');
```

```
  end
else
  % No pivoting on any block.
  fprintf('  k   i      istr(i)  idiag(i)   indb(i)\n');
  for k=1:nb
    i = istb(k);
    fprintf('%3d%4d%10d%10d%10d\n', k, i, istr(i), idiag(i), indb(i));
    for i = istb(k)+1:istb(k+1)-1
      fprintf('%7d%10d%10d%10d\n', i, istr(i), idiag(i), indb(i));
    end
    fprintf(' ------------------------------------\n');
  end
end

function [istb, indb, ifail] = f11df_overlap(n, nz, irow, icol, nb, ...
                                             istb, indb, lindb, nover)

  ifail = 0;

  % This function takes a set of row indices indb defining the diagonal
  % blocks to be used in f11df to define a block Jacobi or additive Schwarz
  % preconditioner, and expands them to allow for nover levels of overlap.
  % The pointer array istb is also updated accordingly, so that the returned
  % values of istb and indb can be passed to f11df to define overlapping
  % diagonal blocks.
  iwork = zeros(3*n+1, 1, nag_int_name);

  % Find the number of non-zero elements in each row of the matrix A, and
  % the start address of each row. Store the start addresses in
  % iwork(n+1,...,2*n+1).
  for k=1:nz
    iwork(irow(k)) = iwork(irow(k)) + 1;
  end
  iwork(n+1) = 1;
  for i = 1:n
    iwork(n+i+1) = iwork(n+i) + iwork(i);
  end

  % Loop over blocks
  for k=1:nb
    % Initialize marker array.
    iwork(1:n) = 0;

    % Mark the rows already in block k in the workspace array.
    for l = istb(k):istb(k+1)-1
      iwork(indb(l)) = 1;
    end

    % Loop over levels of overlap.
    for iover=1:nover
      % Initialize counter of new row indices to be added.
      ind = 0;

      % Loop over the rows currently in the diagonal block.
      for l = istb(k):istb(k+1)-1
        row = indb(l);

        % Loop over non-zero elements in row
        for i = iwork(n+row):iwork(n+row+1)-1

          % If the column index of the non-zero element is not in the
          % existing set for this block, store it to be added later, and
          % mark it in the marker array.
          if (iwork(icol(i))==0)
            iwork(icol(i)) = 1;
            ind = ind + 1;
            iwork(2*n+1+ind) = icol(i);
          end
        end
      end
```

```
      % Shift the indices in indb and add the new entries for block k.
      % Change istb accordingly.
      nadd = ind;
      if (istb(nb+1)+nadd-1>lindb) Then
        ifail = -999;
        return;
      end

      for i = istb(nb+1) - 1:-1:istb(k+1)
        indb(i+nadd) = indb(i);
      end
      n21 = 2*n + 1;
      ik = istb(k+1) - 1;
      indb(ik+1:ik+nadd) = iwork(n21+1:n21+nadd);
      istb(k+1:nb+1) = istb(k+1:nb+1) + nadd;
    end
  end
```

9.2 Program Results

```
     f11df example results

Original matrix
 n  = 9
 nz = 33
 nb = 3
 Block 1: order = 6, start row = 1
 Block 2: order = 9, start row = 1
 Block 3: order = 6, start row = 4

Factorization
 nnzc = 73

 Elements of factorization

        i   j        c(i,j)     index
 C_1   ------------------------------
        1   1    1.562500e-02      34
        1   2   -3.125000e-01      35
        1   4   -3.125000e-01      36
        2   1   -1.875000e-01      37
        2   2    1.659751e-02      38
        2   3   -3.319502e-01      39
        2   5   -3.319502e-01      40
        3   2   -1.991701e-01      41
        3   3    1.666206e-02      42
        3   6   -3.332412e-01      43
        4   1   -1.875000e-01      44
        4   4    1.659751e-02      45
        4   5   -3.319502e-01      46
        5   2   -1.991701e-01      47
        5   4   -1.991701e-01      48
        5   5    1.784656e-02      49
        5   6   -3.569313e-01      50
        6   3   -1.999447e-01      51
        6   5   -2.141588e-01      52
        6   6    1.794754e-02      53
 C_2   ------------------------------
        1   1    1.562500e-02      54
        1   2   -3.125000e-01      55
        1   4   -1.875000e-01      56
        1   5   -3.125000e-01      57
        2   1   -1.875000e-01      58
        2   2    1.659751e-02      59
        2   3   -3.319502e-01      60
        2   6   -1.991701e-01      61
        2   7   -3.319502e-01      62
        3   2   -1.991701e-01      63
        3   3    1.666206e-02      64
        3   8   -1.999447e-01      65
```

```
         3   9   -3.332412e-01      66
         4   1   -3.125000e-01      67
         4   4    1.659751e-02      68
         4   6   -3.319502e-01      69
         5   1   -1.875000e-01      70
         5   5    1.659751e-02      71
         5   7   -3.319502e-01      72
         6   2   -3.319502e-01      73
         6   4   -1.991701e-01      74
         6   6    1.784656e-02      75
         6   8   -3.569313e-01      76
         7   2   -1.991701e-01      77
         7   5   -1.991701e-01      78
         7   7    1.784656e-02      79
         7   9   -3.569313e-01      80
         8   3   -3.332412e-01      81
         8   6   -2.141588e-01      82
         8   8    1.794754e-02      83
         9   3   -1.999447e-01      84
         9   7   -2.141588e-01      85
         9   9    1.794754e-02      86
C_3    -------------------------------
         1   1    1.562500e-02      87
         1   2   -3.125000e-01      88
         1   4   -1.875000e-01      89
         2   1   -1.875000e-01      90
         2   2    1.659751e-02      91
         2   3   -3.319502e-01      92
         2   5   -1.991701e-01      93
         3   2   -1.991701e-01      94
         3   3    1.666206e-02      95
         3   6   -1.999447e-01      96
         4   1   -3.125000e-01      97
         4   4    1.659751e-02      98
         4   5   -3.319502e-01      99
         5   2   -3.319502e-01     100
         5   4   -1.991701e-01     101
         5   5    1.784656e-02     102
         5   6   -3.569313e-01     103
         6   3   -3.332412e-01     104
         6   5   -2.141588e-01     105
         6   6    1.794754e-02     106

 Details of factorized blocks

  k    i      istr(i)  idiag(i)   indb(i)
  1    1        34        34         1
       2        37        38         2
       3        41        42         3
       4        44        45         4
       5        47        49         5
       6        51        53         6
 ------------------------------------
  2    7        54        54         4
       8        58        59         5
       9        63        64         6
      10        67        68         1
      11        70        71         7
      12        73        75         2
      13        77        79         8
      14        81        83         3
      15        84        86         9
 ------------------------------------
  3   16        87        87         7
      17        90        91         8
```

```
     18         94         95          9
     19         97         98          4
     20        100        102          5
     21        104        106          6
 ------------------------------------
```