1 Purpose

nag_sparse_herm_chol_sol (f11jqc) solves a complex sparse Hermitian system of linear equations, represented in symmetric coordinate storage format, using a conjugate gradient or Lanczos method, with incomplete Cholesky preconditioning.

2 Specification

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

void nag_sparse_herm_chol_sol (Nag_SparseSym_Method method, Integer n,
                          Integer nnz, const Complex a[], Integer la,
                          const Integer irow[], const Integer icol[],
                          const Integer ipiv[], const Integer istr[],
                          const Complex b[], double tol, Integer maxitn,
                          Complex x[],
                          double *rnorm, Integer *itn, NagError *fail)
```

3 Description

nag_sparse_herm_chol_sol (f11jqc) solves a complex sparse Hermitian linear system of equations

\[ Ax = b, \]

using a preconditioned conjugate gradient method (see Meijerink and Van der Vorst (1977)), or a preconditioned Lanczos method based on the algorithm SYMMLQ (see 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_herm_chol_sol (f11jqc) uses the incomplete Cholesky factorization determined by nag_sparse_herm_chol_fac (f11jnc) as the preconditioning matrix. A call to nag_sparse_herm_chol_sol (f11jqc) must always be preceded by a call to nag_sparse_herm_chol_fac (f11jnc). Alternative preconditioners for the same storage scheme are available by calling nag_sparse_herm_sol (f11jsc).

The matrix \( A \) and the preconditioning matrix \( M \) are represented in symmetric coordinate storage (SCS) format (see Section 2.1.2 in the f11 Chapter Introduction) in the arrays \( a \), \( irow \) and \( icol \), as returned from nag_sparse_herm_chol_fac (f11jnc). The array \( a \) holds the nonzero entries in the lower triangular parts of these matrices, while \( irow \) and \( icol \) hold the corresponding row and column indices.

4 References


5 Arguments

1: method – Nag_SparseSym_Method
   
   On entry: specifies the iterative method to be used.

   method = Nag_SparseSym.CG
   Conjugate gradient method.

   method = Nag_SparseSym_SYMLQ
   Lanczos method (SYMLQ).

   Constraint: method = Nag_SparseSym.CG or Nag_SparseSym_SYMLQ.

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

   Constraint: n ≥ 1.

3: nnz – Integer
   
   On entry: the number of nonzero 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_herm_chol_fac (f11jnc).

   Constraint: 1 ≤ nnz ≤ n × (n + 1)/2.

4: a[la] – const Complex
   
   On entry: the values returned in the array a by a previous call to nag_sparse_herm_chol_fac (f11jnc).

5: la – Integer
   
   On entry: the dimension of the arrays a, irow and icol. This must be the same value as was supplied in the preceding call to nag_sparse_herm_chol_fac (f11jnc).

   Constraint: la ≥ 2 × nnz.

6: irow[la] – const Integer

7: icol[la] – const Integer

8: ipiv[n] – const Integer

9: istr[n + 1] – const Integer
   
   On entry: the values returned in arrays irow, icol, ipiv and istr by a previous call to nag_sparse_herm_chol_fac (f11jnc).

10: b[n] – const Complex
   
   On entry: the right-hand side vector b.

11: tol – double
   
   On entry: 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 ≤ 0.0, \( \tau = \max(\sqrt{\epsilon}, 10\epsilon, \sqrt{n} \epsilon) \) is used, where \( \epsilon \) is the machine precision. Otherwise \( \tau = \max(\text{tol}, 10\epsilon, \sqrt{n} \epsilon) \) is used.

   Constraint: tol < 1.0.
f1l – Large Scale Linear Systems

f1ljqc

12: maxitn – Integer
   On entry: the maximum number of iterations allowed.
   Constraint: maxitn ≥ 1.

13: x[n] – Complex
   On entry: an initial approximation to the solution vector x.
   On exit: an improved approximation to the solution vector x.

14: rnorm – double *
   On exit: the final value of the residual norm \(\|r_k\|_\infty\), where \(k\) is the output value of itn.

15: itn – Integer *
   On exit: the number of iterations carried out.

16: fail – NagError *
   The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ACCURACY
   The required accuracy could not be obtained. However a reasonable accuracy has been achieved.

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

NE_BAD_PARAM
   On entry, argument \(\langle\text{value}\rangle\) had an illegal value.

NE_COEFF_NOT_POS_DEF
   The matrix of the coefficients a appears not to be positive definite. The computation cannot continue.

NE_CONVERGENCE
   The solution has not converged after \(\langle\text{value}\rangle\) iterations.

NE_INT
   On entry, maxitn = \(\langle\text{value}\rangle\).
   Constraint: maxitn ≥ 1.
   On entry, n = \(\langle\text{value}\rangle\).
   Constraint: n ≥ 1.
   On entry, nnz = \(\langle\text{value}\rangle\).
   Constraint: nnz ≥ 1.

NE_INT_2
   On entry, la = \(\langle\text{value}\rangle\) and nnz = \(\langle\text{value}\rangle\).
   Constraint: la ≥ 2 × nnz.
   On entry, nnz = \(\langle\text{value}\rangle\) and n = \(\langle\text{value}\rangle\).
   Constraint: nnz ≤ n × (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.

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

See Section 3.6.6 in the Essential Introduction for further information.

A serious error, code \(\text{value}\), has occurred in an internal call to nag_sparse_herm_basic_solver (f11gsc). Check all function calls and array sizes. Seek expert help.

A serious error, code \(\text{value}\), has occurred in an internal call to \(\text{value}\). Check all function calls and array sizes. Seek expert help.

NE_INVALID_SCS

On entry, \(i = \langle\text{value}\rangle, \text{icol}[i-1] = \langle\text{value}\rangle, \text{irow}[i-1] = \langle\text{value}\rangle\).
Constraint: \(\text{icol}[i-1] \geq 1\) and \(\text{icol}[i-1] \leq \text{irow}[i-1]\).
Check that \(a, \text{irow}, \text{icol}, \text{ipiv}\) and \(\text{istr}\) have not been corrupted between calls to nag_sparse_herm_chol_fac (f11jnc) and nag_sparse_herm_chol_sol (f11jqc).

On entry, \(i = \langle\text{value}\rangle, \text{irow}[i-1] = \langle\text{value}\rangle\) and \(n = \langle\text{value}\rangle\).
Constraint: \(\text{irow}[i-1] \geq 1\) and \(\text{irow}[i-1] \leq n\).
Check that \(a, \text{irow}, \text{icol}, \text{ipiv}\) and \(\text{istr}\) have not been corrupted between calls to nag_sparse_herm_chol_fac (f11jnc) and nag_sparse_herm_chol_sol (f11jqc).

The SCS representation of the preconditioner is invalid. Check that \(a, \text{irow}, \text{icol}, \text{ipiv}\) and \(\text{istr}\) have not been corrupted between calls to nag_sparse_herm_chol_fac (f11jnc) and nag_sparse_herm_chol_sol (f11jqc).

NE_NO_LICENCE

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.

NE_NOT STRICTLY_INCREASING

On entry, \(a[i-1]\) is out of order: \(i = \langle\text{value}\rangle\). Check that \(a, \text{irow}, \text{icol}, \text{ipiv}\) and \(\text{istr}\) have not been corrupted between calls to nag_sparse_herm_chol_fac (f11jnc) and nag_sparse_herm_chol_sol (f11jqc).

On entry, the location \(\text{irow}[i-1], \text{icol}[i-1]\) is a duplicate: \(i = \langle\text{value}\rangle\). Check that \(a, \text{irow}, \text{icol}, \text{ipiv}\) and \(\text{istr}\) have not been corrupted between calls to nag_sparse_herm_chol_fac (f11jnc) and nag_sparse_herm_chol_sol (f11jqc).

The preconditioner appears not to be positive definite. The computation cannot continue.

NE_REAL

On entry, \(tol = \langle\text{value}\rangle\).
Constraint: \(tol < 1.0\).

7 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 \(\text{rnorm}\).
8 Parallelism and Performance

nag_sparse_herm_chol_sol (f11jqc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_sparse_herm_chol_sol (f11jqc) 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 X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users’ Note for your implementation for any additional implementation-specific information.

9 Further Comments

The time taken by nag_sparse_herm_chol_sol (f11jqc) for each iteration is roughly proportional to the value of \( \text{nnzc} \) returned from the preceding call to nag_sparse_herm_chol_fac (f11jnc). 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 easily be determined \textit{a priori}, as it can depend dramatically on the conditioning and spectrum of the preconditioned matrix of the coefficients \( A = M^{-1}A \).

10 Example

This example solves a complex sparse Hermitian positive definite system of equations using the conjugate gradient method, with incomplete Cholesky preconditioning.

10.1 Program Text

/* nag_sparse_herm_chol_sol (f11jqc) Example Program. *
 * Copyright 2014 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, rnorm, tol;
    Integer i, itn, la, lfill, maxitn, n, nnz, nnzc, npivm;
    /* Arrays */
    char nag_enum_arg[40];
    Complex *a = 0, *b = 0, *x = 0;
    Integer *icol = 0, *ipiv = 0, *irow = 0, *istr = 0;
    /* NAG types */
    Nag_SparseSym_Method method;
    Nag_SparseSym_Piv pstrat;
    Nag_SparseSym_Fact mic;
    NagError fail;

    INIT_FAIL(fail);

    printf("nag_sparse_herm_chol_sol (f11jqc) Example Program Results\n\n");
    /* Skip heading in data file*/
#ifdef _WIN32
    scanf_s("%*[\n"]);
#else
    scanf("%*[\n"]);
#endif
/* Read algorithmic parameters*/
#ifdef _WIN32
    scanf_s("%"NAG_IFMT"%\n", &n);
#else
    scanf("%"NAG_IFMT"%\n", &n);
#endif
#ifdef _WIN32
    scanf_s("%"NAG_IFMT"%\n", &nnz);
#else
    scanf("%"NAG_IFMT"%\n", &nnz);
#endif
/* Allocate memory */
la = 3 * nnz;
if (! (a = NAG_ALLOC(la, Complex)) ||
! (b = NAG_ALLOC(n, Complex)) ||
! (x = 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;
}
#ifdef _WIN32
    scanf_s("%39s%\n", nag_enum_arg, _countof(nag_enum_arg));
#else
    scanf("%39s%\n", nag_enum_arg);
#endif
/* nag_enum_name_to_value (x04nac).
* Converts NAG enum member name to value */
    method = (Nag_SparseNsym_Method) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
    scanf_s("%lf%"NAG_IFMT"%\n", &lfill, &dtol);
#else
    scanf("%lf%"NAG_IFMT"%\n", &lfill, &dtol);
#endif
#ifdef _WIN32
    scanf_s("%39s%\n", nag_enum_arg, _countof(nag_enum_arg));
#else
    scanf("%39s%\n", nag_enum_arg);
#endif
    mic = (Nag_SparseSym_Fact) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
    scanf_s("%lf%\n", &dscale);
#else
    scanf("%lf%\n", &dscale);
#endif
#ifdef _WIN32
    scanf_s("%39s%\n", nag_enum_arg, _countof(nag_enum_arg));
#else
    scanf("%39s%\n", nag_enum_arg);
#endif
    pstrat = (Nag_SparseNsym_Piv) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
    scanf_s("%lf"NAG_IFMT"%\n", &tol, &maxitn);
#else
    scanf("%lf"NAG_IFMT"%\n", &tol, &maxitn);
#endif
/* Read the matrix a */
for (i = 0; i < nnz; i++)
#ifdef _WIN32
    scanf_s("( %lf , %lf )%"NAG_IFMT"%NAG_IFMT"%\n", 
#else
    scanf("( %lf , %lf )%"NAG_IFMT"%NAG_IFMT"%\n", 
#endif
/* Read rhs vector b and initial approximate solution x */
for (i = 0; i < n; i++)
#ifdef _WIN32
scanf_s(" ( %lf , %lf ) ", &b[i].re, &b[i].im);
#else
scanf(" ( %lf , %lf ) ", &b[i].re, &b[i].im);
#endif
#endif
#ifdef _WIN32
scanf_s("%*[
"");
#else
scanf("%*[
"");
#endif
for (i = 0; i < n; i++)
#ifdef _WIN32
scanf_s(" ( %lf , %lf ) ", &x[i].re, &x[i].im);
#else
scanf(" ( %lf , %lf ) ", &x[i].re, &x[i].im);
#endif

if (fail.code != NE_NOERROR)
{
    printf("Error from nag_sparse_herm_chol_fac (f11jnc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Solve Linear System */
/* nag_sparse_herm_chol_sol (f11jqc). */
/* Solution of complex sparse Hermitian linear system, conjugate 
    * gradient/Lanczos method, preconditioner computed by f11jnc 
*/
nag_sparse_herm_chol_sol(method, n, nnz, a, la, irow, icol, ipiv, istr, 
    b, tol, maxitn, x, &rnorm, &itn, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_sparse_herm_chol_sol (f11jqc)\n%s\n", fail.message);
    exit_status = 2;
    goto END;
}
printf("Converged in %10"NAG_IPMT" iterations \n", itn);
printf("Final residual norm = %10.3e\n", rnorm);
printf(" Converged Solution\n");
/* Output x*/
for (i = 0; i < n; i++)
    printf(" (%13.4e, %13.4e)\n", x[i].re, x[i].im);

END:
NAG_FREE(a);
NAG_FREE(b);
NAG_FREE(x);
NAG_FREE(icol);
NAG_FREE(ipiv);
NAG_FREE(irow);
NAG_FREE(istr);
return exit_status;
10.2 Program Data

nag_sparse_herm_chol_sol (f11jqc) Example Program Data

9 : n
23 : nnz
Nag_SparseSym_CG : method
0 0.0 : lfill, dtol
Nag_SparseSym_UnModFact : mic
0.0 : dscale
Nag_SparseSym_MarkPiv : pstrat
1.0e-6 100 : tol, maxitn

( 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.) : b[i], i=0,...,n-1
( 0., 0.)
( 0., 0.)
( 0., 0.)
( 0., 0.)
( 0., 0.)
( 0., 0.)
( 0., 0.) : x[i], i=0,...,n-1
10.3 Program Results

nag_sparse_herm_chol_sol (f11jqc) Example Program Results

Converged in 5 iterations
Final residual norm = 3.197e-14

Converged Solution
(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)