nag_sparse_sym_chol_sol (f11jcc)

1. Purpose
nag_sparse_sym_chol_sol (f11jcc) solves a real sparse symmetric system of linear equations, represented in symmetric coordinate storage format, using a conjugate gradient or Lanczos method, with incomplete Cholesky preconditioning.

2. Specification
#include <nag.h>
#include <nagf11.h>

void nag_sparse_sym_chol_sol(Nag_SparseSym_Method method, Integer n,
Integer nnz, double a[], Integer la, Integer irow[],
Integer icol[], Integer ipiv[], Integer istr[], double b[],
double tol, Integer maxitn, double x[], double *rnorm,
Integer *itn, Nag_Sparse_Comm *comm, NagError *fail)

3. Description
This routine solves a real sparse symmetric linear system of equations:

\[ Ax = b, \]

using a preconditioned conjugate gradient method (Meijerink and van der Vorst (1977)), or a preconditioned Lanczos method based on the algorithm SYMMLQ (Paige and Saunders (1975)). The conjugate gradient method is more efficient if \( A \) is positive-definite, but may fail to converge for indefinite matrices. In this case the Lanczos method should be used instead. For further details see Barrett et al. (1994).

nag_sparse_sym_chol_sol uses the incomplete Cholesky factorization determined by nag_sparse_sym_chol_fac (f11jac) as the preconditioning matrix. A call to nag_sparse_sym_chol_sol must always be preceded by a call to nag_sparse_sym_chol_fac (f11jac). Alternative preconditioners for the same storage scheme are available by calling nag_sparse_sym_sol (f11jec).

The matrix \( A \), and the preconditioning matrix \( M \), are represented in symmetric coordinate storage (SCS) format (see Section 2.1.2. of the Chapter Introduction) in the arrays \( a \), \( irow \) and \( icol \), as returned from nag_sparse_sym_chol_fac (f11jac). The array \( a \) holds the non-zero entries in the lower triangular parts of these matrices, while \( irow \) and \( icol \) hold the corresponding row and column indices.

4. Parameters
method
Input: specifies the iterative method to be used. The possible choices are:

if \( \text{method} = \text{Nag}\_\text{SparseSym}\_\text{CG} \) then the conjugate gradient method is used;
if \( \text{method} = \text{Nag}\_\text{SparseSym}\_\text{Lanczos} \) then the Lanczos method, SYMMLQ is used.

Constraint: \( \text{method} = \text{Nag}\_\text{SparseSym}\_\text{CG} \) or \( \text{Nag}\_\text{SparseSym}\_\text{Lanczos} \).

n
Input: the order of the matrix \( A \). This must be the same value as was supplied in the preceding call to nag_sparse_sym_chol_fac (f11jac).
Constraint: \( n \geq 1 \).

nnz
Input: the number of non-zero elements in the lower triangular part of the matrix \( A \). This must be the same value as was supplied in the preceding call to nag_sparse_sym_chol_fac (f11jac).
Constraint: \( 1 \leq \text{nnz} \leq n \times (n+1)/2 \).
**nag_sparse_sym_chol_sol**

**a[la]**
Input: the values returned in array \(a\) by a previous call to `nag_sparse_sym_chol_fac` (f11jac).

**la**
Input: the dimension of the arrays \(a\), \(irow\) and \(icol\), this must be the same value as returned by a previous call to `nag_sparse_sym_chol_fac` (f11jac).
Constraint: \(la \geq 2 \times nnz\).

**irow[la]**

**icol[la]**

**ipiv[n]**

**istr[n+1]**
Input: the values returned in the arrays \(irow\), \(icol\), \(ipiv\) and \(istr\) by a previous call to `nag_sparse_sym_chol_fac` (f11jac).

**b[n]**
Input: the right-hand side vector \(b\).

**tol**
Input: the required tolerance. Let \(x_k\) denote the approximate solution at iteration \(k\), and \(r_k\) the corresponding residual. The algorithm is considered to have converged at iteration \(k\) if:
\[
\|r_k\|_{\infty} \leq \tau \times (\|b\|_{\infty} + \|A\|_{\infty} \|x_k\|_{\infty}).
\]
If \(tol \leq 0.0\), \(\tau = \max(\sqrt{\epsilon}, \sqrt{n} \epsilon)\) is used, where \(\epsilon\) is the machine precision. Otherwise \(\tau = \max(tol, 10 \epsilon, \sqrt{n} \epsilon)\) is used.
Constraint: \(tol < 1.0\).

**maxitn**
Input: the maximum number of iterations allowed.
Constraint: \(maxitn \geq 1\).

**x[n]**
Input: an initial approximation to the solution vector \(x\).
Output: an improved approximation to the solution vector \(x\).

**rnorm**
Output: the final value of the residual norm \(\|r_k\|_{\infty}\), where \(k\) is the output value of \(itn\).

**itn**
Output: the number of iterations carried out.

**comm**
Input/Output: a pointer to a structure of type `Nag_Sparse_Comm` whose members are used by the iterative solver.

**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 `method` had an illegal value.

**NE_INT_ARG_LT**
On entry, \(n\) must not be less than 1: \(n = \langle value\rangle\).
On entry, `maxitn` must not be less than 1: \(maxitn = \langle value\rangle\).

**NE_INT_2**
On entry, \(nnz = \langle value\rangle\), \(n = \langle value\rangle\).
Constraint: \(1 \leq nnz \leq n \times (n+1)/2\).

**NE_REAL_ARG_GE**
On entry, `tol` must not be greater than or equal to 1.0: \(tol = \langle value\rangle\).
6. Further Comments
The time taken by \texttt{nag\_sparse\_sym\_chol\_sol} for each iteration is roughly proportional to the value of \texttt{nnz} returned from the preceding call to \texttt{nag\_sparse\_sym\_chol\_fac (f11jac)}. One iteration with the Lanczos method (SYMMLQ) requires a slightly larger number of operations than one iteration with the conjugate gradient method.

The number of iterations required to achieve a prescribed accuracy cannot be easily determined a priori, as it can depend dramatically on the conditioning and spectrum of the preconditioned matrix \( A = M^{-1}A \).

Some illustrations of the application of \texttt{nag\_sparse\_sym\_chol\_sol} to linear systems arising from the discretization of two-dimensional elliptic partial differential equations, and to random-valued randomly structured symmetric positive-definite linear systems, can be found in Salvini and Shaw (1995).

6.1. Accuracy
On successful termination, the final residual \( r_k = b - Ax_k \), where \( k = \text{itn} \), satisfies the termination criterion
\[
\|r_k\|_\infty \leq \tau \times (\|b\|_\infty + \|A\|_\infty \|x_k\|_\infty).
\]
The value of the final residual norm is returned in \texttt{rnorm}.

6.2. References


See Also

nag_sparse_sym_chol_fac (f11jac)
nag_sparse_sym_sol (f11jec)
nag_sparse_sym_sort (f11zbc)

Example

This example program solves a symmetric positive-definite system of equations using the conjugate gradient method, with incomplete Cholesky preconditioning.

Program Text

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

main()
{
    double dtol;
    double *a=0, *b=0;
    double *x=0;
    double rnorm, dscale;
    double tol;
    Integer *icol=0;
    Integer *ipiv=0, nnzc, *irow=0, *istr=0;
    Integer i;
    Integer n;
    Integer lfill, npivm;
    Integer maxitn;
    Integer itn;
    Integer nnz;
    Integer num;
    Nag_SparseSym_Method method;
    Nag_SparseSym_Piv pstrat;
    Nag_SparseSym_Fact mic;
    Nag_Sparse_Comm comm;
    char char_enum[20];

    Vprintf("f11jcc Example Program Results\n");
    printf("\%*\n\n");// Skip heading in data file */
    Vscanf(" %*[^\n]");
    /* Read algorithmic parameters */
    Vscanf("%ld%*[\n]&n");
    Vscanf("%ld%*[\n]&nnz");
    Vscanf("%ld%lf%*[\n]%lfill, &dtol");
    Vscanf("%s%*[\n]%s",char_enum);
    if (!strcmp(char_enum, "CG"))
        method = Nag_SparseSym_CG;
    else if (!strcmp(char_enum, "Lanczos"))
        method = Nag_SparseSym_Lanczos;
    else
    {
        Vprintf("Unrecognised string for method enum representation.\n");
        exit(EXIT_FAILURE);
    }
```

Vscanf("%s%lf*[\n]",char_enum, &dscale);
if (!strcmp(char_enum, "ModFact"))
    mic = Nag_SparseSym_ModFact;
else if (!strcmp(char_enum, "UnModFact"))
    mic = Nag_SparseSym_UnModFact;
else
{
    Vprintf("Unrecognised string for mic enum representation.\n");
    exit(EXIT_FAILURE);
}

Vscanf("%s*\[\n]",char_enum);
if (!strcmp(char_enum, "NoPiv"))
    pstrat = Nag_SparseSym_NoPiv;
else if (!strcmp(char_enum, "MarkPiv"))
    pstrat = Nag_SparseSym_MarkPiv;
else if (!strcmp(char_enum, "UserPiv"))
    pstrat = Nag_SparseSym_UserPiv;
else
{
    Vprintf("Unrecognised string for pstrat enum representation.\n");
    exit(EXIT_FAILURE);
}

Vscanf("%lf%ld*\[\n]",&tol, &maxitn);

/* Read the matrix a */
num = 2 * nnz;
irow = NAG_ALLOC(num,Integer);
icol = NAG_ALLOC(num,Integer);
a = NAG_ALLOC(num,double);
b = NAG_ALLOC(n,double);
x = NAG_ALLOC(n,double);
istr = NAG_ALLOC(n+1,Integer);
ipiv = NAG_ALLOC(num,Integer);

if (!irow || !icol || !a || !x || !istr ||!ipiv)
{
    Vprintf("Allocation failure\n");
    exit(EXIT_FAILURE);
}

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

/* Read right-hand side vector b and initial approximate solution x */
for (i = 1; i <= n; ++i)
    Vscanf("%lf",&b[i-1]);
Vscanf("%*[\n]");

for (i = 1; i <= n; ++i)
    Vscanf("%lf",&x[i-1]);
Vscanf("%*[\n]");

/* Calculate incomplete Cholesky factorization */
f11jac(n, nnz, &a, &num, &irow, &icol, lfill, dtol, mic, 
    dscale, pstrat, ipiv, istr, &nnzc, &npivm, &comm, NAGERR_DEFAULT);

/* Solve Ax = b */
f11jcc(method, n, nnz, &a, num, irow, icol, ipiv, istr, b, 
    tol, maxitn, x, &rnorm, &itn, &comm, NAGERR_DEFAULT);

Vprintf("%s%10ld%*\n","Converged in",itn," iterations");
Vprintf("%s%16.3e%*\n","Final residual norm =",rnorm);
nag_sparse_sym_chol_sol

NAG C Library Manual

/* Output x */
for (i = 1; i <= n; ++i)
    Vprintf(" %16.4e\n",x[i-1]);

NAG_FREE(irow);
NAG_FREE(icol);
NAG_FREE(a);
NAG_FREE(b);
NAG_FREE(x);
NAG_FREE(ipiv);
NAG_FREE(istr);
exit (EXIT_SUCCESS);
}

8.2. Program Data

f11jcc Example Program Data

<table>
<thead>
<tr>
<th></th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>16</td>
</tr>
<tr>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>lfill, dtol</td>
</tr>
<tr>
<td></td>
<td>method</td>
</tr>
<tr>
<td></td>
<td>UnModFact 0.0</td>
</tr>
<tr>
<td></td>
<td>mic, dscale</td>
</tr>
<tr>
<td></td>
<td>MarkPiv</td>
</tr>
<tr>
<td></td>
<td>pstrat</td>
</tr>
<tr>
<td></td>
<td>1.0e-6</td>
</tr>
<tr>
<td></td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>tol, maxitn</td>
</tr>
</tbody>
</table>

4. 1 1
1. 2 1
5. 2 2
2. 3 3
2. 4 2
3. 4 4
-1. 5 1
1. 5 4
4. 5 5
1. 6 2
-2. 6 5
3. 6 6
2. 7 1
-1. 7 2
-2. 7 3
5. 7 7

a[i-1], irow[i-1], icol[i-1], i=1,...,nnz

15. 18. -8. 21.
11. 10. 29. b[i-1], i=1,...,n
0. 0. 0. 0.
0. 0. 0. x[i-1], i=1,...,n

8.3. Program Results

f11jcc Example Program Results

Converged in 1 iterations
Final residual norm = 7.105e-15
1.0000e+00
2.0000e+00
3.0000e+00
4.0000e+00
5.0000e+00
6.0000e+00
7.0000e+00

3.f11jcc.6