1 Purpose
nag_sparse_herm_sol (f11jsc) solves a complex sparse Hermitian system of linear equations, represented in symmetric coordinate storage format, using a conjugate gradient or Lanczos method, without preconditioning, with Jacobi or with SSOR preconditioning.

2 Specification

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

void nag_sparse_herm_sol (Nag_SparseSym_Method method,
                         Nag_SparseSym_PrecType precon, Integer n, Integer nnz,
                         const Complex a[], const Integer irow[], const Integer icol[],
                         double omega, const Complex b[], double tol, Integer maxitn,
                         Complex x[], double *rnorm, Integer *itn, double rdiag[],
                         NagError *fail)
```

3 Description
nag_sparse_herm_sol (f11jsc) solves a complex sparse Hermitian linear system of equations

\[ Ax = b, \]

using a preconditioned conjugate gradient method (see Barrett et al. (1994)), 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_sol (f11jsc) allows the following choices for the preconditioner:

- no preconditioning;
- Jacobi preconditioning (see Young (1971));
- symmetric successive-over-relaxation (SSOR) preconditioning (see Young (1971)).

For incomplete Cholesky (IC) preconditioning see nag_sparse_herm_chol_sol (f11jqc).

The matrix \( A \) is represented in symmetric coordinate storage (SCS) format (see Section 2.1.2 in the f11 Chapter Introduction) in the arrays \( a, irow \) and \( icol \). The array \( a \) holds the nonzero entries in the lower triangular part of the matrix, while \( irow \) and \( icol \) hold the corresponding row and column indices.

4 References
5 Arguments

1:  **method**  – Nag_SparseSym_Method  \(\text{Input}\)

   *On entry:* specifies the iterative method to be used.

   - **method** = Nag_SparseSym.CG  
     Conjugate gradient method.
   - **method** = Nag_SparseSym_SYMMLQ  
     Lanczos method (SYMMLQ).

   *Constraint:* **method** = Nag_SparseSym.CG or Nag_SparseSym_SYMMLQ.

2:  **precon**  – Nag_SparseSym_PrecType  \(\text{Input}\)

   *On entry:* specifies the type of preconditioning to be used.

   - **precon** = Nag_SparseSym.NoPrec  
     No preconditioning.
   - **precon** = Nag_SparseSym.JacPrec  
     Jacobi.
   - **precon** = Nag_SparseSym.SSORPrec  
     Symmetric successive-over-relaxation (SSOR).

   *Constraint:* **precon** = Nag_SparseSym.NoPrec, Nag_SparseSym.JacPrec or Nag_SparseSym.SSORPrec.

3:  **n**  – Integer  \(\text{Input}\)

   *On entry:* \(n\), the order of the matrix \(A\).

   *Constraint:* \(n \geq 1\).

4:  **nnz**  – Integer  \(\text{Input}\)

   *On entry:* the number of nonzero elements in the lower triangular part of the matrix \(A\).

   *Constraint:* \(1 \leq \text{nnz} \leq n \times (n + 1)/2\).

5:  **a[nnz]**  – const Complex  \(\text{Input}\)

   *On entry:* the nonzero elements of the lower triangular part of the matrix \(A\), ordered by increasing row index, and by increasing column index within each row. Multiple entries for the same row and column indices are not permitted. The function nag_sparse_herm_sort (f11zpc) may be used to order the elements in this way.

6:  **irow[nnz]**  – const Integer  \(\text{Input}\)

7:  **icol[nnz]**  – const Integer  \(\text{Input}\)

   *On entry:* the row and column indices of the nonzero elements supplied in array **a**.

   *Constraints:*  
   - **irow** and **icol** must satisfy these constraints (which may be imposed by a call to nag_sparse_herm_sort (f11zpc)):  
     \(1 \leq \text{irow}[i] \leq \text{n}\) and \(1 \leq \text{icol}[i] \leq \text{irow}[i]\), for \(i = 0, 1, \ldots, \text{nnz} - 1\);  
     \(\text{irow}[i - 1] < \text{irow}[i]\) or \(\text{irow}[i - 1] = \text{irow}[i]\) and \(\text{icol}[i - 1] < \text{icol}[i]\), for \(i = 1, 2, \ldots, \text{nnz} - 1\).
8: \( \text{omega} \) – double 
\( \text{Input} \)

\( \text{On entry: if precon} = \text{Nag\_SparseSym\_SSORPrec, omega} \) is the relaxation parameter \( \omega \) to be used in the SSOR method. Otherwise \( \text{omega} \) need not be initialized.

\( \text{Constraint: } 0.0 < \text{omega} < 2.0. \)

9: \( b[n] \) – const Complex 
\( \text{Input} \)

\( \text{On entry: the right-hand side vector} \ b. \)

10: \( \text{tol} \) – double 
\( \text{Input} \)

\( \text{On entry: the required tolerance. Let} \ x_k \ \text{denote the approximate solution at iteration} \ k, \ \text{and} \ r_k \ \text{the corresponding residual. The algorithm is considered to have converged at iteration} \ k \ \text{if} \)

\( \|r_k\|_\infty \leq \tau \times (\|b\|_\infty + \|A\|_\infty \|x_k\|_\infty). \)

\( \text{If} \ \text{tol} \leq 0.0, \ \tau = \max(\sqrt{\epsilon}, 10\epsilon, \sqrt{n}\epsilon) \) \text{is used, where} \ \epsilon \ \text{is the machine precision. Otherwise} \)

\( \tau = \max(\text{tol}, 10\epsilon, \sqrt{n}\epsilon) \) \text{is used.}

\( \text{Constraint:} \ \text{tol} < 1.0. \)

11: \( \text{maxitn} \) – Integer 
\( \text{Input} \)

\( \text{On entry: the maximum number of iterations allowed.} \)

\( \text{Constraint:} \ \text{maxitn} \geq 1. \)

12: \( x[n] \) – Complex 
\( \text{Input/Output} \)

\( \text{On entry: an initial approximation to the solution vector} \ x. \)

\( \text{On exit: an improved approximation to the solution vector} \ x. \)

13: \( \text{rnorm} \) – double * 
\( \text{Output} \)

\( \text{On exit: the final value of the residual norm} \ \|r_k\|, \text{where} \ k \ \text{is the output value of} \ \text{itn}. \)

14: \( \text{itn} \) – Integer * 
\( \text{Output} \)

\( \text{On exit: the number of iterations carried out.} \)

15: \( \text{rdiag[n]} \) – double 
\( \text{Output} \)

\( \text{On exit: the elements of the diagonal matrix} \ D^{-1}, \text{where} \ D \ \text{is the diagonal part of} \ A. \text{Note that since} \ A \ \text{is Hermitian the elements of} \ D^{-1} \ \text{are necessarily real.} \)

16: \( \text{fail} \) – NagError * 
\( \text{Input/Output} \)

\( \text{The NAG error argument (see Section 3.6 in the Essential Introduction).} \)

6 \ Error Indicators and Warnings

\textbf{NE\_ACCURACY}

The required accuracy could not be obtained. However, a reasonable accuracy has been achieved and further iterations could not improve the result.

\textbf{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 ⟨value⟩ 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 ⟨value⟩ iterations.

NE_INT
On entry, maxitn = ⟨value⟩.
Constraint: maxitn ≥ 1.

On entry, n = ⟨value⟩.
Constraint: n ≥ 1.

On entry, nnz = ⟨value⟩.
Constraint: nnz ≥ 1.

NE_INT_2
On entry, nnz = ⟨value⟩ and n = ⟨value⟩.
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 ⟨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 ⟨value⟩, has occurred in an internal call to ⟨value⟩. Check all function calls and array sizes. Seek expert help.

NE_INVALID_SCS
On entry, I = ⟨value⟩, icol[I - 1] = ⟨value⟩ and irow[I - 1] = ⟨value⟩.
Constraint: icol[I - 1] ≥ 1 and icol[I - 1] ≤ irow[I - 1].

On entry, i = ⟨value⟩, irow[i - 1] = ⟨value⟩ and n = ⟨value⟩.
Constraint: irow[i - 1] ≥ 1 and irow[i - 1] ≤ n.

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 = ⟨value⟩.

On entry, the location (irow[I - 1], icol[I - 1]) is a duplicate: I = ⟨value⟩. Consider calling nag_sparse_herm_sort (f11zpc) to reorder and sum or remove duplicates.

NE_PRECOND_NOT_POS_DEF
The preconditioner appears not to be positive definite. The computation cannot continue.
NE_REAL
On entry, \(\omega = \langle \text{value} \rangle\).
Constraint: \(0.0 < \omega < 2.0\).
On entry, \(\text{tol} = \langle \text{value} \rangle\).
Constraint: \(\text{tol} < 1.0\).

NE_ZERO_DIAG_ELEM
The matrix \(A\) has a non-real diagonal entry in row \(\langle \text{value} \rangle\).
The matrix \(A\) has a zero diagonal entry in row \(\langle \text{value} \rangle\).
The matrix \(A\) has no diagonal entry in row \(\langle \text{value} \rangle\).

7 Accuracy
On successful termination, the final residual \(r_k = b - A x_k\), 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
\text{nag_sparse_herm_sol (f11jsc)} is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.
\text{nag_sparse_herm_sol (f11jsc)} 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 \text{nag_sparse_herm_sol (f11jsc)} for each iteration is roughly proportional to \(\text{nnz}\). 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 SSOR preconditioning.

10.1 Program Text
/* \text{nag_sparse_herm_sol (f11jsc)} Example Program. *
* Copyright 2014 Numerical Algorithms Group. *
* Mark 23, 2011. */
#include <nag.h>
#include <nag_stdbib.h>
```c
#include <naga02.h>
#include <nagf11.h>

int main(void)
{
    /* Scalars */
    Integer exit_status = 0;
    double omega, rnorm, tol;
    Integer i, itn, maxitn, n, nnz;
    /* Arrays */
    Complex *a = 0, *b = 0, *x = 0;
    double *rdiag = 0;
    Integer *icol = 0, *irow = 0;
    char nag_enum_arg[40];
    /* NAG types */
    Nag_SparseSym_Method method;
    Nag_SparseSym_PrecType precon;
    NagError fail;

    INIT_FAIL(fail);

    printf("nag_sparse_herm_sol (f11jsc) Example Program Results\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("%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_SparseSym_Method) nag_enum_name_to_value(nag_enum_arg);
    #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 */
    precon = (Nag_SparseSym_PrecType) nag_enum_name_to_value(nag_enum_arg);
    #ifdef _WIN32
    scanf_s("%lf%*[\n]", &omega);
    #else
    scanf("%lf%*[\n]", &omega);
    #endif
    #ifdef _WIN32
    scanf_s("%lf"NAG_IFMT"%*[\n]", &tol, &maxitn);
    #else
    scanf("%lf"NAG_IFMT"%*[\n]", &tol, &maxitn);
    #endif

    /* Allocate memory */
    if (!a || (a = NAG_ALLOC((nnz), Complex)) || !b || (b = NAG_ALLOC((n), Complex)) || !x || (x = NAG_ALLOC((n), Complex)) || !rdiag || (rdiag = NAG_ALLOC((n), double)) ||
```
! (icol = NAG_ALLOC((nnz), Integer)) ||
! (irow = NAG_ALLOC((nnz), Integer))
}
{
printf("Allocation failure\n");
exit_status = -1;
goto END;
}
/* Read the matrix a */
for (i = 0; i < nnz; i++)
#ifdef _WIN32
    scanf_s(" ( %lf , %lf ) %"NAG_IFMT"%*[\n] ", &a[i].re, &a[i].im, &irow[i], &icol[i]);
#else
    scanf(" ( %lf , %lf ) %"NAG_IFMT"%*[\n] ", &a[i].re, &a[i].im, &irow[i], &icol[i]);
#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
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
#ifdef _WIN32
    scanf_s(" ( %lf , %lf ) ", &x[i].re, &x[i].im);
#else
    scanf(" ( %lf , %lf ) ", &x[i].re, &x[i].im);
#endif
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif
/* nag_sparse_herm_sol (f11jsc). 
* Solution of complex sparse Hermitian linear system, conjugate
* gradient/Lanczos method, Jacobi or SSOR preconditioner 
*/
nag_sparse_herm_sol(method, precon, n, nnz, a, irow, icol, omega,
b, tol, maxitn, x, &rnorm, &itn, rdiag, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_sparse_herm_sol (f11jsc)\n", fail.message);
    exit_status = 1;
goto END;
}
printf("Converged in %10"NAG_IFMT" iterations \n", itn);
printf("Final residual norm = %10.3e\n", rnorm);
/* Output x*/
printf("\nConverged Solution\n");
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(rdiag);
NAG_FREE(rdiag);
NAG_FREE(icol);
NAG_FREE(icolo);
return exit_status;
10.2 Program Data

nag_sparse_herm_sol (f11jsc) Example Program Data

9

23

Nag_SparseSym_CG : method
Nag_SparseSym_SSORPrec : precon

1.1

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.)
( 0., 0.)

10.3 Program Results

nag_sparse_herm_sol (f11jsc) Example Program Results

Converged in 7 iterations

Final residual norm = 1.477e-05

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)