NAG Library Function Document

nag_sparse_nherm_jacobi (f11dxc)

1 Purpose

nag_sparse_nherm_jacobi (f11dxc) computes the **approximate** solution of a complex, Hermitian or non-Hermitian, sparse system of linear equations applying a number of Jacobi iterations. It is expected that nag_sparse_nherm_jacobi (f11dxc) will be used as a preconditioner for the iterative solution of complex sparse systems of equations.

2 Specification

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

```
void nag_sparse_nherm_jacobi (Nag_SparseNsym_Store store,
     Nag_TransType trans, Nag_InitializeA init, Integer niter, Integer n,
     Integer nnz, const Complex a[], const Integer irow[],
     const Integer icol[], Nag_SparseNsym_CheckData check, const Complex b[],
     Complex x[], Complex diag[], NagError *fail)
```

3 Description

nag_sparse_nherm_jacobi (f11dxc) computes the **approximate** solution of the complex sparse system of linear equations $Ax = b$ using **niter** iterations of the Jacobi algorithm (see also Golub and Van Loan (1996) and Young (1971)):

$$x_{k+1} = x_k + D^{-1}(b - Ax_k) \tag{1}$$

where $k = 1, 2, \ldots,$ **niter** and $x_0 = 0$.

nag_sparse_nherm_jacobi (f11dxc) can be used both for non-Hermitian and Hermitian systems of equations. For Hermitian matrices, either all nonzero elements of the matrix A can be supplied using coordinate storage (CS), or only the nonzero elements of the lower triangle of A, using symmetric coordinate storage (SCS) (see the f11 Chapter Introduction).

It is expected that nag_sparse_nherm_jacobi (f11dxc) will be used as a preconditioner for the iterative solution of complex sparse systems of equations. This may be with either the Hermitian or non-Hermitian suites of functions.

For Hermitian systems the suite consists of:

nag_sparse_herm_basic_setup (f11grc),

nag_sparse_herm_basic_solver (f11gsc),

nag_sparse_herm_basic_diagnostic (f11gtc).

For non-Hermitian systems the suite consists of:

nag_sparse_nherm_basic_setup (f11brc),

nag_sparse_nherm_basic_solver (f11bsc),

nag_sparse_nherm_basic_diagnostic (f11btc).

4 References

Golub G H and Van Loan C F (1996) *Matrix Computations* (3rd Edition) Johns Hopkins University Press, Baltimore

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

5 Arguments

1: **store** – Nag_SparseNsym_Store *Input*

On entry: specifies whether the matrix A is stored using symmetric coordinate storage (SCS) (applicable only to a Hermitian matrix A) or coordinate storage (CS) (applicable to both Hermitian and non-Hermitian matrices).

store = Nag_SparseNsym_StoreCS
The complete matrix A is stored in CS format.

store = Nag_SparseNsym_StoreSCS
The lower triangle of the Hermitian matrix A is stored in SCS format.

Constraint: **store** = Nag_SparseNsym_StoreCS or Nag_SparseNsym_StoreSCS.

2: **trans** – Nag_TransType *Input*

On entry: if **store** = Nag_SparseNsym_StoreCS, specifies whether the approximate solution of $Ax = b$ or of $A^{\mathrm{T}}x = b$ is required.

trans = Nag_NoTrans
The approximate solution of $Ax = b$ is calculated.

trans = Nag_Trans
The approximate solution of $A^{\mathrm{T}}x = b$ is calculated.

Suggested value: if the matrix A is Hermitian and stored in CS format, it is recommended that **trans** = Nag_NoTrans for reasons of efficiency.

Constraint: **trans** = Nag_NoTrans or Nag_Trans.

3: **init** – Nag_InitializeA *Input*

On entry: on first entry, **init** should be set to Nag_InitializeI, unless the diagonal elements of A are already stored in the array **diag**. If **diag** already contains the diagonal of A, it must be set to Nag_InputA.

init = Nag_InputA
diag must contain the diagonal of A.

init = Nag_InitializeI
diag will store the diagonal of A on exit.

Suggested value: **init** = Nag_InitializeI on first entry; **init** = Nag_InputA, subsequently, unless **diag** has been overwritten.

Constraint: **init** = Nag_InputA or Nag_InitializeI.

4: **niter** – Integer *Input*

On entry: the number of Jacobi iterations requested.

Constraint: **niter** ≥ 1.

5: **n** – Integer *Input*

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

Constraint: **n** ≥ 1.

6: **nnz** – Integer *Input*

On entry: if **store** = Nag_SparseNsym_StoreCS, the number of nonzero elements in the matrix A.

If **store** = Nag_SparseNsym_StoreSCS, the number of nonzero elements in the lower triangle of the matrix A.

Constraints:

if **store** $=$ Nag_SparseNsym_StoreCS, $1 \leq \mathbf{nnz} \leq \mathbf{n}^2$;
if **store** $=$ Nag_SparseNsym_StoreSCS, $1 \leq \mathbf{nnz} \leq \mathbf{n} \times (\mathbf{n}+1)/2$.

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

On entry: if **store** $=$ Nag_SparseNsym_StoreCS, the nonzero elements in the matrix A (CS format).

If **store** $=$ Nag_SparseNsym_StoreSCS, the nonzero elements in the lower triangle of the matrix A (SCS format).

In both cases, the elements of either A or of its lower triangle must be ordered by increasing row index and by increasing column index within each row. Multiple entries for the same row and columns indices are not permitted. The function nag_sparse_nherm_sort (f11znc) or nag_sparse_herm_sort (f11zpc) may be used to reorder the elements in this way for CS and SCS storage, respectively.

8: **irow**[**nnz**] – const Integer *Input*
9: **icol**[**nnz**] – const Integer *Input*

On entry: if **store** $=$ Nag_SparseNsym_StoreCS, the row and column indices of the nonzero elements supplied in **a**.

If **store** $=$ Nag_SparseNsym_StoreSCS, the row and column indices of the nonzero elements of the lower triangle of the matrix A supplied in **a**.

Constraints:

$1 \leq \mathbf{irow}[i] \leq \mathbf{n}$, for $i = 0, 1, \ldots, \mathbf{nnz} - 1$;
if **store** $=$ Nag_SparseNsym_StoreCS, $1 \leq \mathbf{icol}[i] \leq \mathbf{n}$, for $i = 0, 1, \ldots, \mathbf{nnz} - 1$;
if **store** $=$ Nag_SparseNsym_StoreSCS, $1 \leq \mathbf{icol}[i] \leq \mathbf{irow}[i]$, for $i = 0, 1, \ldots, \mathbf{nnz} - 1$;
either $\mathbf{irow}[i-1] < \mathbf{irow}[i]$ or both $\mathbf{irow}[i-1] = \mathbf{irow}[i]$ and $\mathbf{icol}[i-1] < \mathbf{icol}[i]$, for $i = 1, 2, \ldots, \mathbf{nnz} - 1$.

10: **check** – Nag_SparseNsym_CheckData *Input*

On entry: specifies whether or not the CS or SCS representation of the matrix A should be checked.

check $=$ Nag_SparseNsym_Check
Checks are carried out on the values of **n**, **nnz**, **irow**, **icol**; if **init** $=$ Nag_InputA, **diag** is also checked.

check $=$ Nag_SparseNsym_NoCheck
None of these checks are carried out.

See also Section 9.2.

Constraint: **check** $=$ Nag_SparseNsym_Check or Nag_SparseNsym_NoCheck.

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

On entry: the right-hand side vector b.

12: **x**[**n**] – Complex *Output*

On exit: the approximate solution vector $x_{\mathbf{niter}}$.

13: **diag**[**n**] – Complex *Input/Output*

On entry: if **init** $=$ Nag_InputA, the diagonal elements of A.

On exit: if **init** $=$ Nag_InputA, unchanged on exit.

If **init** $=$ Nag_InitializeI, the diagonal elements of A.

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

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_INT

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

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

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

NE_INT_2

On entry, $\mathbf{nnz} = \langle value \rangle$ and $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{nnz} \leq \mathbf{n} \times (\mathbf{n} + 1)/2$

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

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.

NE_INVALID_CS

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

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

On entry, $I = \langle value \rangle$, $\mathbf{irow}[I-1] = \langle value \rangle$ and $\mathbf{n} = \langle value \rangle$.
Constraint: $\mathbf{irow}[I-1] \geq 1$ and $\mathbf{irow}[I-1] \leq \mathbf{n}$.

NE_NOT_STRICTLY_INCREASING

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

On entry, the location $(\mathbf{irow}[I-1], \mathbf{icol}[I-1])$ is a duplicate: $I = \langle value \rangle$.

NE_ZERO_DIAG_ELEM

On entry, the diagonal element of the Ith row is zero or missing: $I = \langle value \rangle$.

On entry, the element $\mathbf{diag}[I-1]$ is zero: $I = \langle value \rangle$.

7 Accuracy

In general, the Jacobi method cannot be used on its own to solve systems of linear equations. The rate of convergence is bound by its spectral properties (see, for example, Golub and Van Loan (1996)) and as a solver, the Jacobi method can only be applied to a limited set of matrices. One condition that guarantees convergence is strict diagonal dominance.

However, the Jacobi method can be used successfully as a preconditioner to a wider class of systems of equations. The Jacobi method has good vector/parallel properties, hence it can be applied very efficiently. Unfortunately, it is not possible to provide criteria which define the applicability of the Jacobi method as a preconditioner, and its usefulness must be judged for each case.

8 Parallelism and Performance

nag_sparse_nherm_jacobi (f11dxc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_sparse_nherm_jacobi (f11dxc) 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

9.1 Timing

The time taken for a call to nag_sparse_nherm_jacobi (f11dxc) is proportional to **niter** × **nnz**.

9.2 Use of check

It is expected that a common use of nag_sparse_nherm_jacobi (f11dxc) will be as preconditioner for the iterative solution of complex, Hermitian or non-Hermitian, linear systems. In this situation, nag_sparse_nherm_jacobi (f11dxc) is likely to be called many times. In the interests of both reliability and efficiency, you are recommended to set **check** = Nag_SparseNsym_Check for the first of such calls, and to set **check** = Nag_SparseNsym_NoCheck for all subsequent calls.

10 Example

This example solves the complex sparse non-Hermitian system of equations $Ax = b$ iteratively using nag_sparse_nherm_jacobi (f11dxc) as a preconditioner.

10.1 Program Text

```
/* nag_sparse_nherm_jacobi (f11dxc) 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                 anorm, sigmax, stplhs, stprhs, tol;
  Integer                i, irevcm, iterm, itn, lwork, lwreq, m, maxitn,
                         monit, n, niter, nnz;
  /* Arrays */
```

```
char                    nag_enum_arg[100];
Complex                 *a = 0, *b = 0, *diag = 0, *work = 0, *x = 0;
double                  *wgt = 0;
Integer                 *icol = 0, *irow = 0;
/* NAG types */
Nag_InitializeA         init;
Nag_SparseNsym_Method   method;
Nag_SparseNsym_PrecType precon;
Nag_NormType            norm;
Nag_SparseNsym_Weight   weight;
NagError                fail, fail1;

INIT_FAIL(fail);
INIT_FAIL(fail1);

printf("nag_sparse_nherm_jacobi (f11dxc) Example Program Results\n");
/* Skip heading in data file*/
scanf("%*[^\n]");
scanf("%ld%*[^\n]", &n);
scanf("%ld%*[^\n]", &nnz);
lwork = 300;
if (
    !(a = NAG_ALLOC(nnz, Complex)) ||
    !(b = NAG_ALLOC(n, Complex)) ||
    !(diag = NAG_ALLOC(n, Complex)) ||
    !(work = NAG_ALLOC(lwork, Complex)) ||
    !(x = NAG_ALLOC(n, Complex)) ||
    !(wgt = NAG_ALLOC(n, double)) ||
    !(icol = NAG_ALLOC(nnz, Integer)) ||
    !(irow = NAG_ALLOC(nnz, Integer))
    ) {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
/* Read or initialize the parameters for the iterative solver*/
scanf("%99s%*[^\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("%99s%*[^\n]", nag_enum_arg);
precon = (Nag_SparseNsym_PrecType) nag_enum_name_to_value(nag_enum_arg);
scanf("%99s%*[^\n]", nag_enum_arg);
norm = (Nag_NormType) nag_enum_name_to_value(nag_enum_arg);
scanf("%99s%*[^\n]", nag_enum_arg);
weight = (Nag_SparseNsym_Weight) nag_enum_name_to_value(nag_enum_arg);
scanf("%ld%*[^\n]", &iterm);
scanf("%ld%lf%ld%*[^\n]", &m, &tol, &maxitn);
scanf("%ld%*[^\n]", &monit);
/* Read the parameters for the preconditioner*/
scanf("%ld%*[^\n]", &niter);
anorm = 0.0;
sigmax = 0.0;

/* Read the non-zero elements of 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 right-hand side vector b and initial approximate solution*/
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]");

/* Call to initialize the solver */
/* nag_sparse_nherm_basic_setup (f11brc).
 * Complex sparse non-Hermitian linear systems, setup
 */
nag_sparse_nherm_basic_setup(method, precon, norm, weight, iterm, n, m, tol,
                             maxitn, anorm, sigmax, monit, &lwreq, work,
```

```
                              lwork, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_sparse_nherm_basic_setup (f11brc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }
  /* Call solver repeatedly to solve the equations.
   * Note: the arrays b and x are overwritten; on final exit, x will
   *       contain the solution and b the residual vector.
   */
  irevcm = 0;
  init =Nag_InitializeI;
  while (irevcm != 4) {
    /* nag_sparse_nherm_basic_solver (f11bsc)
     * Complex sparse non-Hermitian linear systems, preconditioned RGMRES, CGS,
     * Bi-CGSTAB or TFQMR method
     */
    nag_sparse_nherm_basic_solver(&irevcm, x, b, wgt, work, lwreq, &fail);
    switch (irevcm) {
    case -1:
      /* nag_sparse_nherm_matvec (f11xnc)
       * Complex sparse non-Hermitian matrix vector multiply
       */
      nag_sparse_nherm_matvec(Nag_ConjTrans, n, nnz, a, irow, icol,
                              Nag_SparseNsym_NoCheck, x, b, &fail1);
      break;
    case 1:
      nag_sparse_nherm_matvec(Nag_NoTrans, n, nnz, a, irow, icol,
                              Nag_SparseNsym_NoCheck, x, b, &fail1);
      break;
    case 2:
      /* nag_sparse_nherm_jacobi (f11dxc).
       * Complex sparse nonsymmetric linear systems, line Jacobi preconditioner
       */
      nag_sparse_nherm_jacobi(Nag_SparseNsym_StoreCS, Nag_NoTrans, init,
                              niter, n, nnz, a, irow, icol,
                              Nag_SparseNsym_Check, x, b, diag, &fail1);
      init = Nag_InputA;
      break;
    case 3:
      /* nag_sparse_nherm_basic_diagnostic (f11btc)
       * Complex sparse nonhermitian linear systems, diagnostic
       */
      nag_sparse_nherm_basic_diagnostic(&itn, &stplhs, &stprhs, &anorm,
                                        &sigmax, work, lwreq,&fail1);
      if (fail1.code == NE_NOERROR)
        printf("Monitoring at iteration no.%4ld residual %14.4e\n",
               itn, stplhs);
    }
    if (fail1.code != NE_NOERROR) irevcm = 6;
  }
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_sparse_nherm_basic_solver (f11bsc)\n%s\n",
             fail.message);
      exit_status = 2;
      goto END;
    }
  /* Obtain information about the computation using
   * nag_sparse_nherm_basic_diagnostic (f11btc).
   * Complex sparse Hermitian linear systems, diagnostic.
   */
  nag_sparse_nherm_basic_diagnostic(&itn, &stplhs, &stprhs, &anorm, &sigmax,
                                    work, lwreq, &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_sparse_nherm_basic_diagnostic (f11btc)\n%s\n",
             fail.message);
      exit_status = 3;
      goto END;
```

```
    }
  /* Print the output data*/
  printf("\nFinal Results\n");
  printf("Number of iterations for convergence:      %4ld \n", itn);
  printf("Residual norm:                             %14.4e \n", stplhs);
  printf("Right-hand side of termination criterion: %14.4e\n", stprhs);
  printf("1-norm of matrix A:                        %14.4e\n", anorm);
  /* Output x*/
  printf("\n%20s%29s\n","Solution vector","Residual vector");
  for (i = 0; i < n; i++)
    printf("(%13.4e, %13.4e)   (%13.4e, %13.4e)\n", x[i].re, x[i].im, b[i].re,
           b[i].im);
 END:
  NAG_FREE(a);
  NAG_FREE(b);
  NAG_FREE(diag);
  NAG_FREE(work);
  NAG_FREE(x);
  NAG_FREE(wgt);
  NAG_FREE(icol);
  NAG_FREE(irow);
  return exit_status;
}
```

10.2 Program Data

```
nag_sparse_nherm_jacobi (f11dxc) Example Program Data
  8                         : n
 24                         : nnz
  Nag_SparseNsym_TFQMR      : method
  Nag_SparseNsym_Prec       : precon
  Nag_OneNorm               : norm
  Nag_SparseNsym_UnWeighted : weight
  1                         : iterm
  2  1.0E-6   20            : m, tol, maxitn
  1                         : monit
  2                         : niter
 (  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_jacobi (f11dxc) Example Program Results
Monitoring at iteration no.   1 residual     1.5062e+02
Monitoring at iteration no.   2 residual     1.5704e+02
Monitoring at iteration no.   3 residual     1.4803e+02
Monitoring at iteration no.   4 residual     8.5215e+01
Monitoring at iteration no.   5 residual     4.2951e+01
Monitoring at iteration no.   6 residual     2.5055e+01
Monitoring at iteration no.   7 residual     1.9090e-01

Final Results
Number of iterations for convergence:       8
Residual norm:                               9.5485e-08
Right-hand side of termination criterion:    8.9100e-04
1-norm of matrix A:                          2.7000e+01

     Solution vector              Residual vector
(   1.0000e+00,    1.0000e+00)   (   4.7145e-09,   -3.9432e-09)
(   2.0000e+00,   -1.0000e+00)   (   8.7691e-09,   -9.8075e-09)
(   3.0000e+00,    1.0000e+00)   (  -2.8739e-09,    4.1582e-09)
(   4.0000e+00,   -1.0000e+00)   (   3.5787e-09,   -1.1121e-08)
(   3.0000e+00,   -1.0000e+00)   (  -6.9210e-09,   -8.6888e-09)
(   2.0000e+00,    1.0000e+00)   (  -2.2468e-09,    8.4933e-09)
(   1.0000e+00,   -1.0000e+00)   (   5.2201e-10,   -3.3375e-09)
(   2.4713e-09,    3.0000e+00)   (  -5.5778e-09,   -1.0732e-08)
```
