

## NAG Library Function Document

### nag\_sparse\_herm\_chol\_fac (f11jnc)

#### 1 Purpose

nag\_sparse\_herm\_chol\_fac (f11jnc) computes an incomplete Cholesky factorization of a complex sparse Hermitian matrix, represented in symmetric coordinate storage format. This factorization may be used as a preconditioner in combination with nag\_sparse\_herm\_chol\_sol (f11jqc).

#### 2 Specification

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

void nag_sparse_herm_chol_fac (Integer n, Integer nnz, Complex a[],
    Integer la, Integer irow[], Integer icol[], Integer lfill, double dtol,
    Nag_SparseSym_Fact mic, double dscale, Nag_SparseSym_Piv pstrat,
    Integer ipiv[], Integer istr[], Integer *nnzc, Integer *npivm,
    NagError *fail)
```

#### 3 Description

nag\_sparse\_herm\_chol\_fac (f11jnc) computes an incomplete Cholesky factorization (see Meijerink and Van der Vorst (1977)) of a complex sparse Hermitian  $n$  by  $n$  matrix  $A$ . It is designed specifically for positive definite matrices, but may also work for some mildly indefinite cases. The factorization is intended primarily for use as a preconditioner with the complex Hermitian iterative solver nag\_sparse\_herm\_chol\_sol (f11jqc).

The decomposition is written in the form

$$A = M + R$$

where

$$M = PLDL^H P^T$$

and  $P$  is a permutation matrix,  $L$  is lower triangular complex with unit diagonal elements,  $D$  is real diagonal 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 factorization may be modified in order to preserve row sums, and the diagonal elements may be perturbed to ensure that the preconditioner is positive definite. Diagonal pivoting may optionally be employed, either with a user-defined ordering, or using the Markowitz strategy (see Markowitz (1957)), which aims to minimize fill-in. For further details see Section 9.

The sparse matrix  $A$  is represented in symmetric coordinate storage (SCS) format (see Section 2.1.2 in the f11 Chapter Introduction). The array **a** stores all the nonzero elements of the lower triangular part of  $A$ , 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 matrix  $M$  is returned in terms of the SCS representation of the lower triangular matrix

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

## 4 References

Chan T F (1991) Fourier analysis of relaxed incomplete factorization preconditioners *SIAM J. Sci. Statist. Comput.* **12**(2) 668–680

Markowitz H M (1957) The elimination form of the inverse and its application to linear programming *Management Sci.* **3** 255–269

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

Salvini S A and Shaw G J (1995) An evaluation of new NAG Library solvers for large sparse symmetric linear systems *NAG Technical Report TR1/95*

Van der Vorst H A (1990) The convergence behaviour of preconditioned CG and CG-S in the presence of rounding errors *Lecture Notes in Mathematics* (eds O Axelsson and L Y Kolotilina) **1457** Springer-Verlag

## 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 lower triangular part of the matrix  $A$ .

*Constraint:*  $1 \leq \mathbf{nnz} \leq n \times (n + 1)/2$ .

3: **a[la]** – Complex *Input/Output*

*On entry:* the nonzero elements in the lower triangular part of 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_herm_sort (f11zpc)` may be used to order the elements in this way.

*On exit:* the first **nnz** elements of **a** contain the nonzero elements of  $A$  and the next **nnzc** elements contain the elements of the lower triangular matrix  $C$ . 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  $C$  (**nnzc** elements).

*Constraint:*  $la \geq 2 \times \mathbf{nnz}$ .

5: **irow[la]** – Integer *Input/Output*

6: **icol[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_herm_sort (f11zpc)`):

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

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

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

- 7: **lfill** – Integer *Input*  
*On entry:* if **lfill**  $\geq 0$  its value is the maximum level of fill allowed in the decomposition (see Section 9.2). A negative value of **lfill** indicates that **dtol** will be used to control the fill instead.
- 8: **dtol** – double *Input*  
*On entry:* if **lfill**  $< 0$ , **dtol** is used as a drop tolerance to control the fill-in (see Section 9.2); otherwise **dtol** is not referenced.  
*Constraint:* if **lfill**  $< 0$ , **dtol**  $\geq 0.0$ .
- 9: **mic** – Nag\_SparseSym\_Fact *Input*  
*On entry:* indicates whether or not the factorization should be modified to preserve row sums (see Section 9.3).  
**mic** = Nag\_SparseSym\_ModFact  
The factorization is modified.  
**mic** = Nag\_SparseSym\_UnModFact  
The factorization is not modified.  
*Constraint:* **mic** = Nag\_SparseSym\_ModFact or Nag\_SparseSym\_UnModFact.
- 10: **dscale** – double *Input*  
*On entry:* the diagonal scaling argument. All diagonal elements are multiplied by the factor  $(1.0 + \mathbf{dscale})$  at the start of the factorization. This can be used to ensure that the preconditioner is positive definite. See also Section 9.3.
- 11: **pstrat** – Nag\_SparseSym\_Piv *Input*  
*On entry:* specifies the pivoting strategy to be adopted.  
**pstrat** = Nag\_SparseSym\_NoPiv  
No pivoting is carried out.  
**pstrat** = Nag\_SparseSym\_MarkPiv  
Diagonal pivoting aimed at minimizing fill-in is carried out, using the Markowitz strategy (see Markowitz (1957)).  
**pstrat** = Nag\_SparseSym\_UserPiv  
Diagonal pivoting is carried out according to the user-defined input array **ipiv**.  
*Suggested value:* **pstrat** = Nag\_SparseSym\_MarkPiv.  
*Constraint:* **pstrat** = Nag\_SparseSym\_NoPiv, Nag\_SparseSym\_MarkPiv or Nag\_SparseSym\_UserPiv.
- 12: **ipiv**[**n**] – Integer *Input/Output*  
*On entry:* if **pstrat** = Nag\_SparseSym\_UserPiv, **ipiv**[ $i - 1$ ] must specify the row index of the diagonal element to be used as a pivot at elimination stage  $i$ . Otherwise **ipiv** need not be initialized.  
*Constraint:* if **pstrat** = Nag\_SparseSym\_UserPiv, **ipiv** must contain a valid permutation of the integers on  $[1, \mathbf{n}]$ .  
*On exit:* the pivot indices. If **ipiv**[ $i - 1$ ] =  $j$ , the diagonal element in row  $j$  was used as the pivot at elimination stage  $i$ .
- 13: **istr**[**n** + 1] – Integer *Output*  
*On exit:* **istr**[ $i - 1$ ] – 1, for  $i = 1, 2, \dots, \mathbf{n}$ , is the starting address in the arrays **a**, **row** and **icol** of row  $i$  of the matrix  $C$ . **istr**[**n**] – 1 is the address of the last nonzero element in  $C$  plus one.

- 14: **nnzc** – Integer \* *Output*  
*On exit:* the number of nonzero elements in the lower triangular matrix  $C$ .
- 15: **npivm** – Integer \* *Output*  
*On exit:* the number of pivots which were modified during the factorization to ensure that  $M$  was positive definite. 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_herm_chol_fac` (f11jnc) again with an increased value of either **lfill** or **dscale**. See also Sections 9.3 and 9.4.
- 16: **fail** – NagError \* *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.

### NE\_BAD\_PARAM

On entry, argument  $\langle value \rangle$  had an illegal value.

### NE\_INT

On entry, **n** =  $\langle value \rangle$ .

Constraint: **n**  $\geq 1$ .

On entry, **nnz** =  $\langle value \rangle$ .

Constraint: **nnz**  $\geq 1$ .

### NE\_INT\_2

On entry, **la** =  $\langle value \rangle$  and **nnz** =  $\langle value \rangle$ .

Constraint: **la**  $\geq 2 \times \mathbf{nnz}$ .

On entry, **nnz** =  $\langle value \rangle$  and **n** =  $\langle value \rangle$ .

Constraint: **nnz**  $\leq \mathbf{n} \times (\mathbf{n} + 1) / 2$

### 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.

A serious error has occurred in an internal call to `nag_sparse_herm_sort` (f11zpc). Check all function calls and array sizes. Seek expert help.

### NE\_INVALID\_ROWCOL\_PIVOT

On entry, a user-supplied value of **ipiv** is repeated.

On entry, a user-supplied value of **ipiv** lies outside the range  $[1, \mathbf{n}]$ .

### NE\_INVALID\_SCS

On entry,  $I = \langle value \rangle$ , **icol** $[I - 1] = \langle value \rangle$  and **irow** $[I - 1] = \langle value \rangle$ .

Constraint: **icol** $[I - 1] \geq 1$  and **icol** $[I - 1] \leq \mathbf{irow}[I - 1]$ .

On entry,  $I = \langle value \rangle$ , **irow** $[I - 1] = \langle value \rangle$  and **n** =  $\langle value \rangle$ .

Constraint: **irow** $[I - 1] \geq 1$  and **irow** $[I - 1] \leq \mathbf{n}$ .

**NE\_NOT\_STRICTLY\_INCREASING**

On entry,  $\mathbf{a}[i - 1]$  is out of order:  $i = \langle \text{value} \rangle$ .

On entry, the location ( $\mathbf{irow}[I - 1], \mathbf{icol}[I - 1]$ ) is a duplicate:  $I = \langle \text{value} \rangle$ . Consider calling `nag_sparse_herm_sort` (f11zpc) to reorder and sum or remove duplicates.

**NE\_REAL**

On entry,  $\mathbf{dtol} = \langle \text{value} \rangle$ .

Constraint:  $\mathbf{dtol} \geq 0.0$

**NE\_TOO\_SMALL**

The number of nonzero entries in the decomposition is too large. The decomposition has been terminated before completion. Either increase  $\mathbf{la}$ , or reduce the fill by setting  $\mathbf{pstrat} = \text{Nag-SparseSym-MarkPiv}$ , reducing  $\mathbf{lfill}$ , or increasing  $\mathbf{dtol}$ .

**7 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 diagonal 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  $\mathbf{lfill}$ , or by reducing  $\mathbf{dtol}$  with  $\mathbf{lfill} < 0$ .

If `nag_sparse_herm_chol_fac` (f11jnc) is used in combination with `nag_sparse_herm_chol_sol` (f11jqc), 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**

`nag_sparse_herm_chol_fac` (f11jnc) is not threaded by NAG in any implementation.

`nag_sparse_herm_chol_fac` (f11jnc) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the Users' Note for your implementation for any additional implementation-specific information.

**9 Further Comments****9.1 Timing**

The time taken for a call to `nag_sparse_herm_chol_fac` (f11jnc) is roughly proportional to  $\mathbf{nnzc}^2/\mathbf{n}$ .

**9.2 Control of Fill-in**

If  $\mathbf{lfill} \geq 0$ , the amount of fill-in occurring in the incomplete factorization is controlled by limiting the maximum 'level' of fill-in to  $\mathbf{lfill}$ . The original nonzero elements of  $A$  are defined to be of level 0. The fill level of a new nonzero 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  $\mathbf{lfill} < 0$ , the fill-in is controlled by means of the 'drop tolerance'  $\mathbf{dtol}$ . A potential fill-in element  $a_{ij}$  occurring in row  $i$  and column  $j$  will not be included if

$$|a_{ij}| < \mathbf{dtol} \times \sqrt{|a_{ii}a_{jj}|}.$$

For either method of control, any elements which are not included are discarded if **mic** = Nag\_SparseSym\_UnModFact, or subtracted from the diagonal element in the elimination row if **mic** = Nag\_SparseSym\_ModFact.

### 9.3 Choice of Arguments

There is unfortunately no choice of the various algorithmic arguments which is optimal for all types of complex Hermitian 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, **mic** = Nag\_SparseSym\_UnModFact and **dscale** = 0.0. If the value returned for **npivm** is significantly larger than zero, i.e., a large number of pivot modifications were required to ensure that  $M$  was positive definite, the preconditioner is not likely to be satisfactory. In this case increase either **lfill** or **dscale** until **npivm** falls to a value close to zero. Once suitable values of **lfill** and **dscale** have been found try setting **mic** = Nag\_SparseSym\_ModFact to see if any improvement can be obtained by using **modified** incomplete Cholesky.

nag\_sparse\_herm\_chol\_fac (f11jnc) is primarily designed for positive definite matrices, but may work for some mildly indefinite problems. If **npivm** cannot be satisfactorily reduced by increasing **lfill** or **dscale** then  $A$  is probably too indefinite for this function.

For certain classes of matrices (typically those arising from the discretization of elliptic or parabolic partial differential equations), the convergence rate of the preconditioned iterative solver can sometimes be significantly improved by using an incomplete factorization which preserves the row-sums of the original matrix. In these cases try setting **mic** = Nag\_SparseSym\_ModFact.

### 9.4 Direct Solution of positive definite Systems

Although it is not their primary purpose, nag\_sparse\_herm\_chol\_fac (f11jnc) and nag\_sparse\_herm\_precon\_ichol\_solve (f11jpc) may be used together to obtain a **direct** solution to a complex Hermitian positive definite linear system. To achieve this the call to nag\_sparse\_herm\_precon\_ichol\_solve (f11jpc) should be preceded by a **complete** Cholesky factorization

$$A = PLDL^H P^T = M.$$

A complete factorization is obtained from a call to nag\_sparse\_herm\_chol\_fac (f11jnc) with **lfill** < 0 and **dtol** = 0.0, provided **npivm** = 0 on exit. A nonzero value of **npivm** indicates that  $A$  is not positive definite, or is ill-conditioned. A factorization with nonzero **npivm** may serve as a preconditioner, but will not result in a direct solution. It is therefore **essential** to check the output value of **npivm** if a direct solution is required.

The use of nag\_sparse\_herm\_chol\_fac (f11jnc) and nag\_sparse\_herm\_precon\_ichol\_solve (f11jpc) as a direct method is illustrated in nag\_sparse\_herm\_precon\_ichol\_solve (f11jpc).

## 10 Example

This example reads in a complex sparse Hermitian matrix  $A$  and calls nag\_sparse\_herm\_chol\_fac (f11jnc) to compute an incomplete Cholesky factorization. It then outputs the nonzero elements of both  $A$  and  $C = L + D^{-1} - I$ .

The call to nag\_sparse\_herm\_chol\_fac (f11jnc) has **lfill** = 0, **mic** = Nag\_SparseSym\_UnModFact, **dscale** = 0.0 and **pstrat** = Nag\_SparseSym\_MarkPiv, giving an unmodified zero-fill factorization of an unperturbed matrix, with Markowitz diagonal pivoting.

### 10.1 Program Text

```
/* nag_sparse_herm_chol_fac (f11jnc) Example Program.
 *
 * Copyright 2011, Numerical Algorithms Group.
 *
 * Mark 23, 2011.
 */
```

```

#include <nag.h>
#include <nag_stdlib.h>
#include <naga02.h>
#include <nagf11.h>

int main(void)
{
    /* Scalars */
    Integer          exit_status = 0;
    double           dscale, dtol;
    Integer          i, la, lfill, n, nnz, nnzc, npivm;
    /* Arrays */
    Complex          *a = 0;
    Integer          *icol = 0, *ipiv = 0, *irow = 0, *istr = 0;
    char             nag_enum_arg[100];
    /* NAG types */
    Nag_SparseSym_Piv pstrat;
    Nag_SparseSym_Fact mic;
    NagError         fail;

    INIT_FAIL(fail);

    printf("nag_sparse_herm_chol_fac (f11jnc) Example Program Results\n");
    /* Skip heading in data file*/
    scanf("%*[\n]");
    /* Read algorithmic parameters*/
    scanf("%ld%*[\n]%ld%*[\n]", &n, &nnz);

    /* Allocate memory */
    la = 3 * nnz;
    if (
        !(a = NAG_ALLOC(la, Complex)) ||
        !(icol = NAG_ALLOC(la, Integer)) ||
        !(ipiv = NAG_ALLOC(n, Integer)) ||
        !(irow = NAG_ALLOC(la, Integer)) ||
        !(istr = NAG_ALLOC(n + 1, Integer))
    )
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
    scanf("%ld%lf%*[\n]", &llfill, &dtol);
    scanf("%99s%*[\n]", nag_enum_arg);
    /* nag_enum_name_to_value (x04nac).
     * Converts NAG enum member name to value
     */
    mic = (Nag_SparseSym_Fact) nag_enum_name_to_value(nag_enum_arg);
    scanf("%lf%*[\n]", &dscale);
    scanf("%99s%*[\n]", nag_enum_arg);
    pstrat = (Nag_SparseNsym_Piv) nag_enum_name_to_value(nag_enum_arg);

    /* Read the matrix a */
    for (i = 0; i < nnz; i++)
        scanf(" ( %lf , %lf ) %ld%ld%*[\n] ",
            &a[i].re, &a[i].im, &irow[i], &icol[i]);

    /* Calculate incomplete Cholesky factorization using
     * nag_sparse_herm_chol_fac (f11jnc).
     */
    nag_sparse_herm_chol_fac(n, nnz, a, la, irow, icol, lfill, dtol, mic, dscale,
        pstrat, ipiv, istr, &nnzc, &npivm, &fail);
    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_sparse_herm_chol_fac (f11jnc).\n%s\n",
            fail.message);
        exit_status = 1;
        goto END;
    }
}

```

```

/* Output original matrix*/
printf(" Original Matrix \n");
printf(" n = %4ld, nnz = %4ld\n", n, nnz);
printf("%8s%16s%23s%9s\n","i","a[i]","irow[i]","icol[i]");
for (i = 0; i < nnz; i++)
    printf("%8ld (%13.4e, %13.4e) %8ld %8ld \n",
        i, a[i].re, a[i].im, irow[i], icol[i]);
printf("\n");

/* Output details of the factorization*/
printf(" Factorization \n");
printf(" n = %4ld, nnzc = %4ld, npivm = %4ld\n",
    n, nnzc, npivm);
printf("%8s%16s%23s%9s\n","i","a[i]","irow[i]","icol[i]");
for (i = nnz; i < nnz + nnzc; i++)
    printf("%8ld (%13.4e, %13.4e) %8ld %8ld \n",
        i, a[i].re, a[i].im, irow[i], icol[i]);
printf("\n%8s%12s\n","i","ipiv[i-1]");
for (i = 1; i <= n; i++)
    printf("%8ld%8ld\n", i, ipiv[i-1]);

END:
NAG_FREE(a);
NAG_FREE(icol);
NAG_FREE(ipiv);
NAG_FREE(irow);
NAG_FREE(istr);
return exit_status;
}

```

## 10.2 Program Data

nag\_sparse\_herm\_chol\_fac (f11jnc) Example Program Data

```

7                : n
16               : nnz
0 0.0           : lfill, dtol
Nag_SparseSym_UnModFact : mic
0.0             : dscale
Nag_SparseSym_MarkPiv  : pstrat
( 6., 0.) 1      1
( 1.,-2.) 2      1
( 9., 0.) 2      2
( 4., 0.) 3      3
( 2., 2.) 4      2
( 5., 0.) 4      4
( 0.,-1.) 5      1
( 1., 0.) 5      4
( 4., 0.) 5      5
( 1., 3.) 6      2
( 0.,-2.) 6      5
( 3., 0.) 6      6
( 2., 1.) 7      1
(-1., 0.) 7      2
(-3.,-1.) 7      3
( 5., 0.) 7      7      : a[i], irow[i], icol[i] i=0,...,nnz-1

```

## 10.3 Program Results

nag\_sparse\_herm\_chol\_fac (f11jnc) Example Program Results

```

Original Matrix
n = 7, nnz = 16
i      a[i]      irow[i]  icol[i]
0 ( 6.0000e+00, 0.0000e+00) 1      1
1 ( 1.0000e+00, -2.0000e+00) 2      1
2 ( 9.0000e+00, 0.0000e+00) 2      2
3 ( 4.0000e+00, 0.0000e+00) 3      3
4 ( 2.0000e+00, 2.0000e+00) 4      2
5 ( 5.0000e+00, 0.0000e+00) 4      4
6 ( 0.0000e+00, -1.0000e+00) 5      1
7 ( 1.0000e+00, 0.0000e+00) 5      4

```



8 (	4.0000e+00,	0.0000e+00)	5	5
9 (	1.0000e+00,	3.0000e+00)	6	2
10 (	0.0000e+00,	-2.0000e+00)	6	5
11 (	3.0000e+00,	0.0000e+00)	6	6
12 (	2.0000e+00,	1.0000e+00)	7	1
13 (	-1.0000e+00,	0.0000e+00)	7	2
14 (	-3.0000e+00,	-1.0000e+00)	7	3
15 (	5.0000e+00,	0.0000e+00)	7	7

Factorization

```

n = 7, nnzc = 16, npivm = 0
  i      a[i]
16 ( 2.5000e-01, 0.0000e+00)  irow[i] icol[i]
17 ( 2.0000e-01, 0.0000e+00)      1      1
18 ( 2.0000e-01, 0.0000e+00)      2      2
19 ( 2.6316e-01, 0.0000e+00)      3      2
20 ( 2.6316e-01, 0.0000e+00)      3      3
21 ( 0.0000e+00, -5.2632e-01)      4      3
22 ( 5.1351e-01, 0.0000e+00)      4      4
23 ( 0.0000e+00, 2.6316e-01)      5      3
24 ( 1.7431e-01, 0.0000e+00)      5      5
25 ( -7.5000e-01, -2.5000e-01)     6      1
26 ( 3.4862e-01, 1.7431e-01)      6      5
27 ( 6.1408e-01, 0.0000e+00)      6      6
28 ( 4.0000e-01, -4.0000e-01)     7      2
29 ( 5.1351e-01, -1.5405e+00)     7      4
30 ( 1.7431e-01, -3.4862e-01)     7      5
31 ( -6.1408e-01, 5.3521e-01)     7      6
    ( 3.1974e+00, 0.0000e+00)     7      7

```

```

  i      ipiv[i-1]
 1         3
 2         4
 3         5
 4         6
 5         1
 6         7
 7         2

```

---