NAG Toolbox

nag_sparse_real_symm_precon_ichol_solve (f11jb)

1 Purpose

nag_sparse_real_symm_precon_ichol_solve (f11jb) solves a system of linear equations involving the incomplete Cholesky preconditioning matrix generated by nag_sparse_real_symm_precon_ichol (f11ja).

2 Syntax

```
[x, ifail] = nag_sparse_real_symm_precon_ichol_solve(a, irow, icol, ipiv, istr, check, y, 'n', n, 'la', la)
[x, ifail] = f11jb(a, irow, icol, ipiv, istr, check, y, 'n', n, 'la', la)
```

3 Description

nag_sparse_real_symm_precon_ichol_solve (f11jb) solves a system of linear equations

$$Mx = y$$

involving the preconditioning matrix $M = PLDL^{\mathrm{T}}P^{\mathrm{T}}$, 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_real_symm_precon_ichol (f11ja).

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_real_symm_precon_ichol_solve (f11jb) 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_real_symm_precon_ichol (f11ja). The permutation matrix P is returned from nag_sparse_real_symm_precon_ichol (f11ja) via the array **ipiv**.

It is envisaged that a common use of nag_sparse_real_symm_precon_ichol_solve (f11jb) will be to carry out the preconditioning step required in the application of nag_sparse_real_symm_basic_solver (f11ge) to sparse symmetric linear systems. nag_sparse_real_symm_precon_ichol_solve (f11jb) is used for this purpose by the Black Box function nag_sparse_real_symm_solve_ichol (f11jc).

nag_sparse_real_symm_precon_ichol_solve (f11jb) may also be used in combination with nag_sparse_real_symm_precon_ichol (f11ja) to solve a sparse symmetric positive definite system of linear equations directly (see Section 9.4 in nag_sparse_real_symm_precon_ichol (f11ja)). This use of nag_sparse_real_symm_precon_ichol_solve (f11jb) is demonstrated in Section 10.

4 References

None.

5 Parameters

5.1 Compulsory Input Parameters

1: **a**(**la**) – REAL (KIND=nag_wp) array

The values returned in the array **a** by a previous call to nag_sparse_real_symm_precon_ichol (f11ja).

2: **irow**(**la**) – INTEGER array
3: **icol**(**la**) – INTEGER array
4: **ipiv**(**n**) – INTEGER array
5: **istr**(**n** + **1**) – INTEGER array

The values returned in arrays **irow**, **icol**, **ipiv** and **istr** by a previous call to nag_sparse_real_symm_precon_ichol (f11ja).

6: **check** – CHARACTER(1)

Specifies whether or not the input data should be checked.

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

check = 'N'
No checks are carried out.

See also Section 9.2.

Constraint: **check** = 'C' or 'N'.

7: **y**(**n**) – REAL (KIND=nag_wp) array

The right-hand side vector y.

5.2 Optional Input Parameters

1: **n** – INTEGER

Default: the dimension of the arrays **ipiv**, **y**. (An error is raised if these dimensions are not equal.)

n, the order of the matrix M. This **must** be the same value as was supplied in the preceding call to nag_sparse_real_symm_precon_ichol (f11ja).

Constraint: **n** ≥ 1.

2: **la** – INTEGER

Default: the dimension of the arrays **a**, **irow**, **icol**. (An error is raised if these dimensions are not equal.)

The dimension of the arrays **a**, **irow** and **icol**. this **must** be the same value returned by the preceding call to nag_sparse_real_symm_precon_ichol (f11ja).

5.3 Output Parameters

1: **x**(**n**) – REAL (KIND=nag_wp) array

The solution vector x.

2: **ifail** – INTEGER

ifail = 0 unless the function detects an error (see Section 5).

6 Error Indicators and Warnings

Errors or warnings detected by the function:

ifail = 1

On entry, **check** $\neq$ 'C' or 'N'.

ifail = 2

On entry, **n** < 1.

ifail $= 3$

On entry, the SCS representation of the preconditioning matrix M is invalid. Further details are given in the error message. Check that the call to nag_sparse_real_symm_precon_ichol_solve (f11jb) has been preceded by a valid call to nag_sparse_real_symm_precon_ichol (f11ja) and that the arrays **a**, **irow**, **icol**, **ipiv** and **istr** have not been corrupted between the two calls.

ifail $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

ifail $= -399$

Your licence key may have expired or may not have been installed correctly.

ifail $= -999$

Dynamic memory allocation failed.

7 Accuracy

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

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

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

8 Further Comments

8.1 Timing

The time taken for a call to nag_sparse_real_symm_precon_ichol_solve (f11jb) is proportional to the value of **nnzc** returned from nag_sparse_real_symm_precon_ichol (f11ja).

8.2 Use of check

It is expected that a common use of nag_sparse_real_symm_precon_ichol_solve (f11jb) will be to carry out the preconditioning step required in the application of nag_sparse_real_symm_basic_solver (f11ge) to sparse symmetric linear systems. In this situation nag_sparse_real_symm_precon_ichol_solve (f11jb) 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 **check** $=$ 'C' for the first of such calls, and to set **check** $=$ 'N' for all subsequent calls.

9 Example

This example reads in a symmetric positive definite sparse matrix A and a vector y. It then calls nag_sparse_real_symm_precon_ichol (f11ja), with **lfill** $= -1$ and **dtol** $= 0.0$, to compute the **complete** Cholesky decomposition of A:

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

Then it calls nag_sparse_real_symm_precon_ichol_solve (f11jb) to solve the system

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

It then repeats the exercise for the same matrix permuted with the bandwidth-reducing Reverse Cuthill–McKee permutation, calculated with nag_sparse_sym_rcm (f11ye).

9.1 Program Text

```
     function f11jb_example

fprintf('f11jb example results\n\n');

% Solve sparse symmetric system Ax = b using CG method with
% Incomplete Cholesky preconditioning (IC)

% Define A and b
n  = nag_int(9);
nz = nag_int(23);
a = zeros(3*nz,1);
irow = zeros(3*nz,1,nag_int_name);
icol = irow;
a(1:nz)    = [ 4    -1     6     1     2     3     2     4 ...
                1     2     6    -4     1    -1     6    -1 ...
               -1     3     1     1    -1     1     4 ];
irow(1:nz) = [ 1     2     2     3     3     4     5     5 ...
                6     6     6     7     7     7     7     8 ...
                8     8     9     9     9     9     9 ];
icol(1:nz) = [ 1     1     2     2     3     4     1     5 ...
                3     4     6     2     5     6     7     4 ...
                6     8     1     5     6     8     9 ];
b          = [4.10 -2.94  1.41 ...
              2.53  4.35  1.29 ...
              5.01  0.52  4.57];

% Setup IC factorization

lfill  = nag_int(-1);
dtol   = 0;
mic    = 'N';
dscale = 0;
ipiv   = zeros(n, 1, nag_int_name);
[a, irow, icol, ipiv, istr, nnzc, npivm, ifail] = ...
    f11ja(...
        nz, a, irow, icol, lfill, dtol, mic, dscale, ipiv);

% Solve Ax = b, using Cholesky factorization
[x, ifail] = f11jb(...
                  a, irow, icol, ipiv, istr, 'C', b);
fprintf('\n   Solution of linear System\n');
fprintf('%16.4f\n', x);

% Redo calculation after reverse Cuthill-McKee permutation for
% bandwidth reduction

% Reverse Cuthill-McKee
[irow,icol,a,perm_fwd,perm_inv] =  do_rcm(n,nz,irow,icol,a);

% Repeat factorization and solution
[a, irow, icol, ipiv, istr, nnzc, npivm, ifail] = ...
    f11ja(...
        nz, a, irow, icol, lfill, dtol, mic, dscale, ipiv);

% Permute rhs vector and solution (PAP^T)(Px) = Pb.
y = b(perm_fwd(:));
[u, ifail] = f11jb(...
                  a, irow, icol, ipiv, istr, 'C', y);

x = u(perm_inv(:));
fprintf('\n   Solution of linear System with reverse Cuthill-McKee\n');
fprintf('%16.4f\n', x);

function [irow_out,icol_out,a_out,perm_fwd,perm_inv] = ...
         do_rcm(n,nz,irow,icol,a)

  % Reverse Cuthill-McKee for symmetric matrix

  % Add upper triangle, sort and remove duplicate diagonal elements
```

```
irow(nz+1:2*nz) = icol(1:nz);
icol(nz+1:2*nz) = irow(1:nz);
a(nz+1:2*nz)    = a(1:nz);
nnz = nz + nz;
[nnz,a,icol,irow,istr,ifail] = f11za(n,nnz,a,icol,irow,'R','F');

% Reverse Cuthill-McKee
lopts(1:5) = [false false true true true];
mask = zeros(n,1,nag_int_name);
[perm_fwd, info, ifail] = f11ye(istr, irow, lopts, mask,'n',n);

% Inverse perm
perm_inv(perm_fwd(1:n)) = [1:n];

% Apply permutation on column/row indices
icol(1:nnz) = perm_inv(icol(1:nnz));
irow(1:nnz) = perm_inv(irow(1:nnz));

% collect only lower triangle
j = 0;
for i=1:nnz
  if icol(i)<=irow(i)
    j = j + 1;
    a(j) = a(i);
    irow(j) = irow(i);
    icol(j) = icol(i);
  end
end

nnz = nag_int(j);
% Sort collected lower triangle
[nnz, a, irow, icol, istr, ifail] = ...
f11zb(...
       n, nnz, a, irow, icol, 'S', 'K');

% copy to output arrays
a_out = zeros(3*nz,1);
irow_out = zeros(3*nz,1,nag_int_name);
icol_out = irow_out;
a_out(1:nnz) = a(1:nnz);
irow_out(1:nnz) = irow(1:nnz);
icol_out(1:nnz) = icol(1:nnz);
```

9.2 Program Results

```
    f11jb example results

 Solution of linear System
        0.7000
        0.1600
        0.5200
        0.7700
        0.2800
        0.2100
        0.9300
        0.2000
        0.9000

 Solution of linear System with reverse Cuthill-McKee
        0.7000
        0.1600
        0.5200
        0.7700
        0.2800
        0.2100
        0.9300
        0.2000
        0.9000
```