NAG Library Function Document

nag_sparse_sym_precon_ichol_solve (f11jbc)

1 Purpose

nag_sparse_sym_precon_ichol_solve (f11jbc) solves a system of linear equations involving the incomplete Cholesky preconditioning matrix generated by nag_sparse_sym_chol_fac (f11jac).

2 Specification

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

void nag_sparse_sym_precon_ichol_solve (Integer n, const double a[],
           Integer la, const Integer irow[], const Integer icol[],
           const Integer ipiv[], const Integer istr[],
           Nag_SparseSym_CheckData check, const double y[], double x[],
           NagError *fail)
```

3 Description

nag_sparse_sym_precon_ichol_solve (f11jbc) solves a system of linear equations

\[ Mx = y \]

involving the preconditioning matrix \( M = PLDL^T P^T \), corresponding to an incomplete Cholesky decomposition of a sparse symmetric matrix stored in symmetric coordinate storage (SCS) format (see Section 2.1.2 in the f11 Chapter Introduction), as generated by nag_sparse_sym_chol_fac (f11jac).

In the above decomposition \( L \) is a lower triangular sparse matrix with unit diagonal, \( D \) is a diagonal matrix and \( P \) is a permutation matrix. \( L \) and \( D \) are supplied to nag_sparse_sym_precon_ichol_solve (f11jbc) through the matrix

\[ C = L + D^{-1} - I \]

which is a lower triangular \( n \) by \( n \) sparse matrix, stored in SCS format, as returned by nag_sparse_sym_chol_fac (f11jac). The permutation matrix \( P \) is returned from nag_sparse_sym_chol_fac (f11jac) via the array ipiv.

It is envisaged that a common use of nag_sparse_sym_precon_ichol_solve (f11jbc) will be to carry out the preconditioning step required in the application of nag_sparse_sym_basic_solver (f11gec) to sparse symmetric linear systems. nag_sparse_sym_precon_ichol_solve (f11jbc) is used for this purpose by the Black Box function nag_sparse_sym_chol_sol (f11jec).

nag_sparse_sym_precon_ichol_solve (f11jbc) may also be used in combination with nag_sparse_sym_chol_fac (f11jac) to solve a sparse symmetric positive definite system of linear equations directly (see Section 9.4 in nag_sparse_sym_chol_fac (f11jac)). This use of nag_sparse_sym_precon_ichol_solve (f11jbc) is demonstrated in Section 10.

4 References

None.
5 Arguments

1: \textbf{n} – Integer \hspace{1cm} \textit{Input}
\textit{On entry:} \(n\), the order of the matrix \(M\). This \textbf{must} be the same value as was supplied in the preceding call to \texttt{nag_sparse_sym_chol_fac} (f11jac).
\textit{Constraint:} \(n \geq 1\).

2: \textbf{a}[\textbf{la}] – const double \hspace{1cm} \textit{Input}
\textit{On entry:} the values returned in the array \textbf{a} by a previous call to \texttt{nag_sparse_sym_chol_fac} (f11jac).

3: \textbf{la} – Integer \hspace{1cm} \textit{Input}
\textit{On entry:} the dimension of the arrays \textbf{a}, \textbf{irow} and \textbf{icol}. This \textbf{must} be the same value returned by the preceding call to \texttt{nag_sparse_sym_chol_fac} (f11jac).

4: \textbf{irow}[\textbf{la}] – const Integer \hspace{1cm} \textit{Input}
5: \textbf{icol}[\textbf{la}] – const Integer \hspace{1cm} \textit{Input}
6: \textbf{ipiv}[\textbf{n}] – const Integer \hspace{1cm} \textit{Input}
7: \textbf{istr}[\textbf{n} + 1] – const Integer \hspace{1cm} \textit{Input}
\textit{On entry:} the values returned in arrays \textbf{irow}, \textbf{icol}, \textbf{ipiv} and \textbf{istr} by a previous call to \texttt{nag_sparse_sym_chol_fac} (f11jac).

8: \textbf{check} – \texttt{Nag_SparseSym_CheckData} \hspace{1cm} \textit{Input}
\textit{On entry:} specifies whether or not the input data should be checked.
\textbf{check} = \texttt{Nag_SparseSym_Check}
Checks are carried out on the values of \textbf{n}, \textbf{irow}, \textbf{icol}, \textbf{ipiv} and \textbf{istr}.
\textbf{check} = \texttt{Nag_SparseSym_NoCheck}
No checks are carried out.
\textit{Constraint:} \textbf{check} = \texttt{Nag_SparseSym_Check} or \texttt{Nag_SparseSym_NoCheck}.

9: \textbf{y}[\textbf{n}] – const double \hspace{1cm} \textit{Input}
\textit{On entry:} the right-hand side vector \(y\).

10: \textbf{x}[\textbf{n}] – double \hspace{1cm} \textit{Output}
\textit{On exit:} the solution vector \(x\).

11: \textbf{fail} – \texttt{NagError *} \hspace{1cm} \textit{Input/Output}
The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

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

\textbf{NE_BAD_PARAM}
On entry, argument \langle\textit{value}\rangle had an illegal value.
On entry, \( n = \langle \text{value} \rangle \).
Constraint: \( n \geq 1 \).

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.

Check that \( a, \text{irow, icol, ipiv} \) and \( \text{istr} \) have not been corrupted between calls to
\( \text{nag_sparse_sym_chol_fac (f11jac)} \) and \( \text{nag_sparse_sym_precon_ichol_solve (f11jbc)} \).

Check that \( a, \text{irow, icol, ipiv} \) and \( \text{istr} \) have not been corrupted between calls to
\( \text{nag_sparse_sym_chol_fac (f11jac)} \) and \( \text{nag_sparse_sym_precon_ichol_solve (f11jbc)} \).

Check that \( a, \text{irow, icol, ipiv} \) and \( \text{istr} \) have not been corrupted between calls to
\( \text{nag_sparse_sym_chol_fac (f11jac)} \) and \( \text{nag_sparse_sym_precon_ichol_solve (f11jbc)} \).

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.

Check that \( a, \text{irow, icol, ipiv} \) and \( \text{istr} \) have not been corrupted between calls to
\( \text{nag_sparse_sym_chol_fac (f11jac)} \) and \( \text{nag_sparse_sym_precon_ichol_solve (f11jbc)} \).

The computed solution \( x \) is the exact solution of a perturbed system of equations \( (M + \delta M)x = y \),
where
\[
|\delta M| \leq c(n)\varepsilon P\|L\|D\|L^T\|P^T,
\]
\( c(n) \) is a modest linear function of \( n \), and \( \varepsilon \) is the \textit{machine precision}.

Not applicable.

It is expected that a common use of \( \text{nag_sparse_sym_precon_ichol_solve (f11jbc)} \) will be to carry out
the preconditioning step required in the application of \( \text{nag_sparse_sym_basic_solver (f11gec)} \) to sparse
symmetric linear systems. In this situation \( \text{nag_sparse_sym_precon_ichol_solve (f11jbc)} \) is likely to be
called many times with the same matrix $M$. In the interests of both reliability and efficiency, you are recommended to set \texttt{check} = \texttt{NagSparseSym\_Check} for the first of such calls, and to set \texttt{check} = \texttt{NagSparseSym\_NoCheck} for all subsequent calls.

10 Example

This example reads in a symmetric positive definite sparse matrix $A$ and a vector $y$. It then calls \texttt{nag\_sparse\_sym\_chol\_fac (f11jac)}, with \texttt{lfill} = $-1$ and \texttt{dtol} = 0.0, to compute the \texttt{complete} Cholesky decomposition of $A$:

$$A = PLDL^TP^T.$$  

Then it calls \texttt{nag\_sparse\_sym\_precon\_ichol\_solve (f11jbc)} to solve the system

$$PLDL^TP^Tx = y.$$  

It then repeats the exercise for the same matrix permuted with the bandwidth-reducing Reverse Cuthill–McKee permutation, calculated with \texttt{nag\_sparse\_sym\_rcm (f11yec)}.

10.1 Program Text

```c
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf11.h>
#include <nagm01.h>

void do_rcm(Integer n, Integer nnz, Integer *irow, Integer *icol,
             double *a, double *y, Integer *istr, Integer *perm_fwd,
             Integer *perm_inv) ;

int main(void)
{
    /* Scalars */
    Integer exit_status = 0;
    double dscale, dtol;
    Integer i, la, lfill, n, nnz, nnzc, npivm;
    /* Arrays */
    double *a = 0, *x = 0, *y = 0;
    Integer *icol = 0, *ipiv = 0, *irow = 0, *istr = 0,
    *perm_fwd = 0, *perm_inv = 0;
    /* NAG types */
    Nag_SparseSym_Fact mic;
    Nag_SparseSym_Piv pstrat;
    Nag_SparseSym_CheckData check;
    Nag_Sparse_Comm comm;
    NagError fail;

    INIT_FAIL(fail);
    printf("nag_sparse_sym_precon_ichol_solve (f11jbc) Example Program Results\n");
    printf("\n");
    /* Skip heading in data file*/
    #ifdef _WIN32
    scanf_s("%*[^\n"]);
    #else
    scanf("%*[^\n"]);
    #endif
    /* Read order of matrix and number of non-zero entries*/
    #ifdef _WIN32
    scanf_s("%NAG_IFMT%*[\n"]",&n);
    #else
    scanf("%NAG_IFMT%*[\n"]",&n);
    #endif
```

f11jbc

NAG Library Manual

f11jbc.4

Mark 25
```c
/* Allocate memory */
la = 3 * nnz;
if (a = NAG_ALLOC(la, double)) ||
(x = NAG_ALLOC(n, double)) ||
(y = NAG_ALLOC(n, double)) ||
(icol = NAG_ALLOC(la, Integer)) ||
(ipiv = NAG_ALLOC(n, Integer)) ||
(irow = NAG_ALLOC(la, Integer)) ||
(istr = NAG_ALLOC(n + 1, Integer)) ||
(perm_fwd = NAG_ALLOC(n, Integer)) ||
(perm_inv = NAG_ALLOC(n, Integer))
{
  printf("Allocation failure\n");
  exit_status = -1;
  goto END;
}

/* Read the matrix A*/
for (i = 0; i < nnz; i++)
#endif _WIN32
  scanf_s("%lf"NAG_IFMT"%lf"NAG_IFMT", &a[i], &irow[i], &icol[i]);
#else
  scanf("%lf"NAG_IFMT"%lf", &a[i], &irow[i], &icol[i]);
#endif

/* Read the vector y*/
for (i = 0; i < n ; i++)
#endif _WIN32
  scanf_s("%lf", &y[i]);
#else
  scanf("%lf", &y[i]);
#endif

lfill = -1;
dtol = 0.0;
dscale = 0.0;
mic = Nag_SparseSym_UnModFact;
pstrat = Nag_SparseSym_MarkPiv;
/* Calculate Cholesky factorization using
   * nag_sparse_sym_chol_fac (f11jac).
   */
  nag_sparse_sym_chol_fac(n, nnz, &a, &l, &irow, &icol, lfill, dtol, mic,
  dscale, pstrat, ipiv, istr, &nnzc, &npivm, &comm,
  &fail);
if (fail.code != NE_NOERROR)
{
  printf("Error from nag_sparse_sym_chol_fac (f11jac).\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 linear system involving incomplete Cholesky factorization
      * T T
      * P L D L P x = y
      * using nag_sparse_sym_precon_ichol_solve (f11jbc).
      */
```
check = Nag_SparseSym_Check;
nag_sparse_sym_precon_ichol_solve(n, a, la, irow, icol, ipiv, istr,
    check, y, x, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_sparse_sym_precon_ichol_solve (f11jbc).\n%s\n", fail.message);
    exit_status = 2;
    goto END;
}
/* Output results*/
printf(" Solution of linear system \n");
for (i = 0; i < n; i++)
    printf("%16.4e\n", x[i]);
printf("\n");

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

void do_rcm(Integer n, Integer nnz, Integer *irow, Integer *icol, 
    double *a, double *y, Integer *istr, Integer *perm_fwd, 
    Integer *perm_inv)
{
    Integer j, i, nnz_cs, nnz_scs, info[4], mask[1];
    double *yy;
    NagError fail;
    SET_FAIL(fail);
    yy = NAG_ALLOC(n,double);
    /* SCS to CS, must add the upper triangle entries. */
    j=nnz;
    for (i=0; i<nnz; i++) {
        if (irow[i]>icol[i])
            /* strictly lower triangle, add the transposed */
            a[j]=a[i];
            irow[j]=icol[i];
            icol[j]=irow[i];
            j++;
        }
    }
    nnz_cs = j;

    /* Compute reverse Cuthill-McKee permutation 
    * Compute reverse Cuthill-McKee permutation for bandwidth reduction 
    */
    do_rcm(n, nnz, irow, icol, a, y, istr, perm_fwd, perm_inv);
    SET_FAIL(fail);
    nag_sparse_sym_chol_fac(n, nnz, &a, &la, &irow, &icol, lfill, dtol, mic,
        dscale, pstrat, ipiv, istr, &nnzc, &npivm, &comm,
        &fail);
    if (npivm != 0) printf("Factorization is not complete \n");
    else {
        nag_sparse_sym_precon_ichol_solve(n, a, la, irow, icol, ipiv, istr,
            check, y, x, &fail);
        printf(" Solution of linear system with Reverse Cuthill-McKee\n");
        for (i = 0; i < n; i++)
            printf("%16.4e\n", x[perm_inv[i]-1]);
        printf("\n");
    }
}

f11jbc.6 Mark 25

f11jbc

NAG Library Manual
/* Reorder, CS to CCS, icolzp in istr */
nag_sparse_nsym_sort(n, &nnz_cs, a, icol, irow, Nag_SparseNsym_FailDups,
    Nag_SparseNsym_FailZeros, istr, &fail);

/* Calculate reverse Cuthill-McKee */
nag_sparse_sym_rcm(n, nnz_cs, istr, irow, lopts, mask, perm_fwd, info, &fail);

/* compute inverse perm, in perm_inv */
for (i=0; i<n; i++) perm_inv[perm_fwd[i]-1] = i+1;

/* Apply permutation on column/row indices */
for (i=0; i<nnz_cs ; i++) {
    icol[i] = perm_inv[icol[i]-1];
    irow[i] = perm_inv[irow[i]-1];
}

/* restrict to lower triangle, SCS format
 * copying entries upwards */
j=0;
for (i=0; i<nnz_cs; i++) {
    if (irow[i]>=icol[i]) {
        /* non-upper triangle, bubble up */
        a[j] = a[i];
        icol[j] = icol[i];
        irow[j] = irow[i];
        j++;
    }
}
nnz_scs = j;

/* sort */
nag_sparse_sym_sort(n,&nnz_scs, a, irow, icol, Nag_SparseSym_SumDups,
    Nag_SparseSym_KeepZeros, istr, &fail);

/* permute rhs vector */
for (i=0; i<n; i++) yy[i] = y[perm_fwd[i]-1];
for (i=0; i<n; i++) y[i] = yy[i];
NAG_FREE(yy);

10.2 Program Data

nag_sparse_sym_precon_ichol_solve (f11jbc) Example Program Data
9
23 : n
4. 1 1
-1. 2 1
6. 2 2
1. 3 2
2. 3 3
3. 4 4
2. 5 1
4. 5 5
1. 6 3
2. 6 4
6. 6 6
-4. 7 2
1. 7 5
-1. 7 6
6. 7 7
-1. 8 4
-1. 8 6
3. 8 8
1. 9 1
1. 9 5
-1. 9 6
1. 9 8
4. 9 9
4.10 -2.94 1.41
2.53 4.35 1.29
5.01 0.52 4.57 : y[i], i=0,...,n-1
10.3 Program Results

nag_sparse_sym_precon_ichol_solve (f11jbc) Example Program Results
Solution of linear system
7.0000e-01
1.6000e-01
5.2000e-01
7.7000e-01
2.8000e-01
2.1000e-01
9.3000e-01
2.0000e-01
9.00000e-01

Solution of linear system with Reverse Cuthill-McKee
7.0000e-01
1.6000e-01
5.2000e-01
7.7000e-01
2.8000e-01
2.1000e-01
9.3000e-01
2.0000e-01
9.00000e-01