NAG Library Function Document

nag_sparse_nherm_sol (f11dsc)

1 Purpose

nag_sparse_nherm_sol (f11dsc) solves a complex sparse non-Hermitian system of linear equations, represented in coordinate storage format, using a restarted generalized minimal residual (RGMRES), conjugate gradient squared (CGS), stabilized bi-conjugate gradient (Bi-CGSTAB), or transpose-free quasi-minimal residual (TFQMR) method, without preconditioning, with Jacobi, or with SSOR preconditioning.

2 Specification

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

```
void nag_sparse_nherm_sol (Nag_SparseNsym_Method method,
     Nag_SparseNsym_PrecType precon, Integer n, Integer nnz,
     const Complex a[], const Integer irow[], const Integer icol[],
     double omega, const Complex b[], Integer m, double tol, Integer maxitn,
     Complex x[], double *rnorm, Integer *itn, NagError *fail)
```

3 Description

nag_sparse_nherm_sol (f11dsc) solves a complex sparse non-Hermitian system of linear equations:

$$Ax = b,$$

using an RGMRES (see Saad and Schultz (1986)), CGS (see Sonneveld (1989)), Bi-CGSTAB(ℓ) (see Van der Vorst (1989) and Sleijpen and Fokkema (1993)), or TFQMR (see Freund and Nachtigal (1991) and Freund (1993)) method.

nag_sparse_nherm_sol (f11dsc) 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 *LU* (ILU) preconditioning see nag_sparse_nherm_fac_sol (f11dqc).

The matrix A is represented in coordinate storage (CS) format (see Section 2.1.1 in the f11 Chapter Introduction) in the arrays **a**, **irow** and **icol**. The array **a** holds the nonzero entries in the matrix, while **irow** and **icol** hold the corresponding row and column indices.

nag_sparse_nherm_sol (f11dsc) is a Black Box function which calls nag_sparse_nherm_basic_setup (f11brc), nag_sparse_nherm_basic_solver (f11bsc) and nag_sparse_nherm_basic_diagnostic (f11btc). If you wish to use an alternative storage scheme, preconditioner, or termination criterion, or require additional diagnostic information, you should call these underlying functions directly.

4 References

Freund R W (1993) A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems *SIAM J. Sci. Comput.* **14** 470–482

Freund R W and Nachtigal N (1991) QMR: a Quasi-Minimal Residual Method for Non-Hermitian Linear Systems *Numer. Math.* **60** 315–339

Saad Y and Schultz M (1986) GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems *SIAM J. Sci. Statist. Comput.* **7** 856–869

Sleijpen G L G and Fokkema D R (1993) BiCGSTAB(ℓ) for linear equations involving matrices with complex spectrum *ETNA* **1** 11–32

Sonneveld P (1989) CGS, a fast Lanczos-type solver for nonsymmetric linear systems *SIAM J. Sci. Statist. Comput.* **10** 36–52

Van der Vorst H (1989) Bi-CGSTAB, a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems *SIAM J. Sci. Statist. Comput.* **13** 631–644

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

5 Arguments

1: **method** – Nag_SparseNsym_Method *Input*

On entry: specifies the iterative method to be used.

method = Nag_SparseNsym_RGMRES
Restarted generalized minimum residual method.

method = Nag_SparseNsym_CGS
Conjugate gradient squared method.

method = Nag_SparseNsym_BiCGSTAB
Bi-conjugate gradient stabilized (ℓ) method.

method = Nag_SparseNsym_TFQMR
Transpose-free quasi-minimal residual method.

Constraint: **method** = Nag_SparseNsym_RGMRES, Nag_SparseNsym_CGS, Nag_SparseNsym_BiCGSTAB or Nag_SparseNsym_TFQMR.

2: **precon** – Nag_SparseNsym_PrecType *Input*

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

precon = Nag_SparseNsym_NoPrec
No preconditioning.

precon = Nag_SparseNsym_JacPrec
Jacobi.

precon = Nag_SparseNsym_SSORPrec
Symmetric successive-over-relaxation (SSOR).

Constraint: **precon** = Nag_SparseNsym_NoPrec, Nag_SparseNsym_JacPrec or Nag_SparseNsym_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 matrix A.

Constraint: $1 \leq \mathbf{nnz} \leq \mathbf{n}^2$.

5: **a**[**nnz**] – const Complex *Input*

On entry: the nonzero elements 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_nherm_sort (f11znc) 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 **a**.

Constraints:

irow and **icol** must satisfy the following constraints (which may be imposed by a call to nag_sparse_nherm_sort (f11znc)):

$1 \leq$ **irow**$[i] \leq$ **n** and $1 \leq$ **icol**$[i] \leq$ **n**, for $i = 0, 1, \ldots,$ **nnz** $- 1$;
either **irow**$[i-1] <$ **irow**$[i]$ or both **irow**$[i-1] =$ **irow**$[i]$ and **icol**$[i-1] <$ **icol**$[i]$, for $i = 1, 2, \ldots,$ **nnz** $- 1$.

8: **omega** – double *Input*

On entry: if **precon** = Nag_SparseNsym_SSORPrec, **omega** is the relaxation parameter ω to be used in the SSOR method. Otherwise **omega** need not be initialized and is not referenced.

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

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

On entry: the right-hand side vector b.

10: **m** – Integer *Input*

On entry: if **method** = Nag_SparseNsym_RGMRES, **m** is the dimension of the restart subspace.

If **method** = Nag_SparseNsym_BiCGSTAB, **m** is the order ℓ of the polynomial Bi-CGSTAB method.

Otherwise, **m** is not referenced.

Constraints:

if **method** = Nag_SparseNsym_RGMRES, $0 <$ **m** $\leq \min($**n**$, 50)$;
if **method** = Nag_SparseNsym_BiCGSTAB, $0 <$ **m** $\leq \min($**n**$, 10)$.

11: **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 $\tau = \max($**tol**$, 10\epsilon, \sqrt{n}\epsilon)$ is used.

Constraint: **tol** < 1.0.

12: **maxitn** – Integer *Input*

On entry: the maximum number of iterations allowed.

Constraint: **maxitn** ≥ 1.

13: **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.

14: **rnorm** – double * *Output*

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

15: **itn** – Integer * *Output*

On exit: the number of iterations carried out.

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

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ACCURACY

The required accuracy could not be obtained. However, a reasonable accuracy may have been achieved.

NE_ALG_FAIL

Algorithmic breakdown. A solution is returned, although it is possible that it is completely inaccurate.

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 3.2.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_BAD_PARAM

On entry, argument $\langle value\rangle$ had an illegal value.

NE_CONVERGENCE

The solution has not converged after $\langle value\rangle$ iterations.

NE_ENUM_INT_2

On entry, $\mathbf{m} = \langle value\rangle$ and $\mathbf{n} = \langle value\rangle$.
Constraint: $0 < \mathbf{m} \leq \min(\mathbf{n}, \langle value\rangle)$.

On entry, **method** $= \langle value\rangle$, $\mathbf{n} = \langle value\rangle$ and $\mathbf{m} = \langle value\rangle$.
Constraint: if **method** = Nag_SparseNsym_BiCGSTAB, $0 < \mathbf{m} \leq \min(\mathbf{n}, 10)$.

On entry, **method** $= \langle value\rangle$, $\mathbf{n} = \langle value\rangle$ and $\mathbf{m} = \langle value\rangle$.
Constraint: if **method** = Nag_SparseNsym_RGMRES, $0 < \mathbf{m} \leq \min(\mathbf{n}, 50)$.

NE_INT

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

On entry, $\mathbf{n} = \langle value\rangle$.
Constraint: $\mathbf{n} \geq 1$.

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

NE_INT_2

On entry, **nnz** $= \langle value\rangle$ and $\mathbf{n} = \langle value\rangle$.
Constraint: $1 \leq \mathbf{nnz} \leq \mathbf{n}^2$.

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 How to Use the NAG Library and its Documentation for further information.

NE_INVALID_CS

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

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_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in How to Use the NAG Library and its Documentation for further information.

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: $I = \langle value \rangle$.

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 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 $k =$ **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_nherm_sol (f11dsc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_sparse_nherm_sol (f11dsc) 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' Notefor your implementation for any additional implementation-specific information.

9 Further Comments

The time taken by nag_sparse_nherm_sol (f11dsc) for each iteration is roughly proportional to **nnz**.

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 coefficient matrix $\bar{A} = M^{-1}A$, for some preconditioning matrix M.

10 Example

This example solves a complex sparse non-Hermitian system of equations using the CGS method, with no preconditioning.

10.1 Program Text

```
/* nag_sparse_nherm_sol (f11dsc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */
#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, m, maxitn, n, nnz;
  /* Arrays */
  Complex *a = 0, *b = 0, *x = 0;
  Integer *icol = 0, *irow = 0;
  char nag_enum_arg[40];
  /* NAG types */
  Nag_SparseNsym_Method method;
  Nag_SparseNsym_PrecType precon;
  NagError fail;

  INIT_FAIL(fail);

  printf("nag_sparse_nherm_sol (f11dsc) Example Program Results\n\n");
  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
#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
#ifdef _WIN32
  scanf_s("%39s%*[^\n]", nag_enum_arg, (unsigned)_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("%39s%*[^\n]", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
  scanf("%39s%*[^\n]", nag_enum_arg);
#endif
  precon = (Nag_SparseNsym_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("%" NAG_IFMT "%lf%" NAG_IFMT "%*[^\n]", &m, &tol, &maxitn);
#else
  scanf("%" NAG_IFMT "%lf%" NAG_IFMT "%*[^\n]", &m, &tol, &maxitn);
#endif
  if (!(a = NAG_ALLOC((nnz), Complex)) ||
      !(b = NAG_ALLOC((n), Complex)) ||
      !(x = NAG_ALLOC((n), Complex)) ||
      !(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 "%" NAG_IFMT "%*[^\n] ",
            &a[i].re, &a[i].im, &irow[i], &icol[i]);
#else
    scanf(" ( %lf , %lf ) %" NAG_IFMT "%" NAG_IFMT "%*[^\n] ",
          &a[i].re, &a[i].im, &irow[i], &icol[i]);
#endif
  /* Read rhs vector b and initial approximate solution x */
#ifdef _WIN32
  for (i = 0; i < n; i++)
    scanf_s(" ( %lf , %lf ) ", &b[i].re, &b[i].im);
#else
  for (i = 0; i < n; i++)
    scanf(" ( %lf , %lf ) ", &b[i].re, &b[i].im);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
#ifdef _WIN32
  for (i = 0; i < n; i++)
    scanf_s(" ( %lf , %lf ) ", &x[i].re, &x[i].im);
#else
  for (i = 0; i < n; i++)
    scanf(" ( %lf , %lf ) ", &x[i].re, &x[i].im);
#endif

  /* solve ax = b */
  /* nag_sparse_nherm_sol (f11dsc).
   * Solution of complex sparse non-Hermitian linear system, RGMRES, CGS,
   * Bi-CGSTAB or TFQMR method, Jacobi or SSOR preconditioner Black Box.
   */
  nag_sparse_nherm_sol(method, precon, n, nnz, a, irow, icol, omega, b, m,
                       tol, maxitn, x, &rnorm, &itn, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_sparse_nherm_sol (f11dsc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }
```

```
  printf("Converged in%13" NAG_IFMT " iterations\n", itn);
  printf("Final residual norm = %11.3e\n\n", rnorm);
  /* Output x */
  printf("%14s\n", "Solution");
  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(irow);
  return exit_status;
}
```

10.2 Program Data

```
nag_sparse_nherm_sol (f11dsc) Example Program Data
  5                        : n
 16                        : nnz
  Nag_SparseNsym_CGS       : method
  Nag_SparseNsym_NoPrec    : precon
  1.05                     : omega
  1        1.e-10 1000     : m, tol, maxitn
 (  2.,  3.)    1    1
 (  1., -1.)    1    2
 ( -1.,  0.)    1    4
 (  0.,  2.)    2    2
 ( -2.,  1.)    2    3
 (  1.,  0.)    2    5
 (  0., -1.)    3    1
 (  5.,  4.)    3    3
 (  3., -1.)    3    4
 (  1.,  0.)    3    5
 ( -2.,  2.)    4    1
 ( -3.,  1.)    4    4
 (  0.,  3.)    4    5
 (  4., -2.)    5    2
 ( -2.,  0.)    5    3
 ( -6.,  1.)    5    5   : a[i], irow[i], icol[i], i=0,...,nnz-1
 ( -3.,  3.)
 (-11.,  5.)
 ( 23., 48.)
 (-41.,  2.)
 (-28.,-31.)               : b[i], i=0,...,n-1
 (  0.,  0.)
 (  0.,  0.)
 (  0.,  0.)
 (  0.,  0.)
 (  0.,  0.)               : x[i], i=0,...,n-1
```

10.3 Program Results

```
nag_sparse_nherm_sol (f11dsc) Example Program Results

Converged in            5 iterations
Final residual norm =   1.052e-10

      Solution
(   1.0000e+00,    2.0000e+00)
(   2.0000e+00,    3.0000e+00)
(   3.0000e+00,    4.0000e+00)
(   4.0000e+00,    5.0000e+00)
(   5.0000e+00,    6.0000e+00)
```