

nag_sparse_nsym_fac (f11dac)

1. Purpose

nag_sparse_nsym_fac (f11dac) computes an incomplete LU factorization of a real sparse nonsymmetric matrix, represented in coordinate storage format. This factorization may be used as a preconditioner in combination with **nag_sparse_nsym_fac_sol (f11dcc)**.

2. Specification

```
#include <nag.h>
#include <nagf11.h>

void nag_sparse_nsym_fac(Integer n, Integer nnz, double **a, Integer *la,
                        Integer **irow, Integer **icol, Integer lfill, double dtol,
                        Nag_SparseNsym_Piv pstrat, Nag_SparseNsym_Fact milu,
                        Integer ipivp[], Integer ipivq[], Integer istr[], Integer idiag[],
                        Integer *nnzc, Integer *npivm, NagError *fail)
```

3. Description

This routine computes an incomplete LU factorization (Meijerink and van der Vorst (1977) and Meijerink and van der Vorst (1981)) of a real sparse nonsymmetric n by n matrix A . The factorization is intended primarily for use as a preconditioner with the iterative solver **nag_sparse_nsym_fac_sol (f11dcc)**.

The decomposition is written in the form

$$A = M + R$$

where

$$M = PLDUQ$$

and L is lower triangular with unit diagonal elements, D is diagonal, U is upper triangular with unit diagonals, P and Q are permutation matrices, and R is a remainder matrix.

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

The argument **pstrat** defines the pivoting strategy to be used. 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 matrix.

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

The preconditioning matrix M is returned in terms of the CS representation of the matrix

$$C = L + D^{-1} + U - 2I.$$

Further algorithmic details are given in Section 6.3.

4. Parameters

n

Input: the order of the matrix A .

Constraint: $n \geq 1$.

nnz

Input: the number of non-zero elements in the matrix A .

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

a[la]

Input: the non-zero 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 routine nag_sparse_nsym_sort (f11zac) may be used to order the elements in this way.

Output: the first **nnz** entries of **a** contain the non-zero elements of A and the next **nnzc** entries contain the elements of the matrix C . Matrix elements are ordered by increasing row index, and by increasing column index within each row

la

Input: the dimension of the arrays **a**, **irow** and **icol** as declared in the calling program. These arrays must be of sufficient size to store both A (**nnz** elements) and C (**nnzc** elements); for this reason the length of the arrays may be changed internally by calls to **realloc**. It is therefore *imperative* that these arrays are *allocated* using the NAG macro **NAG_ALLOC** and **NOT** declared as automatic arrays.

Output: if internal allocation has taken place then **la** is set to **nnz** + **nnzc**, otherwise it remains unchanged.

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

irow[la]**icol[la]**

Input: the row and column indices of the non-zero elements supplied in **a**.

Constraints: **irow** and **icol** must satisfy the following constraints (which may be imposed by a call to nag_sparse_nsym_sort (f11zac)):

$$1 \leq \mathbf{irow}[i] \leq \mathbf{n} \text{ and } 1 \leq \mathbf{icol}[i] \leq \mathbf{n}, \text{ for } i = 0, 1, \dots, \mathbf{nnz}-1.$$

$$\mathbf{irow}[i-1] < \mathbf{irow}[i], \text{ or } \mathbf{irow}[i-1] = \mathbf{irow}[i] \text{ and } \mathbf{icol}[i-1] < \mathbf{icol}[i], \text{ for } i = 1, 2, \dots, \mathbf{nnz}-1.$$

Output: the row and column indices of the non-zero elements returned in **a**.

lfill

Input: if **lfill** ≥ 0 its value is the maximum level of fill allowed in the decomposition (see Section 6.2). A negative value of **lfill** indicates that **dtol** will be used to control the fill instead.

dtol

Input: if **lfill** < 0 then **dtol** is used as a drop tolerance to control the fill-in (see Section 6.2); otherwise **dtol** is not referenced.

Constraint: **dtol** ≥ 0.0 if **lfill** < 0 .

pstrat

Input: specifies the pivoting strategy to be adopted as follows:

if **pstrat** = **Nag_SparseNsym_NoPiv** then no pivoting is carried out;

if **pstrat** = **Nag_SparseNsym_UserPiv** then pivoting is carried out according to the user-defined input value of **ipivp** and **ipivq**;

if **pstrat** = **Nag_SparseNsym_PartialPiv** then partial pivoting by columns for stability is carried out;

if **pstrat** = **Nag_SparseNsym_CompletePiv** then complete pivoting by rows for sparsity, and by columns for stability, is carried out.

Suggested value: **pstrat** = **Nag_SparseNsym_CompletePiv**.

Constraint: **pstrat** = **Nag_SparseNsym_NoPiv**, **Nag_SparseNsym_UserPiv**, **Nag_SparseNsym_PartialPiv** or **Nag_SparseNsym_CompletePiv**.

milu

Input: indicates whether or not the factorization should be modified to preserve row sums (see Section 6.4):

if **milu** = **Nag_SparseNsym_ModFact** the factorization is modified (MILU);

if **milu** = **Nag_SparseNsym_UnModFact** then the factorization is not modified.

Constraint: **milu** = **Nag_SparseNsym_ModFact** or **Nag_SparseNsym_UnModFact**.

ipivp[n]
ipivq[n]

Input: if **pstrat** = **Nag_SparseNsym_UserPiv**, then **ipivp**[$k-1$] and **ipivq**[$k-1$] must specify the row and column indices of the element used as a pivot at elimination stage k . Otherwise **ipivp** and **ipivq** need not be initialized.

Constraint: if **pstrat** = **Nag_SparseNsym_UserPiv**, then **ipivp** and **ipivq** must both hold valid permutations of the integers on [1,**n**].

Output: the pivot indices. If **ipivp**[$k-1$] = i and **ipivq**[$k-1$] = j then the element in row i and column j was used as the pivot at elimination stage k .

istr[n+1]

Output: **istr**[$i-1$], for $i = 1, 2, \dots, \mathbf{n}$ holds the index of arrays **a**, **irow** and **icol** where row i of the matrix C starts. **istr**[**n**] holds the address of the last non-zero element in C plus one.

idiag[n]

Output: **idiag**[$i-1$], for $i = 1, 2, \dots, \mathbf{n}$ holds the index in the arrays **a**, **irow** and **icol** which holds the diagonal element in row i of the matrix C .

nnzc

Output: the number of non-zero elements in the matrix C .

npivm

Output: if **npivm** > 0 it gives the number of pivots which were modified during the factorization to ensure that M exists. If **npivm** = -1 no pivot modifications were required, but a local restart occurred (Section 6.3). The quality of the preconditioner will generally depend on the returned value of **npivm**. If **npivm** is large the preconditioner may not be satisfactory. In this case it may be advantageous to call **nag_sparse_nsym_fac** again with an increased value of **lfill**, a reduced value of **dtol**, or **pstrat** = **Nag_SparseNsym_CompletePiv**.

fail

The NAG error parameter, see the Essential Introduction to the NAG C Library.

5. Error Indications and Warnings

NE_BAD_PARAM

On entry, parameter **milu** had an illegal value.
 On entry, parameter **pstrat** had an illegal value.

NE_REAL_INT_ARG_CONS

On entry, **dtol** = $\langle value \rangle$ and **lfill** = $\langle value \rangle$.
 These parameters must satisfy **dtol** \geq 0.0 if **lfill** < 0.

NE_2_INT_ARG_LT

On entry, **la** = $\langle value \rangle$ while **nnz** = $\langle value \rangle$.
 These parameters must satisfy **la** \geq $2 \times \mathbf{nnz}$.

NE_INT_ARG_LT

On entry, **n** must not be less than 1: **n** = $\langle value \rangle$.

NE_INT_2

On entry, **nnz** = $\langle value \rangle$, **n** = $\langle value \rangle$.
 Constraint: $1 \leq \mathbf{nnz} \leq \mathbf{n}^2$.

NE_INVALID_ROWCOL_PIVOT

On entry, **pstrat** = **Nag_SparseNsym_UserPiv**, but one or both of the arrays **ipivp** and **ipivq** does not represent a valid permutation of the integers in [1,**n**]. An input value of **ipivp** or **ipivq** is either out of range or repeated.

NE_NONSYMM_MATRIX_DUP

A non-zero matrix element has been supplied which does not lie within the matrix A , is out of order or has duplicate row and column indices, i.e., one or more of the following constraints has been violated:

$1 \leq \mathbf{irow}[i] \leq \mathbf{n}$, $1 \leq \mathbf{icol}[i] \leq \mathbf{n}$, for $i = 0, 1, \dots, \mathbf{nnz}-1$.

$\mathbf{irow}[i-1] < \mathbf{irow}[i]$, or
 $\mathbf{irow}[i-1] = \mathbf{irow}[i]$ and $\mathbf{icol}[i-1] < \mathbf{icol}[i]$, for $i = 1, 2, \dots, \mathbf{nnz}-1$.

Call nag_sparse_nsym_sort (f11zac) to reorder and sum or remove duplicates.

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 consult NAG for assistance.

NE_ALLOC_FAIL

Memory allocation failed.

6. Further Comments

The time taken for a call to nag_sparse_nsym_fac is roughly proportional to $\mathbf{nnzc}^2/\mathbf{n}$.

6.1. Accuracy

The accuracy of the factorization 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 . The factorization can generally be made more accurate by increasing **lfill**, or by reducing **dtol** with **lfill** < 0 .

If nag_sparse_nsym_fac is used in combination with nag_sparse_nsym_fac_sol (f11dcc), the more accurate the factorization the fewer iterations will be required. However, the cost of the decomposition will also generally increase.

6.2 Control of Fill-in

If **lfill** ≥ 0 the amount of fill-in occurring in the incomplete factorization is controlled by limiting the maximum **level** of fill-in to **lfill**. The original non-zero elements of A are defined to be of level 0. The fill level of a new non-zero location occurring during the factorization is defined as:

$$k = \max(k_e, k_c) + 1,$$

where k_e is the level of fill of the element being eliminated, and k_c is the level of fill of the element causing the fill-in.

If **lfill** < 0 the fill-in is controlled by means of the **drop tolerance dtol**. A potential fill-in element a_{ij} occurring in row i and column j will not be included if:

$$|a_{ij}| < \mathbf{dtol} \times \alpha,$$

where α is the maximum absolute value element in the matrix A .

For either method of control, any elements which are not included are discarded unless **milu** = **Nag.SparseNsym.ModFact**, in which case their contributions are subtracted from the pivot element in the relevant elimination row, to preserve the row-sums of the original matrix.

Should the factorization process break down a local restart process is implemented as described in Section 6.3. This will affect the amount of fill present in the final factorization.

6.3 Algorithmic Details

The factorization is constructed row by row. At each elimination stage a row index is chosen. In the case of complete pivoting this index is chosen in order to reduce fill-in. Otherwise the rows are treated in the order given, or some user-defined order.

The chosen row is copied from the original matrix A and modified according to those previous elimination stages which affect it. During this process any fill-in elements are either dropped or kept according to the values of **lfill** or **dtol**. In the case of a modified factorization (**milu** = **Nag.SparseNsym.ModFact**) the sum of the dropped terms for the given row is stored.

Finally the pivot element for the row is chosen and the multipliers are computed for this elimination stage. For partial or complete pivoting the pivot element is chosen in the interests of stability as the element of largest absolute value in the row. Otherwise the pivot element is chosen in the order given, or some user-defined order.

If the factorization breaks down because the chosen pivot element is zero, or there is no non-zero pivot available, a local restart recovery process is implemented. The modification of the given pivot row according to previous elimination stages is repeated, but this time keeping all fill. Note that in this case the final factorization will include more fill than originally specified by the user-supplied value of **lfill** or **dtol**. The local restart usually results in a suitable non-zero pivot arising. The original criteria for dropping fill-in elements is then resumed for the next elimination stage (hence the **local** nature of the restart process). Should this restart process also fail to produce a non-zero pivot element an arbitrary unit pivot is introduced in an arbitrarily chosen column. `nag_sparse_nsym_fac` (f11dac) returns an integer parameter **npivm** which gives the number of these arbitrary unit pivots introduced. If no pivots were modified but local restarts occurred **npivm** = -1 is returned.

6.4 Choice of Parameters

There is unfortunately no choice of the various algorithmic parameters which is optimal for all types of matrix, and some experimentation will generally be required for each new type of matrix encountered.

If the matrix A is not known to have any particular special properties the following strategy is recommended. Start with **lfill** = 0 and **pstrat** = **Nag_SparseNsym_CompletePiv**. If the value returned for **npivm** is significantly larger than zero, i.e., a large number of pivot modifications were required to ensure that M existed, the preconditioner is not likely to be satisfactory. In this case increase **lfill** until **npivm** falls to a value close to zero.

If A has non-positive off-diagonal elements, is non-singular, and has only non-negative elements in its inverse, it is called an ‘M-matrix’. It can be shown that no pivot modifications are required in the incomplete LU factorization of an M-matrix (Meijerink and van der Vorst (1977)). In this case a good preconditioner can generally be expected by setting **lfill** = 0, **pstrat** = **Nag_SparseNsym_NoPiv** and **milu** = **Nag_SparseNsym_ModFact**.

Some illustrations of the application of `nag_sparse_nsym_fac` (f11dac) to linear systems arising from the discretization of two-dimensional elliptic partial differential equations, and to random-valued randomly structured linear systems, can be found in Salvini and Shaw (1996).

6.5 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.
- Salvini S A and Shaw G J (1996) An evaluation of new NAG Library solvers for large sparse unsymmetric linear systems *NAG Technical Report TR2/96*, NAG Ltd, Oxford.

7. See Also

`nag_sparse_nsym_fac_sol` (f11dcc)
`nag_sparse_nsym_sort` (f11zac)

8. Example

This example program reads in a sparse matrix A and calls `nag_sparse_nsym_fac` to compute an incomplete LU factorization. It then outputs the non-zero elements of both A and $C = L + D^{-1} + U - 2I$.

The call to `nag_sparse_nsym_fac` has **lfill** = 0, and **pstrat** = **Nag_SparseNsym_CompletePiv**, giving an unmodified zero-fill LU factorization, with row pivoting for sparsity and column pivoting for stability.

8.1. Program Text

```
/* nag_sparse_nsym_fac(f11dac) Example Program.
 *
 * Copyright 1998 Numerical Algorithms Group.
```

```

*
* Mark 5, 1998.
*/

#include <nag.h>
#include <nag_stdlib.h>
#include <nag_string.h>
#include <stdio.h>
#include <nagf11.h>

main()
{
    double dtol;
    double *a=0;

    Integer *icol=0, *irow=0, *istr=0, *idiag=0, *ipivp=0, *ipivq=0;
    Integer nnzc ;
    Integer i, n, lfill, npivm;
    Integer nnz;
    Integer num;

    Nag_SparseNsym_Fact milu;
    Nag_SparseNsym_Piv pstrat;

    char char_enum[20];

    Vprintf("f11dac Example Program Results\n");
    /* Skip heading in data file */
    Vscanf("%*[^\\n]");

    Vscanf("%ld%*[^\\n]", &n);
    Vscanf("%ld%*[^\\n]", &nnz);
    Vscanf("%ld%lf%*[^\\n]", &lfill, &dtol);

    Vscanf("%s%*[^\\n]", char_enum);
    if (!strcmp(char_enum, "NoPiv"))
        pstrat = Nag_SparseNsym_NoPiv;
    else if (!strcmp(char_enum, "UserPiv"))
        pstrat = Nag_SparseNsym_UserPiv;
    else if (!strcmp(char_enum, "PartialPiv"))
        pstrat = Nag_SparseNsym_PartialPiv;
    else if (!strcmp(char_enum, "CompletePiv"))
        pstrat = Nag_SparseNsym_CompletePiv;
    else
    {
        Vprintf("Unrecognised string for pstrat enum representation.\n");
        exit (EXIT_FAILURE);
    }

    Vscanf("%s%*[^\\n]", char_enum);
    if (!strcmp(char_enum, "ModFact"))
        milu = Nag_SparseNsym_ModFact;
    else if (!strcmp(char_enum, "UnModFact"))
        milu = Nag_SparseNsym_UnModFact;
    else
    {
        Vprintf("Unrecognised string for method enum representation.\n");
        exit (EXIT_FAILURE);
    }

    num = 2*nnz;
    istr = NAG_ALLOC(n+1, Integer);
    idiag = NAG_ALLOC(n, Integer);
    ipivp = NAG_ALLOC(n, Integer);
    ipivq = NAG_ALLOC(n, Integer);
    irow = NAG_ALLOC(num, Integer);
    icol = NAG_ALLOC(num, Integer);
    a = NAG_ALLOC(num, double);

    if (!istr || !idiag || !ipivp || !ipivq || !irow || !icol || !a)

```

```

    {
        Vprintf("Allocation failure\n");
        exit (EXIT_FAILURE);
    }

    /* Read the matrix a */

    for (i = 1; i <= nnz; ++i)
        Vscanf("%lf%ld%ld%*[^\\n]", &a[i-1], &irow[i-1], &icol[i-1]);

    /* Calculate incomplete LU factorization */

    f11dac(n, nnz, &a, &num, &irow, &icol, lfill, dtol, pstrat, milu,
        ipivp, ipivq, istr, idiag, &nnzc, &npivm, NAGERR_DEFAULT);

    /* Output original matrix */

    Vprintf(" Original Matrix \\n n      = %6ld\\n", n);
    Vprintf(" nnz      = %6ld\\n", nnz);

    for (i = 1; i <= nnz; ++i)
        Vprintf("%8ld%16.6e%8ld%8ld\\n", i, a[i-1], irow[i-1], icol[i-1]);
    Vprintf("\\n");

    /* Output details of the factorization */

    Vprintf(" Factorization \\n n      = %6ld\\n", n);
    Vprintf(" nnz      = %6ld\\n", nnzc);
    Vprintf(" npivm = %6ld\\n", npivm);

    for (i = nnz + 1; i <= nnz + nnzc; ++i)
        Vprintf("%8ld%16.6e%8ld%8ld\\n", i, a[i-1], irow[i-1], icol[i-1]);

    Vprintf("\\n          i          ipivp[i-1] ipivq[i-1] \\n"); /* */

    for (i = 1; i <= n; ++i)
        Vprintf("%10ld%10ld%10ld\\n", i, ipivp[i-1], ipivq[i-1]);

    NAG_FREE(istr);
    NAG_FREE(idiag);
    NAG_FREE(ipivp);
    NAG_FREE(ipivq);
    NAG_FREE(irow);
    NAG_FREE(icol);
    NAG_FREE(a);
    exit(EXIT_SUCCESS);
}

```

8.2. Program Data

f11dac Example Program Data

4		n	
11		nnz	
1	0.0	lfill, dtol	
	CompletePiv	pstrat	
	UnModFact	milu	
1.	1	2	
1.	1	3	
-1.	2	1	
2.	2	3	
2.	2	4	
3.	3	1	
-2.	3	4	
1.	4	1	
-2.	4	2	
1.	4	3	
1.	4	4	a[i-1], irow[i-1], icol[i-1], i=1,...,nnz

8.3. Program Results

f11dac Example Program Results

Original Matrix

```

n      =      4
nnz    =     11
  1    1.000000e+00    1    2
  2    1.000000e+00    1    3
  3   -1.000000e+00    2    1
  4    2.000000e+00    2    3
  5    2.000000e+00    2    4
  6    3.000000e+00    3    1
  7   -2.000000e+00    3    4
  8    1.000000e+00    4    1
  9   -2.000000e+00    4    2
 10    1.000000e+00    4    3
 11    1.000000e+00    4    4
    
```

Factorization

```

n      =      4
nnz    =     11
npivm  =      0
 12    1.000000e+00    1    1
 13    1.000000e+00    1    3
 14    3.333333e-01    2    2
 15   -6.666667e-01    2    4
 16   -3.333333e-01    3    2
 17    5.000000e-01    3    3
 18    6.666667e-01    3    4
 19   -2.000000e+00    4    1
 20    3.333333e-01    4    2
 21    1.500000e+00    4    3
 22   -3.000000e+00    4    4
    
```

```

i      ipivp[i-1] ipivq[i-1]
 1      1          2
 2      3          1
 3      2          3
 4      4          4
    
```