NAG Library Function Document

nag_sparse_nherm_fac_sol (f11dqc)

1 Purpose

nag_sparse_nherm_fac_sol (f11dqc) 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, with incomplete LU preconditioning.

2 Specification

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

```
void nag_sparse_nherm_fac_sol (Nag_SparseNsym_Method method, Integer n,
     Integer nnz, const Complex a[], Integer la, const Integer irow[],
     const Integer icol[], const Integer ipivp[], const Integer ipivq[],
     const Integer istr[], const Integer idiag[], const Complex b[],
     Integer m, double tol, Integer maxitn, Complex x[], double *rnorm,
     Integer *itn, NagError *fail)
```

3 Description

nag_sparse_nherm_fac_sol (f11dqc) solves a complex sparse non-Hermitian linear system of equations

$$Ax = b,$$

using a preconditioned 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_fac_sol (f11dqc) uses the incomplete LU factorization determined by nag_sparse_nherm_fac (f11dnc) as the preconditioning matrix. A call to nag_sparse_nherm_fac_sol (f11dqc) must always be preceded by a call to nag_sparse_nherm_fac (f11dnc). Alternative preconditioners for the same storage scheme are available by calling nag_sparse_nherm_sol (f11dsc).

The matrix A, and the preconditioning matrix M, are represented in coordinate storage (CS) format (see Section 2.1.1 in the f11 Chapter Introduction) in the arrays **a**, **irow** and **icol**, as returned from nag_sparse_nherm_fac (f11dnc). The array **a** holds the nonzero entries in these matrices, while **irow** and **icol** hold the corresponding row and column indices.

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

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: **n** – Integer *Input*

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_nherm_fac (f11dnc).

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

3: **nnz** – Integer *Input*

On entry: the number of nonzero elements in the matrix A. This **must** be the same value as was supplied in the preceding call to nag_sparse_nherm_fac (f11dnc).

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

4: **a**[**la**] – const Complex *Input*

On entry: the values returned in the array **a** by a previous call to nag_sparse_nherm_fac (f11dnc).

5: **la** – Integer *Input*

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_nherm_fac (f11dnc).

Constraint: $\mathbf{la} \geq 2 \times \mathbf{nnz}$.

6: **irow**[**la**] – const Integer *Input*
7: **icol**[**la**] – const Integer *Input*
8: **ipivp**[**n**] – const Integer *Input*
9: **ipivq**[**n**] – const Integer *Input*
10: **istr**[$\mathbf{n}+\mathbf{1}$] – const Integer *Input*
11: **idiag**[**n**] – const Integer *Input*

On entry: the values returned in arrays **irow**, **icol**, **ipivp**, **ipivq**, **istr** and **idiag** by a previous call to nag_sparse_nherm_fac (f11dnc).

ipivp and **ipivq** are restored on exit.

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

On entry: the right-hand side vector b.

13: **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, $0 < \mathbf{m} \leq \min(\mathbf{n}, 50)$;
if **method** = Nag_SparseNsym_BiCGSTAB, $0 < \mathbf{m} \leq \min(\mathbf{n}, 10)$.

14: **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.

15: **maxitn** – Integer *Input*

On entry: the maximum number of iterations allowed.

Constraint: **maxitn** ≥ 1.

16: **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.

17: **rnorm** – double * *Output*

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

18: **itn** – Integer * *Output*

On exit: the number of iterations carried out.

19: **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 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.

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_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, **la** $= \langle value \rangle$ and **nnz** $= \langle value \rangle$.
Constraint: **la** $\geq 2 \times$ **nnz**.

On entry, **m** $= \langle value \rangle$ and **n** $= \langle value \rangle$.
Constraint: **m** ≥ 1 and **m** $\leq \min($**n**$, \langle value \rangle)$.

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

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**.
Check that **a**, **irow**, **icol**, **ipivp**, **ipivq**, **istr** and **idiag** have not been corrupted between calls to nag_sparse_nherm_fac_sol (f11dqc) and nag_sparse_nherm_fac (f11dnc).

On entry, $i = \langle value \rangle$, **irow**$[i-1] = \langle value \rangle$, **n** $= \langle value \rangle$.
Constraint: **irow**$[i-1] \geq 1$ and **irow**$[i-1] \leq$ **n**.
Check that **a**, **irow**, **icol**, **ipivp**, **ipivq**, **istr** and **idiag** have not been corrupted between calls to nag_sparse_nherm_fac_sol (f11dqc) and nag_sparse_nherm_fac (f11dnc).

NE_INVALID_CS_PRECOND

The CS representation of the preconditioner is invalid.
Check that **a**, **irow**, **icol**, **ipivp**, **ipivq**, **istr** and **idiag** have not been corrupted between calls to nag_sparse_nherm_fac (f11dnc) and nag_sparse_nherm_fac_sol (f11dqc).

NE_NOT_STRICTLY_INCREASING

On entry, **a**$[i-1]$ is out of order: $i = \langle value \rangle$.
Check that **a**, **irow**, **icol**, **ipivp**, **ipivq**, **istr** and **idiag** have not been corrupted between calls to nag_sparse_nherm_fac_sol (f11dqc) and nag_sparse_nherm_fac (f11dnc).

On entry, the location (**irow**$[i-1]$, **icol**$[i-1]$) is a duplicate: $i = \langle value \rangle$.
Check that **a**, **irow**, **icol**, **ipivp**, **ipivq**, **istr** and **idiag** have not been corrupted between calls to nag_sparse_nherm_fac_sol (f11dqc) and nag_sparse_nherm_fac (f11dnc).

NE_REAL

On entry, **tol** = ⟨*value*⟩.
Constraint: **tol** < 1.0.

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_nherm_fac_sol (f11dqc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_sparse_nherm_fac_sol (f11dqc) 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_nherm_fac_sol (f11dqc) for each iteration is roughly proportional to the value of **nnzc** returned from the preceding call to nag_sparse_nherm_fac (f11dnc).

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 coefficient matrix $\bar{A} = M^{-1}A$.

10 Example

This example solves a complex sparse non-Hermitian linear system of equations using the CGS method, with incomplete LU preconditioning.

10.1 Program Text

```
/* nag_sparse_nherm_fac_sol (f11dqc) 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               dtol, rnorm, tol;
  Integer              i, itn, la, lfill, m, maxitn, n, nnz, nnzc, npivm;
  /* Arrays */
  Complex              *a = 0, *b = 0, *x = 0;
  Integer              *icol = 0, *idiag = 0, *ipivp = 0, *ipivq = 0,
                       *irow = 0, *istr = 0;
  char                 nag_enum_arg[40];
  /* NAG types */
```

```
  Nag_SparseNsym_Method method;
  Nag_SparseNsym_Piv    pstrat;
  Nag_SparseNsym_Fact   milu;
  NagError              fail;

  INIT_FAIL(fail);

  printf("nag_sparse_nherm_fac_sol (f11dqc) Example Program Results\n\n");

  /* Skip heading in data file*/
  scanf("%*[^\n]");
  /* Read algorithmic parameters*/
  scanf("%ld%ld%*[^\n]", &n, &m);
  scanf("%ld%*[^\n]", &nnz);
  la = 2 * nnz;
  if (
      !(a = NAG_ALLOC((la), Complex)) ||
      !(b = NAG_ALLOC((n), Complex)) ||
      !(x = NAG_ALLOC((n), Complex)) ||
      !(icol = NAG_ALLOC((la), Integer)) ||
      !(idiag = NAG_ALLOC((n), Integer)) ||
      !(ipivp = NAG_ALLOC((n), Integer)) ||
      !(ipivq = NAG_ALLOC((n), Integer)) ||
      !(irow = NAG_ALLOC((la), Integer)) ||
      !(istr = NAG_ALLOC((n + 1), Integer))
      ) {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
  scanf("%39s%*[^\n]", nag_enum_arg);
  /* 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);
  scanf("%ld%lf%*[^\n]", &lfill, &dtol);
  scanf("%39s%*[^\n]", nag_enum_arg);
  pstrat = (Nag_SparseNsym_Piv) nag_enum_name_to_value(nag_enum_arg);
  scanf("%39s%*[^\n]", nag_enum_arg);
  milu = (Nag_SparseNsym_Fact) nag_enum_name_to_value(nag_enum_arg);
  scanf("%lf%ld%*[^\n]", &tol, &maxitn);
  /* 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);

  /* Calculate incomplete LU factorization*/
  /* nag_sparse_nherm_fac (f11dnc)
   * Complex sparse non-Hermitian linear systems, incomplete LU factorization
   */
  nag_sparse_nherm_fac(n, nnz, a, la, irow, icol, lfill, dtol, pstrat, milu,
                       ipivp, ipivq, istr, idiag, &nnzc, &npivm, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_sparse_nherm_fac (f11dnc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }
  /* solve ax = b */
  /* nag_sparse_nherm_fac_sol (f11dqc).
   * Solution of complex sparse non-Hermitian linear system, RGMRES, CGS,
   * Bi-CGSTAB or TFQMR method, preconditioner computed by
   * nag_sparse_nherm_fac (f11dnc) (Black Box).
   */
  nag_sparse_nherm_fac_sol(method, n, nnz, a, la, irow, icol, ipivp, ipivq,
                           istr, idiag, b, m, tol, maxitn, x, &rnorm, &itn,
                           &fail);
  if (fail.code != NE_NOERROR) {
```

```
    printf("Error from nag_sparse_nherm_fac_sol (f11dqc).\n%s\n",
           fail.message);
    exit_status = 2;
    goto END;
  }
  printf("Converged in%12ld iterations\n", itn);
  printf("Final residual norm =%11.3e\n\n", rnorm);
  /* Output x*/
  printf("%16s\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(idiag);
  NAG_FREE(ipivp);
  NAG_FREE(ipivq);
  NAG_FREE(irow);
  NAG_FREE(istr);
  return exit_status;
}
```

10.2 Program Data

```
nag_sparse_nherm_fac_sol (f11dqc) Example Program Data
  8           4              : n, m
  24                         : nnz
  Nag_SparseNsym_CGS         : method
  0           0.0            : lfill, dtol
  Nag_SparseNsym_CompletePiv : pstrat
  Nag_SparseNsym_UnModFact   : milu
  1.0E-10   100              : tol, maxitn
 ( 2., 1.)    1    1
 (-1., 1.)    1    4
 ( 1.,-3.)    1    8
 ( 4., 7.)    2    1
 (-3., 0.)    2    2
 ( 2., 4.)    2    5
 (-7.,-5.)    3    3
 ( 2., 1.)    3    6
 ( 3., 2.)    4    1
 (-4., 2.)    4    3
 ( 0., 1.)    4    4
 ( 5.,-3.)    4    7
 (-1., 2.)    5    2
 ( 8., 6.)    5    5
 (-3.,-4.)    5    7
 (-6.,-2.)    6    1
 ( 5.,-2.)    6    3
 ( 2., 0.)    6    6
 ( 0.,-5.)    7    3
 (-1., 5.)    7    5
 ( 6., 2.)    7    7
 (-1., 4.)    8    2
 ( 2., 0.)    8    6
 ( 3., 3.)    8    8          : a[i], irow[i], icol[i], i=0,...,nnz-1
 (  7., 11.)
 (  1., 24.)
 (-13.,-18.)
 (-10.,  3.)
 ( 23., 14.)
 ( 17., -7.)
 ( 15., -3.)
 ( -3., 20.)                 : b[i], i=0,...,n-1
 (  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_nherm_fac_sol (f11dqc) Example Program Results

Converged in           4 iterations
Final residual norm =  1.348e-11

        Solution
 (   1.0000e+00,    1.0000e+00)
 (   2.0000e+00,   -1.0000e+00)
 (   3.0000e+00,    1.0000e+00)
 (   4.0000e+00,   -1.0000e+00)
 (   3.0000e+00,   -1.0000e+00)
 (   2.0000e+00,    1.0000e+00)
 (   1.0000e+00,   -1.0000e+00)
 (  -1.7424e-12,    3.0000e+00)
```