NAG Library Function Document

nag_sparse_nherm_precon_bdilu (f11dtc)

1 Purpose
nag_sparse_nherm_precon_bdilu (f11dtc) computes a block diagonal incomplete $LU$ factorization of a complex sparse non-Hermitian 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_nherm_basic_solver (f11bsc) or nag_sparse_nherm_precon_bdilu_solve (f11duc).

2 Specification
#include <nag.h>
#include <nagf11.h>

void nag_sparse_nherm_precon_bdilu (Integer n, Integer nnz, Complex a[],
   Integer la, Integer irow[], Integer icol[], Integer nb,
   const Integer istb[], const Integer indb[], Integer lindb,
   const Integer lfill[], const double dtol[],
   const Nag_SparseNsym_Piv pstrat[], const Nag_SparseNsym_Fact milu[],
   Integer ipivp[], Integer ipivq[], Integer istr[], Integer idia[],
   Integer *nnzc, Integer npivm[], NagError *fail)

3 Description
nag_sparse_nherm_precon_bdilu (f11dtc) 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 $A_b, b = 1, 2, \ldots, nb$, of a complex sparse non-Hermitian $n$ 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_nherm_basic_solver (f11bsc) and nag_sparse_nherm_precon_bdilu_solve (f11duc).

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 $A_b, b = 1, 2, \ldots, nb$, is factorized as:

$$A_b = M_b + R_b$$

where

$$M_b = P_bL_bD_bU_bQ_b$$

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 $lfill[b - 1]$, or the drop tolerance $dtol[b - 1]$.

The parameter $pstrat[b - 1]$ 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 $M_b$, $b = 1, 2, \ldots, nb$, are returned in terms of the CS representations of the matrices

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

4 References


5 Arguments

1: $n$ – Integer

*Input*

On entry: $n$, the order of the matrix $A$.

*Constraint:* $n \geq 1$.

2: $nnz$ – Integer

*Input*

On entry: the number of nonzero elements in the matrix $A$.

*Constraint:* $1 \leq nnz \leq n^2$.

3: $a[\text{la}]$ – Complex

*Input/Output*

On entry: 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_nherm_sort (f11znc) may be used to order the elements in this way.

On exit: the first $nnz$ entries of $a$ contain the nonzero elements of $A$ and the next $nnz$ 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.

4: $la$ – Integer

*Input*

On entry: the dimension of the arrays $a$, $irow$ and $icol$. These arrays must be of sufficient size to store both $A$ ($nnz$ elements) and $C$ ($nnz$ 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:* $la \geq 2 \times nnz$.

5: $irow[la]$ – Integer

*Input/Output*

On entry: 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_nherm_sort (f11znc)):

$$1 \leq irow[i-1] \leq n \text{ and } 1 \leq icol[i-1] \leq n, \text{ for } i = 1, 2, \ldots, nnz;$$

either $irow[i-1] < irow[i]$ or both $irow[i-1] = irow[i]$ and $icol[i-1] < icol[i]$, for $i = 1, 2, \ldots, nnz$. 

6: $icol[la]$ – Integer

*Input/Output*
On exit: the row and column indices of the nonzero elements returned in a.

7: nb – Integer

On entry: the number of diagonal blocks to factorize.

Constraint: \(1 \leq \text{nb} \leq n\).

8: istb[\text{nb} + 1] – const Integer

On entry: \(\text{istb}[b - 1]\), for \(b = 1, 2, \ldots, \text{nb}\), holds the indices in arrays indb, ipivp, ipivq and idiaq that, on successful exit from this function, define block \(b\). Let \(r_b\) denote the number of rows in block \(b\); then \(\text{istb}[b] = \text{istb}[b - 1] + r_b\), for \(b = 1, 2, \ldots, \text{nb}\). Thus, \(\text{istb}[\text{nb}]\) holds the sum of the number of rows in all blocks plus \(\text{istb}[0]\).

Constraint: \(\text{istb}[0] \geq 1, \text{istb}[b - 1] < \text{istb}[b]\), for \(b = 1, 2, \ldots, \text{nb}\).

9: indb[lindb] – const Integer

On entry: indb must hold the row indices appearing in each diagonal block, stored consecutively. Thus the elements \(\text{indb}[k_b - 1]\), for \(k_b = \text{istb}[b - 1], \text{istb}[b - 1] + 1, \ldots, \text{istb}[b] - 2, \text{istb}[b] - 1\), are the row indices in the \(b\)th block, for \(b = 1, 2, \ldots, \text{nb}\).

Constraint: \(1 \leq \text{indb}[m - 1] \leq n\), for \(m = \text{istb}[0], \text{istb}[0] + 1, \ldots, \text{istb}[\text{nb}] - 1\).

10: lindb – Integer

On entry: the dimension of the arrays indb, ipivp, ipivq and idiaq.

Constraint: \(\text{lindb} \geq \text{istb}[\text{nb}] - 1\).

11: lfill[\text{nb}] – const Integer

On entry: if \(\text{lfill}[b - 1] \geq 0\) its value is the maximum level of fill allowed in the decomposition of the block \(b\) (see Section 9.2 in nag_sparse_nherm_fac (f11dnc)). A negative value of \(\text{lfill}[b - 1]\) indicates that \(\text{dtol}[b - 1]\) will be used to control the fill in block \(b\) instead.

12: dtol[\text{nb}] – const double

On entry: if \(\text{lfill}[b - 1] < 0\) then \(\text{dtol}[b - 1]\) is used as a drop tolerance in block \(b\) to control the fill-in (see Section 9.2 in nag_sparse_nherm_fac (f11dnc)); otherwise \(\text{dtol}[b - 1]\) is not referenced.

Constraint: if \(\text{lfill}[b - 1] < 0\), \(\text{dtol}[b - 1] \geq 0.0\), for \(b = 1, 2, \ldots, \text{nb}\).

13: pstrat[\text{nb}] – const Nag_SparseNsym_Piv

On entry: \(\text{pstrat}[b - 1]\), for \(b = 1, 2, \ldots, \text{nb}\), specifies the pivoting strategy to be adopted in the block as follows:

\[
\begin{align*}
\text{pstrat}[b - 1] &= \text{Nag_SparseNsym_NoPiv} & & \text{No pivoting is carried out.} \\
\text{pstrat}[b - 1] &= \text{Nag_SparseNsym_UserPiv} & & \text{Pivoting is carried out according to the user-defined input values of ipivp and ipivq.} \\
\text{pstrat}[b - 1] &= \text{Nag_SparseNsym_PartialPiv} & & \text{Partial pivoting by columns for stability is carried out.} \\
\text{pstrat}[b - 1] &= \text{Nag_SparseNsym_CompletePiv} & & \text{Complete pivoting by rows for sparsity, and by columns for stability, is carried out.}
\end{align*}
\]

Suggested value: \(\text{pstrat}[b - 1] = \text{Nag_SparseNsym_CompletePiv}\), for \(b = 1, 2, \ldots, \text{nb}\).

Constraint: \(\text{pstrat}[b - 1] = \text{Nag_SparseNsym_NoPiv}, \text{Nag_SparseNsym_UserPiv}, \text{Nag_SparseNsym_PartialPiv}\) or \(\text{Nag_SparseNsym_CompletePiv}\), for \(b = 1, 2, \ldots, \text{nb}\).
14: \textbf{milu}[\textit{nb}] – const Nag_SparseNsym_Fact \hspace{1cm} \textit{Input}

On entry: \textbf{milu}[b - 1], for \(b = 1, 2, \ldots, \text{nb}\), indicates whether or not the factorization in block \(b\) should be modified to preserve row-sums (see Section 9.4 in nag_sparse_nherm_fac (f11dnc)).

\textbf{milu}[b - 1] = Nag_SparseNsym_ModFact
The factorization is modified.

\textbf{milu}[b - 1] = Nag_SparseNsym_UnModFact
The factorization is not modified.

\textit{Constraint: milu}[b - 1] = Nag_SparseNsym_ModFact or Nag_SparseNsym_UnModFact, for \(b = 1, 2, \ldots, \text{nb}\).

15: \textbf{ipivp}[\textit{lindb}] – Integer \hspace{1cm} \textit{Input/Output}
16: \textbf{ipivq}[\textit{lindb}] – Integer \hspace{1cm} \textit{Input/Output}

On entry: if \textbf{pstrat}[b - 1] = Nag_SparseNsym_UserPiv, then \textbf{ipivp}[\textit{istb}][b - 1] + k - 2\) and \textbf{ipivq}[\textit{istb}][b - 1] + k - 2\) 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 \textbf{ipivp} and \textbf{ipivq} need not be initialized.

\textit{Constraint: if pstrat}[b - 1] = Nag_SparseNsym_UserPiv, the elements \textit{istb}[b - 1] - 1 to \textit{istb}[b] - 2\) of \textbf{ipivp} and \textbf{ipivq} must both hold valid permutations of the integers on \([1, \textit{istb}[b] - \textit{istb}[b - 1]]\).

On exit: the row and column indices of the pivot elements, arranged consecutively for each block, as for \textit{indb}. If \textbf{ipivp}[\textit{istb}][b - 1] + k - 2\) = \(i\) and \textbf{ipivq}[\textit{istb}][b - 1] + k - 2\) = \(j\), then the element in row \(i\) and column \(j\) of \(A_b\) was used as the pivot at elimination stage \(k\).

17: \textbf{istr}[\textit{lindb} + 1] – Integer \hspace{1cm} \textit{Output}

On exit: \textbf{istr}[\textit{istb}][b - 1] + k - 2\), gives the index in the arrays \textit{a}, \textit{irow} and \textit{icol} of row \(k\) of the matrix \(C_b\), for \(b = 1, 2, \ldots, \text{nb}\) and \(k = 1, 2, \ldots, \textit{istb}[b] - \textit{istb}[b - 1]\).


18: \textbf{idiag}[\textit{lindb}] – Integer \hspace{1cm} \textit{Output}

On exit: \textbf{idiag}[\textit{istb}][b - 1] + k - 2\), gives the index in the arrays \textit{a}, \textit{irow} and \textit{icol} of the diagonal element in row \(k\) of the matrix \(C_b\), for \(b = 1, 2, \ldots, \text{nb}\) and \(k = 1, 2, \ldots, \textit{istb}[b] - \textit{istb}[b - 1]\).

19: \textbf{nnzc} – Integer \hspace{1cm} \textit{Output}

On exit: the sum total number of nonzero elements in the matrices \(C_b\), for \(b = 1, 2, \ldots, \text{nb}\).

20: \textbf{npivm}[\textit{nb}] – Integer \hspace{1cm} \textit{Output}

On exit: if \textbf{npivm}[b - 1] > 0 it gives the number of pivots which were modified during the factorization to ensure that \(M_b\) exists.

If \textbf{npivm}[b - 1] = -1 no pivot modifications were required, but a local restart occurred (see Section 9.3 in nag_sparse_nherm_fac (f11dnc)). The quality of the preconditioner will generally depend on the returned values of \textbf{npivm}[b - 1], for \(b = 1, 2, \ldots, \text{nb}\).

If \textbf{npivm}[b - 1] is large, for some \(b\), the preconditioner may not be satisfactory. In this case it may be advantageous to call nag_sparse_nherm_precon_bdilu (f11dtc) again with an increased value of \textbf{lfill}[b - 1], a reduced value of \textbf{dtol}[b - 1], or \textbf{pstrat}[b - 1] = Nag_SparseNsym_CompletePiv.

21: \textbf{fail} – NagError \hspace{1cm} \textit{Input/Output}

The NAG error argument (see Section 3.6 in the Essential Introduction).
6  Error Indicators and Warnings

NE_ALLOC_FAIL
    Dynamic memory allocation failed.
    See Section 3.2.1.2 in the Essential Introduction for further information.

NE_BAD_PARAM
    On entry, argument \textlangle value\trangle had an illegal value.

NE_INT
    On entry, \textit{istb}[0] = \textlangle value\trangle.
    Constraint: \textit{istb}[0] \geq 1.
    On entry, \textit{n} = \textlangle value\trangle.
    Constraint: \textit{n} \geq 1.
    On entry, \textit{nnz} = \textlangle value\trangle.
    Constraint: \textit{nnz} \geq 1.

NE_INT_2
    On entry, \textit{la} = \textlangle value\trangle and \textit{nnz} = \textlangle value\trangle.
    Constraint: \textit{la} \geq 2 \times \textit{nnz}.
    On entry, \textit{nb} = \textlangle value\trangle and \textit{n} = \textlangle value\trangle.
    Constraint: 1 \leq \textit{nb} \leq \textit{n}.
    On entry, \textit{nnz} = \textlangle value\trangle and \textit{n} = \textlangle value\trangle.
    Constraint: \textit{nnz} \leq \textit{n}^2.

NE_INT_3
    On entry, \textit{lindb} = \textlangle value\trangle, \textit{istb}[\textit{nb}] - 1 = \textlangle value\trangle and \textit{nb} = \textlangle value\trangle.
    Constraint: \textit{lindb} \geq \textit{istb}[\textit{nb}] - 1.

NE_INT_ARRAY
    On entry, \textit{indb}[\textlangle value\trangle] = \textlangle value\trangle and \textit{n} = \textlangle value\trangle.
    Constraint: 1 \leq \textit{indb}[m - 1] \leq \textit{n}, for \textit{m} = \textit{istb}[0], \textit{istb}[0] + 1, \ldots, \textit{istb}[\textit{nb}] - 1.

NE_INTERNAL_ERROR
    An internal error has occurred in this function. Check the function call and any array sizes. If the
call is correct then please contact NAG for assistance.
    An unexpected error has been triggered by this function. Please contact NAG.
    See Section 3.6.6 in the Essential Introduction for further information.

NE_INVALID_CS
    On entry, \textit{icol}[\textlangle value\trangle] = \textlangle value\trangle and \textit{n} = \textlangle value\trangle.
    Constraint: 1 \leq \textit{icol}[j - 1] \leq \textit{n}, for \textit{j} = 1, 2, \ldots, \textit{nnz}.
    On entry, \textit{irow}[\textlangle value\trangle] = \textlangle value\trangle and \textit{n} = \textlangle value\trangle.
    Constraint: 1 \leq \textit{irow}[i - 1] \leq \textit{n}, for \textit{i} = 1, 2, \ldots, \textit{nnz}.

NE_INVALID_ROWCOL_PIVOT
    On entry, the user-supplied value of \textit{ipivp} for block \textlangle value\trangle lies outside its range.
    On entry, the user-supplied value of \textit{ipivp} for block \textlangle value\trangle was repeated.
    On entry, the user-supplied value of \textit{ipivq} for block \textlangle value\trangle lies outside its range.
On entry, the user-supplied value of ipivq for block (value) was repeated.

**NE_NO_LICENCE**

Your licence key may have expired or may not have been installed correctly. See Section 3.6.5 in the Essential Introduction for further information.

**NE_NOT_STRICTLY_INCREASING**

On entry, element (value) of a was out of order.
On entry, for \( b = (value) \), \( \text{istb}[b] = (value) \) and \( \text{istb}[b-1] = (value) \).
Constraint: \( \text{istb}[b] > \text{istb}[b-1] \), for \( b = 1,2,\ldots,\text{nb} \).
On entry, location (value) of (irow,icol) was a duplicate.

**NE_REAL_ARRAY**

On entry, \( dtol[b/(value)] = (value) \).
Constraint: \( dtol[b-1] \geq 0.0 \), for \( b = 1,2,\ldots,\text{nb} \).

**NE_TOO_SMALL**

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.

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 \( lfill[b-1] \), or by reducing the drop tolerance \( dtol[b-1] \) with \( lfill[b-1] < 0 \).

If nag_sparse_nherm_precon_bdilu (f11dtc) is used in combination with nag_sparse_nherm_basic_solver (f11bsc) or nag_sparse_nherm_precon_bdilu_solve (f11duc), the more accurate the factorization the fewer iterations will be required. However, the cost of the decomposition will also generally increase.

8 Parallelism and Performance

Not applicable.

9 Further Comments

nag_sparse_nherm_precon_bdilu (f11dtc) calls nag_sparse_nherm_fac (f11dnc) internally for each block \( A_b \). The comments and advice provided in Section 9 in nag_sparse_nherm_fac (f11dnc) on timing, control of fill, algorithmic details, and choice of parameters, are all therefore relevant to nag_sparse_nherm_precon_bdilu (f11dtc), if interpreted blockwise.

10 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_nherm_precon_bdilu_solve (f11duc).
10.1 Program Text

/* nag_sparse_nherm_precon_bdilu (f11dtc) Example Program. */
/* Copyright 2014 Numerical Algorithms Group. */
/* Mark 24, 2013. */
*/
#include <nag.h>
#include <nagf11.h>
#include <nag_stdlib.h>

static void overlap (Integer *n, Integer *nnz, Integer *irow, Integer *icol,
                      Integer *nb, Integer *istb, Integer *indb, Integer *lindb,
                      Integer *nover, Integer *iwork);

int main (void) {
    /* Scalars */
    double dtolg;
    Integer i, j, k, la, lfillg, lindb, liwork, minval, mb, n, nb, nnz, nnzc, nover;
    Integer exit_status = 0, maxval_ret = 9999;
    Nag_SparseNsym_Piv pstrag;
    Nag_SparseNsym_Fact milug;

    /* Arrays */
    char nag_enum_arg[40];
    double *dtol = 0;
    Complex *a = 0;
    Integer *icol = 0, *idiag = 0, *indb = 0, *ipivp = 0, *ipivq = 0, *irow = 0;
    Integer *istb = 0, *istr = 0, *iwork = 0, *lfill = 0, *npivm = 0;
    Nag_SparseNsym_Piv *pstrat;
    Nag_SparseNsym_Fact *milu;

    /* Nag Types */
    NagError fail;

    /* Print example header */
    printf("nag_sparse_nherm_precon_bdilu (f11dtc) Example Program Results\n\n");

    /* Skip heading in data file */
    #ifdef _WIN32
    scanf_s("%*\n");
    #else
    scanf("%*\n");
    #endif

    /* Get the matrix order and number of non-zero entries. */
    #ifdef _WIN32
    scanf_s("%"NAG_IFMT"%*[\n]", &n);
    #else
    scanf("%"NAG_IFMT"%*[\n]", &n);
    #endif
    #ifdef _WIN32
    scanf_s("%"NAG_IFMT"%*[\n]", &nnz);
    #else
    scanf("%"NAG_IFMT"%*[\n]", &nnz);
    #endif
    la = 20 * nnz;
    lindb = 3 * n;
    liwork = 9 * n + 3;

    /* Allocate arrays */
    a = NAG_ALLOC( la, Complex);
    irow = NAG_ALLOC( la, Integer);
    icol = NAG_ALLOC( la, Integer);
    idiag = NAG_ALLOC( lindb, Integer);
    indb = NAG_ALLOC( lindb, Integer);
ipivp = NAG_ALLOC(lindb, Integer);
ipivq = NAG_ALLOC(lindb, Integer);
istr = NAG_ALLOC(lindb+1, Integer);
iwork = NAG_ALLOC(liwork, Integer);
if ( (!a) || (!irow) || (!icol) || (!idiag) || (!indb) || (!ipivp) ||
    (!ipivq) || (!istr) || (!iwork) ) {
    printf("Allocation failure!\n");
    exit_status = -1;
}

/* Initialise arrays */
for (i = 0; i < la; i++) {
    a[i].re = 0.0;
    a[i].im = 0.0;
    irow[i] = 0;
    icol[i] = 0;
}
for (i = 0; i < lindb; i++) {
    indb[i] = 0;
    ipivp[i] = 0;
    ipivq[i] = 0;
    istr[i] = 0;
    idia[i] = 0;
}
istr[lindb] = 0;
for (i = 0; i < liwork; i++) {
    iwork[i] = 0;
}
/* Read the matrix A */
for (i = 0; i < nnz; i++) {
#ifdef _WIN32
    scanf_s(" ( %lf , %lf ) "%NAG_IFMT" "%NAG_IFMT"",
        &a[i].re, &a[i].im, &irow[i], &icol[i] );
#else
    scanf(" ( %lf , %lf ) "%NAG_IFMT" "%NAG_IFMT"",
        &a[i].re, &a[i].im, &irow[i], &icol[i] );
#endif
#endif
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
/* Read algorithmic parameters */
#ifdef _WIN32
    scanf_s("%"NAG_IFMT" %lf %*[\n]", &fillg, &dtolg);
#else
    scanf("%"NAG_IFMT" %lf %*[\n]", &fillg, &dtolg);
#endif
/* nag_enum_name_to_value (x04nac): Converts NAG enum member name to value */
#ifdef _WIN32
    scanf_s("%"NAG_IFMT" %lf %*[\n]", nag_enum_arg, _countof(nag_enum_arg));
#else
    scanf("%"NAG_IFMT" %lf %*[\n]", nag_enum_arg);
#endif
pstrag = (Nag_SparseNsym_Piv) nag_enum_name_to_value( nag_enum_arg );

#ifdef _WIN32
    scanf_s("%"NAG_IFMT" %lf %*[\n]", nag_enum_arg, _countof(nag_enum_arg));
#else
    scanf("%"NAG_IFMT" %lf %*[\n]", nag_enum_arg);
#endif
milug = (Nag_SparseNsym_Fact) nag_enum_name_to_value( nag_enum_arg );
/* Read algorithmic parameters */
#ifdef _WIN32
    scanf_s("%"NAG_IFMT" %"NAG_IFMT" %*[\n]", &nb, &nover);
#else
    scanf("%"NAG_IFMT" %"NAG_IFMT" %*[\n]", &nb, &nover);
#endif

/* Allocate arrays */
dtol = NAG_ALLOC( nb, double );
istb = NAG_ALLOC( nb+1, Integer );
lfill = NAG_ALLOC( nb, Integer );
npivm = NAG_ALLOC( nb, Integer );
pstrat = (Nag_SparseNsym_Piv *) NAG_ALLOC( nb, Nag_SparseNsym_Piv);
milu = (Nag_SparseNsym_Fact *) NAG_ALLOC( nb, Nag_SparseNsym_Fact);

if ( (!dtol) || (!istb) || (!lfill) || (!npivm) || (!pstrat) || (!milu) ) {
    printf("Allocation failure!\n");
exit_status = -1;
}

/* Initialise arrays */
for( i = 0; i < nb; i++ ) {
    dtol[i] = 0.0;
    istb[i] = 0;
    lfill[i] = 0;
    npivm[i] = 0;
    milu[i] = 0;
    pstrat[i] = 0;
}

istb[nb] = 0;

/* Define diagonal block indices. */
/* In this example use blocks of MB consecutive rows and initialise */
/* assuming no overlap. */
mb = (n + nb - 1)/nb;
for ( k = 0; k < nb; k++ ) {
    istb[k] = k * mb + 1;
}

for ( i = 0; i < n; i++ ) {
    indb[i] = i + 1;
}

/* Modify INDB and ISTB to account for overlap. */
overlap(&n, &nnz, irow, icol, &nb, istb, indb, &lindb, &nover, iwork);

/* Output matrix and blocking details */
printf(" Original Matrix\n");
printf(" n = %4"NAG_IFMT"\n", n);
printf(" nnz = %4"NAG_IFMT"\n", nnz);
printf(" nb = %4"NAG_IFMT"\n", nb);
for ( k = 0; k < nb; k++ ) {
    printf(" Block = %4"NAG_IFMT",%12s = %4"NAG_IFMT",", k+1, "order",
        istb[k+1] - istb[k]);
    minval = indb[istb[k]-1];
    for ( j = istb[k]; j < istb[k+1]-1; j++ ) {
        minval=MIN( minval, indb[j] );
    }
    printf("%13s = %4"NAG_IFMT"\n", "start row", minval);
}

/* Set algorithmic parameters for each block from global values */
for ( k = 0; k < nb; k++ ) {
    lfill[k] = 1fillg;
dtol[k] = dtolg;
pstrat[k] = pstrag;
milu[k] = milug;
}

/* Initialise fail */
INIT_FAIL(fail);

/* Calculate factorization */

* nag_sparse_nherm_precon_bdilu (f11dtc). Calculates incomplete LU
* factorization of local or overlapping diagonal blocks, mostly used
* as incomplete LU preconditioner for complex sparse matrix.
*

nag_sparse_nherm_precon_bdilu(n, nnz, a, la, irow, icol, nb, istb, indb,
    lindb, lfill, dtol, pstrat, milu, ipivp,
    ipivq, istr, idiag, &nnzc, npivm, &fail);

if( fail.code != NE_NOERROR ) {
    printf("Error from nag_sparse_nherm_precon_bdilu (f11dtc).\n%s\n", fail.message);
    exit(-2);
}

/* Output details of the factorization */
printf(" Factorization\n");
printf(" nnzc = %4"NAG_FMT"\n\n", nnzc);
printf(" Elements of factorization\n");
printf(" i j c(i, j) Index\n");
for( k = 0; k < nb; k++) {
    printf(" C_%1"NAG_FMT" ---------------------------\n", k+1);
    /* Elements of the k-th block */
    for ( i = istr[istb[k]-1]-1; i < istr[istb[k+1]-1]-1; i++) {
        printf("%9"NAG_FMT" %4"NAG_FMT" %13.5e %13.5e %7"NAG_FMT"
", irow[i], icol[i], a[i].re, a[i].im, i+1);
    }
}
k = 0;
maxval_ret = npivm[k];
for( k = 1; k < nb; k++) {
    maxval_ret = MAX( maxval_ret, npivm[k] );
}
printf("\n Details of factorized blocks\n");
if ( maxval_ret > 0 ) {
    /* Including pivoting details. */
    printf(" k i istr(i) idiag(i) indb(i) ipivp(i) ipivq(i)\n\n", k+1);
    /* Elements of the k-th block */
    for ( i = istr[istb[k]-1]-1; i < istr[istb[k+1]-1]-1; i++) {
        printf("%9"NAG_FMT" %4"NAG_FMT" %13.5e %13.5e %7"NAG_FMT"
", irow[i], icol[i], a[i].re, a[i].im, i+1);
    }
}

else {
    /* No pivoting on any block. */
}
printf(" k i istr(i) idia(i) indb(i)\n");
for ( k = 0; k < nb; k++) {
    i = istb[k] - 1;
    printf("%3"NAG_IFMT" %3"NAG_IFMT" %10"NAG_IFMT" ", k+1, i+1, istr[i]);
    printf("%10"NAG_IFMT" %10"NAG_IFMT"
", idia[i], indb[i]);
    for ( i = istb[k]; i < istb[k+1] - 1; i++) {
        printf("%7"NAG_IFMT" %10"NAG_IFMT" %10"NAG_IFMT" %10"NAG_IFMT"\n", i+1, istr[i], idia[i], indb[i]);
    }
    printf(" --------------------------------------\n");
}
NAG_FREE(a);
NAG_FREE(irow);
NAG_FREE(icol);
NAG_FREE(idia);
NAG_FREE(indb);
NAG_FREE(ipivp);
NAG_FREE(ipivq);
NAG_FREE(istr);
NAG_FREE(dtol);
NAG_FREE(istb);
NAG_FREE(lfill);
NAG_FREE(npivm);
NAG_FREE(pstrat);
NAG_FREE(milu);
NAG_FREE(iwork);
return exit_status;
} /* ************************************************************************** */
static void overlap(Integer *n, Integer *nnz, Integer *irow, Integer *icol,
    Integer *nb, Integer *istb, Integer *indb, Integer *lindb,
    Integer *nover, Integer *iwork) {
    /* Purpose */
    /* ======== */
    /* This routine takes a set of row indices INDB defining the diagonal blocks */
    /* to be used in nag_sparse_nherm_precon_bdilu (f11dtc) 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 on to */
    /* nag_sparse_nherm_precon_bdilu (f11dtc) to define overlapping diagonal */
    /* blocks. */
    /* ----------------------------------------------------------------------- */
    /* Scalars */
    Integer i, ik, ind, iover, j, k, l, n21, nadd, row;
    /* Find the number of nonzero elements in each row of the matrix A, and start */
    /* address of each row. Store the start addresses in iwork(n,...,2*n-1). */
    /* */
    for ( i = 0; i < (*n); i++) {
        iwork[i] = 0;
    }
    for ( i = 0; i < (*nnz); i++) {
        iwork[irow[i]-1] = iwork[irow[i]-1] + 1;
    }
    iwork[ (*n )] = 1;
    for ( i = 0; i < (*n); i++) {
        iwork[(*n)+i+1] = iwork[(*n)+i] + iwork[i];
    }
}
} /* Loop over blocks */
for ( k = 0; k < (*nb); k++) {

  /* Initialize marker array */
  for (j = 0; j < (*n); j++) {
    iwork[j] = 0;
  }

  /* Mark the rows already in block K in the workspace array */
  for (l = istb[k]; l < istb[k+1]; l++) {
    iwork[indb[l-1]-1] = 1;
  }

  /* Loop over levels of overlap */
  for (iover = 1; iover <= (*nover); iover++) {

    /* Initialize counter of new row indices to be added */
    ind = 0;

    /* Loop over the rows currently in the diagonal block */
    for (l = istb[k]; l < istb[k+1]; l++) {
      row = indb[l-1];

      /* Loop over non-zero elements in row ROW */
      for (i = iwork[(*n)+row-1]; i < iwork[(*n)+row]; i++) {

        /* 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-1]-1]==0) {
          iwork[icol[i-1]-1] = 1;
          iwork[2*(n)+1+ind] = icol[i-1];
          ind = ind + 1;
        }
      }
    }

    /* Shift the indices in INDB and add the new entries for block K. */
    /* Change ISTB accordingly. */
    nadd = ind;
    if (istb[(*nb)]+nadd-1 > (*lindb)) {
      printf("**** lindb too small, lindb = "$NAG_IFMT" ****\n", *lindb);
      exit(-1);
    }
    for (i = istb[(*nb)] - 1; i >= istb[k+1]; i--) {
      indb[i+nadd-1] = indb[i-1];
    }

    n2l = 2 * (n) + 1;
    ik = istb[k+1] - 1;

    for (j = 0; j < nadd; j++) {
      indb[ik + j] = iwork[n2l + j];
    }

    for (j = k+1; j < (*nb)+1; j++) {
      istb[j] = istb[j] + nadd;
    }
  }
}
return;
10.2 Program Data

nag_sparse_nherm_precon_bdilu (f11dtc) Example Program Data

```
9
nnz
( 96.0, -64.0) 1 1
(-20.0, 22.0) 1 2
(-36.0, 14.0) 1 4
(-12.0, 10.0) 2 1
( 96.0, -64.0) 2 2
(-20.0, 22.0) 2 3
(-36.0, 14.0) 2 5
(-12.0, 10.0) 3 2
( 96.0, -64.0) 3 3
(-36.0, 14.0) 3 6
(-28.0, 18.0) 4 1
( 96.0, -64.0) 4 4
(-20.0, 22.0) 4 5
(-36.0, 14.0) 4 7
(-28.0, 18.0) 5 2
(-12.0, 10.0) 5 4
( 96.0, -64.0) 5 5
(-20.0, 22.0) 5 6
(-36.0, 14.0) 5 8
(-28.0, 18.0) 6 3
(-12.0, 10.0) 6 5
( 96.0, -64.0) 6 6
(-36.0, 14.0) 6 9
(-28.0, 18.0) 7 4
( 96.0, -64.0) 7 7
(-20.0, 22.0) 7 8
(-28.0, 18.0) 8 5
(-12.0, 10.0) 8 7
( 96.0, -64.0) 8 8
(-20.0, 22.0) 8 9
(-28.0, 18.0) 9 6
(-12.0, 10.0) 9 8
( 96.0, -64.0) 9 9
```

0 0.0 :lfillg, dtolg
Nag_SparseNsym_NoPiv :pstrag
Nag_SparseNsym_ModFact :milug
3 1 :nb, nover

10.3 Program Results

nag_sparse_nherm_precon_bdilu (f11dtc) Example Program Results

Original Matrix

n = 9
nnz = 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  7.21154e-03 4.80769e-03  34
1  2 -2.50000e-01 6.25000e-02  35
1  4 -3.26923e-01 -7.21154e-02  36
2  1 -1.34615e-01 1.44231e-02  37
2  2  7.51634e-03 4.87709e-03  38
2  3 -2.57623e-01 6.78176e-02  39
2  5 -3.38867e-01 -7.03465e-02  40
3  2 -1.38967e-01 1.66383e-02  41
3  3  7.52786e-03 4.87530e-03  42
```
```
<table>
<thead>
<tr>
<th>k</th>
<th>i</th>
<th>istr(i)</th>
<th>idiag(i)</th>
<th>indb(i)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>7.21154e-03</td>
<td>4.80769e-03</td>
<td>54</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>-3.39257e-01</td>
<td>-7.01208e-02</td>
<td>43</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>-2.88462e-01</td>
<td>-4.80769e-03</td>
<td>44</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>7.82358e-03</td>
<td>5.49946e-03</td>
<td>45</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>-2.98245e-01</td>
<td>-1.26443e-03</td>
<td>47</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>-2.89400e-01</td>
<td>6.89668e-02</td>
<td>50</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>-2.98536e-01</td>
<td>-1.00694e-02</td>
<td>51</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>-1.55586e-01</td>
<td>1.49357e-02</td>
<td>52</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>8.28693e-03</td>
<td>5.64169e-03</td>
<td>53</td>
</tr>
</tbody>
</table>

C_2
```

```
<table>
<thead>
<tr>
<th>k</th>
<th>i</th>
<th>istr(i)</th>
<th>idiag(i)</th>
<th>indb(i)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>7.21154e-03</td>
<td>4.80769e-03</td>
<td>54</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-2.50000e-01</td>
<td>6.25000e-02</td>
<td>55</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>-2.88462e-01</td>
<td>-4.80769e-03</td>
<td>56</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>-3.26923e-01</td>
<td>-7.21154e-02</td>
<td>57</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>-1.34615e-01</td>
<td>1.44231e-02</td>
<td>58</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>7.51634e-03</td>
<td>4.87709e-03</td>
<td>59</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>-2.57623e-01</td>
<td>6.78176e-02</td>
<td>60</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>-2.98245e-01</td>
<td>-1.26443e-03</td>
<td>61</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>-3.38867e-01</td>
<td>-7.03465e-02</td>
<td>62</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>-1.38967e-01</td>
<td>1.66383e-02</td>
<td>63</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>7.52786e-03</td>
<td>4.87530e-03</td>
<td>64</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>-2.98536e-01</td>
<td>-1.00694e-02</td>
<td>65</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>-3.39257e-01</td>
<td>-7.01208e-02</td>
<td>66</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>-3.26923e-01</td>
<td>-7.21154e-02</td>
<td>67</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>7.82358e-03</td>
<td>5.49946e-03</td>
<td>68</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>-2.89400e-01</td>
<td>-1.26443e-03</td>
<td>69</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>-2.89400e-01</td>
<td>-1.26443e-03</td>
<td>70</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>-1.55586e-01</td>
<td>1.49357e-02</td>
<td>71</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>-3.82693e-03</td>
<td>5.64169e-03</td>
<td>72</td>
</tr>
</tbody>
</table>

C_3
```

```
<table>
<thead>
<tr>
<th>k</th>
<th>i</th>
<th>istr(i)</th>
<th>idiag(i)</th>
<th>indb(i)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>7.21154e-03</td>
<td>4.80769e-03</td>
<td>87</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>-2.50000e-01</td>
<td>6.25000e-02</td>
<td>88</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>-2.88462e-01</td>
<td>-4.80769e-03</td>
<td>89</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>-1.34615e-01</td>
<td>1.44231e-02</td>
<td>90</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>7.51634e-03</td>
<td>4.87709e-03</td>
<td>91</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>-2.57623e-01</td>
<td>6.78176e-02</td>
<td>92</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>-2.98245e-01</td>
<td>-1.26443e-03</td>
<td>93</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>-1.38967e-01</td>
<td>1.66383e-02</td>
<td>94</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>7.52786e-03</td>
<td>4.87530e-03</td>
<td>95</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>-2.98536e-01</td>
<td>-1.00694e-02</td>
<td>96</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>-3.26923e-01</td>
<td>-7.21154e-02</td>
<td>97</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>7.82358e-03</td>
<td>5.49946e-03</td>
<td>98</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>-2.98245e-01</td>
<td>-1.26443e-03</td>
<td>99</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>-3.38867e-01</td>
<td>-7.03465e-02</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>-1.48878e-01</td>
<td>1.22423e-02</td>
<td>101</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>8.26388e-03</td>
<td>5.64193e-03</td>
<td>102</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>-2.89400e-01</td>
<td>6.89668e-02</td>
<td>103</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>-3.39257e-01</td>
<td>-7.01208e-02</td>
<td>104</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>8.28693e-03</td>
<td>5.64169e-03</td>
<td>105</td>
</tr>
</tbody>
</table>

Details of factorized blocks
```

```
<table>
<thead>
<tr>
<th>k</th>
<th>i</th>
<th>istr(i)</th>
<th>idiag(i)</th>
<th>indb(i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>34</td>
<td>34</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>37</td>
<td>38</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

-1
```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>41</td>
<td>42</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>44</td>
<td>45</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>47</td>
<td>49</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>51</td>
<td>53</td>
<td>6</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>54</td>
<td>54</td>
</tr>
<tr>
<td>8</td>
<td>58</td>
<td>59</td>
<td>5</td>
</tr>
<tr>
<td>9</td>
<td>63</td>
<td>64</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>67</td>
<td>68</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>70</td>
<td>71</td>
<td>7</td>
</tr>
<tr>
<td>12</td>
<td>73</td>
<td>75</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>77</td>
<td>79</td>
<td>8</td>
</tr>
<tr>
<td>14</td>
<td>81</td>
<td>83</td>
<td>3</td>
</tr>
<tr>
<td>15</td>
<td>84</td>
<td>86</td>
<td>9</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>87</td>
<td>87</td>
</tr>
<tr>
<td>17</td>
<td>90</td>
<td>91</td>
<td>8</td>
</tr>
<tr>
<td>18</td>
<td>94</td>
<td>95</td>
<td>9</td>
</tr>
<tr>
<td>19</td>
<td>97</td>
<td>98</td>
<td>4</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
<td>102</td>
<td>5</td>
</tr>
<tr>
<td>21</td>
<td>104</td>
<td>106</td>
<td>6</td>
</tr>
</tbody>
</table>