NAG Library Function Document

nag_sparse_herm_precon_ichol_solve (f11jpc)

1 Purpose

nag_sparse_herm_precon_ichol_solve (f11jpc) solves a system of complex linear equations involving the incomplete Cholesky preconditioning matrix generated by nag_sparse_herm_chol_fac (f11jnc).

2 Specification

```
#include <nag.h>
#include <nagf11.h>
void nag_sparse_herm_precon_ichol_solve (Integer n, const Complex a[],
     Integer la, const Integer irow[], const Integer icol[],
     const Integer ipiv[], const Integer istr[],
     Nag_SparseSym_CheckData check, const Complex y[], Complex x[],
     NagError *fail)
```

3 Description

nag_sparse_herm_precon_ichol_solve (f11jpc) solves a system of linear equations

$$Mx = y$$

involving the preconditioning matrix $M = PLDL^{\mathrm{H}}P^{\mathrm{T}}$, corresponding to an incomplete Cholesky decomposition of a complex sparse Hermitian matrix stored in symmetric coordinate storage (SCS) format (see Section 2.1.2 in the f11 Chapter Introduction), as generated by nag_sparse_herm_chol_fac (f11jnc).

In the above decomposition L is a complex lower triangular sparse matrix with unit diagonal, D is a real diagonal matrix and P is a permutation matrix. L and D are supplied to nag_sparse_herm_precon_ichol_solve (f11jpc) through the matrix

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

which is a lower triangular n by n complex sparse matrix, stored in SCS format, as returned by nag_sparse_herm_chol_fac (f11jnc). The permutation matrix P is returned from nag_sparse_herm_chol_fac (f11jnc) via the array **ipiv**.

nag_sparse_herm_precon_ichol_solve (f11jpc) may also be used in combination with nag_sparse_herm_chol_fac (f11jnc) to solve a sparse complex Hermitian positive definite system of linear equations directly (see nag_sparse_herm_chol_fac (f11jnc)). This is illustrated in Section 10.

4 References

None.

5 Arguments

1: **n** – Integer *Input*

On entry: n, the order of the matrix M. This **must** be the same value as was supplied in the preceding call to nag_sparse_herm_chol_fac (f11jnc).

Constraint: **n** ≥ 1.

2: **a**[**la**] – const Complex *Input*

On entry: the values returned in the array **a** by a previous call to nag_sparse_herm_chol_fac (f11jnc).

3: **la** – Integer *Input*

On entry: the dimension of the arrays **a**, **irow** and **icol**. This **must** be the same value supplied in the preceding call to nag_sparse_herm_chol_fac (f11jnc).

4: **irow**[**la**] – const Integer *Input*
5: **icol**[**la**] – const Integer *Input*
6: **ipiv**[**n**] – const Integer *Input*
7: **istr**[**n** + **1**] – const Integer *Input*

On entry: the values returned in arrays **irow**, **icol**, **ipiv** and **istr** by a previous call to nag_sparse_herm_chol_fac (f11jnc).

8: **check** – Nag_SparseSym_CheckData *Input*

On entry: specifies whether or not the input data should be checked.

check = Nag_SparseSym_Check
Checks are carried out on the values of **n**, **irow**, **icol**, **ipiv** and **istr**.

check = Nag_SparseSym_NoCheck
None of these checks are carried out.

Constraint: **check** = Nag_SparseSym_Check or Nag_SparseSym_NoCheck.

9: **y**[**n**] – const Complex *Input*

On entry: the right-hand side vector y.

10: **x**[**n**] – Complex *Output*

On exit: the solution vector x.

11: **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 ⟨*value*⟩ had an illegal value.

NE_INT

On entry, **n** = ⟨*value*⟩.
Constraint: **n** ≥ 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.

NE_INVALID_ROWCOL_PIVOT

Check that **a**, **irow**, **icol**, **ipiv** and **istr** have not been corrupted between calls to nag_sparse_herm_chol_fac (f11jnc) and nag_sparse_herm_precon_ichol_solve (f11jpc).

NE_INVALID_SCS

Check that **a**, **irow**, **icol**, **ipiv** and **istr** have not been corrupted between calls to nag_sparse_herm_chol_fac (f11jnc) and nag_sparse_herm_precon_ichol_solve (f11jpc).

NE_INVALID_SCS_PRECOND

Check that **a**, **irow**, **icol**, **ipiv** and **istr** have not been corrupted between calls to nag_sparse_herm_chol_fac (f11jnc) and nag_sparse_herm_precon_ichol_solve (f11jpc).

NE_NOT_STRICTLY_INCREASING

Check that **a**, **irow**, **icol**, **ipiv** and **istr** have not been corrupted between calls to nag_sparse_herm_chol_fac (f11jnc) and nag_sparse_herm_precon_ichol_solve (f11jpc).

7 Accuracy

The computed solution x is the exact solution of a perturbed system of equations $(M + \delta M)x = y$, where

$$|\delta M| \leq c(n)\epsilon P|L||D|\left|L^{\mathrm{H}}\right|P^{\mathrm{T}},$$

$c(n)$ is a modest linear function of n, and ϵ is the ***machine precision***.

8 Parallelism and Performance

Not applicable.

9 Further Comments

9.1 Timing

The time taken for a call to nag_sparse_herm_precon_ichol_solve (f11jpc) is proportional to the value of **nnzc** returned from nag_sparse_herm_chol_fac (f11jnc).

10 Example

This example reads in a complex sparse Hermitian positive definite matrix A and a vector y. It then calls nag_sparse_herm_chol_fac (f11jnc), with **lfill** $= -1$ and **dtol** $= 0.0$, to compute the **complete** Cholesky decomposition of A:

$$A = PLDL^{\mathrm{H}}P^{\mathrm{T}}.$$

Finally it calls nag_sparse_herm_precon_ichol_solve (f11jpc) to solve the system

$$PLDL^{\mathrm{H}}P^{\mathrm{T}}x = y.$$

10.1 Program Text

```
/* nag_sparse_herm_precon_ichol_solve (f11jpc) 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, *x = 0, *y = 0;
  Integer                  *icol = 0, *ipiv = 0, *irow = 0, *istr = 0;
  /* NAG types */
  Nag_SparseSym_Fact       mic;
  Nag_SparseNsym_Piv       pstrat;
  Nag_SparseSym_CheckData  check;
  NagError                 fail;

  INIT_FAIL(fail);

  printf("nag_sparse_herm_precon_ichol_solve (f11jpc) Example Program Results");
  printf("\n\n");
  /* Skip heading in data file*/
  scanf("%*[^\n]");
  scanf("%ld%*[^\n]", &n);
  scanf("%ld%*[^\n]", &nnz);

  /* Allocate memory */
  la = 3 * nnz;
  if (
      !(a = NAG_ALLOC(la, Complex)) ||
      !(x = NAG_ALLOC(n, Complex)) ||
      !(y = NAG_ALLOC(n, 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;
    }
  /* Read the matrix a */
  for (i = 0; i <= nnz - 1; i++)
    scanf(" ( %lf , %lf ) %ld%ld%*[^\n] ",
          &a[i].re, &a[i].im, &irow[i], &icol[i]);
  /* Read the vector y*/
  for (i = 0; i <= n - 1; i++)
    scanf(" ( %lf , %lf ) ", &y[i].re, &y[i].im);

  lfill = -1;
  dtol = 0.0;
  dscale = 0.0;
  mic = Nag_SparseSym_UnModFact;
  pstrat = Nag_SparseSym_MarkPiv;
  /* Calculate 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;
    }
  /* Check the output value of npivm */
  if (npivm != 0)
    printf("Factorization is not complete \n");
  else
    {
```

```
      /* Solve complex linear system involving incomplete Cholesky factorization
       *
       *              H T
       *       P L D L P x = y
       *
       * using nag_sparse_herm_precon_ichol_solve (f11jpc).
       */
      check = Nag_SparseSym_Check;
      nag_sparse_herm_precon_ichol_solve(n, a, la, irow, icol, ipiv, istr,
                                         check, y, x, &fail);
      if (fail.code != NE_NOERROR) {
        printf("Error from nag_sparse_herm_precon_ichol_solve (f11jpc).\n%s\n",
               fail.message);
        exit_status = 2;
        goto END;
      }
      /* Output results*/
      printf("Solution of linear system \n");
      for (i = 0; i <= n - 1; i++)
        printf(" (%13.4e, %13.4e) \n", x[i].re, x[i].im);
    }

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

10.2 Program Data

```
nag_sparse_herm_precon_ichol_solve (f11jpc) Example Program Data
  9                    : n
 23                    : nnz
 ( 6., 0.)   1    1
 (-1., 1.)   2    1
 ( 6., 0.)   2    2
 ( 0., 1.)   3    2
 ( 5., 0.)   3    3
 ( 5., 0.)   4    4
 ( 2.,-2.)   5    1
 ( 4., 0.)   5    5
 ( 1., 1.)   6    3
 ( 2., 0.)   6    4
 ( 6., 0.)   6    6
 (-4., 3.)   7    2
 ( 0., 1.)   7    5
 (-1., 0.)   7    6
 ( 6., 0.)   7    7
 (-1.,-1.)   8    4
 ( 0.,-1.)   8    6
 ( 9., 0.)   8    8
 ( 1., 3.)   9    1
 ( 1., 2.)   9    5
 (-1., 0.)   9    6
 ( 1., 4.)   9    8
 ( 9., 0.)   9    9      : a[i], irow[i], icol[i], i=0,...,nnz-1
 ( 8.,54.) (-10.,-92.)
 (25.,27.) (26., -28.)
 (54.,12.) (26.,-22.)
 (47.,65.) (71.,-57.)
 (60.,70.)               : y[i], i=0,...,n-1
```

10.3 Program Results

```
nag_sparse_herm_precon_ichol_solve (f11jpc) Example Program Results

Solution of linear system
 (   1.0000e+00,    9.0000e+00)
 (   2.0000e+00,   -8.0000e+00)
 (   3.0000e+00,    7.0000e+00)
 (   4.0000e+00,   -6.0000e+00)
 (   5.0000e+00,    5.0000e+00)
 (   6.0000e+00,   -4.0000e+00)
 (   7.0000e+00,    3.0000e+00)
 (   8.0000e+00,   -2.0000e+00)
 (   9.0000e+00,    1.0000e+00)
```