NAG Library Function Document

nag_sparse_herm_sol (f11jsc)

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

```
#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

Barrett R, Berry M, Chan T F, Demmel J, Donato J, Dongarra J, Eijkhout V, Pozo R, Romine C and Van der Vorst H (1994) *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods* SIAM, Philadelphia

Paige C C and Saunders M A (1975) Solution of sparse indefinite systems of linear equations *SIAM J. Numer. Anal.* **12** 617–629

Young D (1971) *Iterative Solution of Large Linear Systems* Academic Press, New York

5 Arguments

1: **method** – Nag_SparseSym_Method *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 *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 *Input*

On entry: n, the order of the matrix A.

Constraint: $\mathbf{n} \geq 1$.

4: **nnz** – Integer *Input*

On entry: the number of nonzero elements in the lower triangular part of the matrix A.

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

5: **a**[**nnz**] – const Complex *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 *Input*
7: **icol**[**nnz**] – const Integer *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)):

$1 \leq \mathbf{irow}[i] \leq \mathbf{n}$ and $1 \leq \mathbf{icol}[i] \leq \mathbf{irow}[i]$, for $i = 0, 1, \ldots, \mathbf{nnz} - 1$;
$\mathbf{irow}[i-1] < \mathbf{irow}[i]$ or $\mathbf{irow}[i-1] = \mathbf{irow}[i]$ and $\mathbf{icol}[i-1] < \mathbf{icol}[i]$, for $i = 1, 2, \ldots, \mathbf{nnz} - 1$.

8: **omega** – double *Input*

On entry: if **precon** $=$ Nag_SparseSym_SSORPrec, **omega** is the relaxation argument ω to be used in the SSOR method. Otherwise **omega** need not be initialized.

Constraint: $0.0 <$ **omega** < 2.0.

9: **b**[**n**] – const Complex *Input*

On entry: the right-hand side vector b.

10: **tol** – double *Input*

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

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

If **tol** ≤ 0.0, $\tau = \max(\sqrt{\epsilon}, 10\epsilon, \sqrt{n}\epsilon)$ is used, where ϵ is the ***machine precision***. Otherwise $\tau = \max(\mathbf{tol}, 10\epsilon, \sqrt{n}\epsilon)$ is used.

Constraint: **tol** < 1.0.

11: **maxitn** – Integer *Input*

On entry: the maximum number of iterations allowed.

Constraint: **maxitn** ≥ 1.

12: **x**[**n**] – Complex *Input/Output*

On entry: an initial approximation to the solution vector x.

On exit: an improved approximation to the solution vector x.

13: **rnorm** – double * *Output*

On exit: the final value of the residual norm $\|r_k\|$, where k is the output value of **itn**.

14: **itn** – Integer * *Output*

On exit: the number of iterations carried out.

15: **rdiag**[**n**] – double *Output*

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

16: **fail** – NagError * *Input/Output*

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 and further iterations could not improve the result.

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BAD_PARAM

On entry, argument $\langle$*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 $\langle value \rangle$ iterations.

NE_INT

On entry, **maxitn** $= \langle value \rangle$.
Constraint: **maxitn** ≥ 1.

On entry, **n** $= \langle value \rangle$.
Constraint: **n** ≥ 1.

On entry, **nnz** $= \langle value \rangle$.
Constraint: **nnz** ≥ 1.

NE_INT_2

On entry, **nnz** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: **nnz** $\leq$ **n** $\times$ (**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.

A serious error, code $\langle value \rangle$, 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 $\langle value \rangle$, has occurred in an internal call to $\langle value \rangle$. Check all function calls and array sizes. Seek expert help.

NE_INVALID_SCS

On entry, $I = \langle value \rangle$, **icol**$[I-1] = \langle value \rangle$ and **irow**$[I-1] = \langle value \rangle$.
Constraint: **icol**$[I-1] \geq 1$ and **icol**$[I-1] \leq$ **irow**$[I-1]$.

On entry, $i = \langle value \rangle$, **irow**$[i-1] = \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: **irow**$[i-1] \geq 1$ and **irow**$[i-1] \leq$ **n**.

NE_NOT_STRICTLY_INCREASING

On entry, **a**$[i-1]$ is out of order: $i = \langle value \rangle$.

On entry, the location (**irow**$[I-1]$, **icol**$[I-1]$) is a duplicate: $I = \langle value \rangle$. 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** $= \langle value \rangle$.
Constraint: $0.0 <$ **omega** < 2.0.

On entry, **tol** $= \langle value \rangle$.
Constraint: **tol** < 1.0.

NE_ZERO_DIAG_ELEM

The matrix A has a non-real diagonal entry in row $\langle value \rangle$.

The matrix A has a zero diagonal entry in row $\langle value \rangle$.

The matrix A has no diagonal entry in row $\langle value \rangle$.

7 Accuracy

On successful termination, the final residual $r_k = b - Ax_k$, where $k = \textbf{itn}$, satisfies the termination criterion

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

The value of the final residual norm is returned in **rnorm**.

8 Parallelism and Performance

nag_sparse_herm_sol (f11jsc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

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 Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

The time taken by nag_sparse_herm_sol (f11jsc) for each iteration is roughly proportional to **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 *a priori*, as it can depend dramatically on the conditioning and spectrum of the preconditioned matrix of the coefficients $\bar{A} = M^{-1}A$.

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

```
/* nag_sparse_herm_sol (f11jsc) Example Program.
 *
 * Copyright 2011, 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                 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*/
  scanf("%*[^\n]");
  /* Read algorithmic parameters*/
  scanf("%ld%*[^\n]", &n);
  scanf("%ld%*[^\n]", &nnz);
  scanf("%39s%*[^\n]", nag_enum_arg);
  /* 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);
  scanf("%39s%*[^\n]", nag_enum_arg);
  precon = (Nag_SparseSym_PrecType) nag_enum_name_to_value(nag_enum_arg);
  scanf("%lf%*[^\n]", &omega);
  scanf("%lf%ld%*[^\n]", &tol, &maxitn);

  /* Allocate memory */
  if (
      !(a = NAG_ALLOC((nnz), Complex)) ||
      !(b = NAG_ALLOC((n), Complex)) ||
      !(x = NAG_ALLOC((n), Complex)) ||
      !(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++)
    scanf(" ( %lf , %lf ) %ld%ld%*[^\n] ",
          &a[i].re, &a[i].im, &irow[i], &icol[i]);
  /* Read rhs vector b and initial approximate solution x*/
  for (i = 0; i < n; i++)
    scanf(" ( %lf , %lf ) ", &b[i].re, &b[i].im);
  scanf("%*[^\n]");
  for (i = 0; i < n; i++)
    scanf(" ( %lf , %lf ) ", &x[i].re, &x[i].im);
  scanf("%*[^\n]");

  /* 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%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }
  printf("Converged in  %10ld  iterations \n", itn);
  printf("Final residual norm = %10.3e\n", rnorm);

  /* Output x*/
  printf("\n     Converged 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(icol);
  NAG_FREE(irow);
  return exit_status;
}
```

10.2 Program Data

```
nag_sparse_herm_sol (f11jsc) Example Program Data
  9                       : n
  23                      : nnz
 Nag_SparseSym_CG         : method
 Nag_SparseSym_SSORPrec   : precon
  1.1                     : omega
  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.)
 (  0.,   0.)             : x[i], i = 0,...,n-1
```

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)
```