NAG Library Function Document

nag_sparse_nherm_basic_setup (f11brc)

1 Purpose

nag_sparse_nherm_basic_setup (f11brc) is a setup function, the first in a suite of three functions for the iterative solution of a complex general (non-Hermitian) system of simultaneous linear equations. nag_sparse_nherm_basic_setup (f11brc) must be called before nag_sparse_nherm_basic_solver (f11bsc), the iterative solver. The third function in the suite, nag_sparse_nherm_basic_diagnostic (f11btc), can be used to return additional information about the computation.

These three functions are suitable for the solution of large sparse general (non-Hermitian) systems of equations.

2 Specification

```
#include <nag.h>
#include <nagf11.h>
void nag_sparse_nherm_basic_setup (Nag_SparseNsym_Method method,
     Nag_SparseNsym_PrecType precon, Nag_NormType norm,
     Nag_SparseNsym_Weight weight, Integer iterm, Integer n, Integer m,
     double tol, Integer maxitn, double anorm, double sigmax, Integer monit,
     Integer *lwreq, Complex work[], Integer lwork, NagError *fail)
```

3 Description

The suite consisting of the functions nag_sparse_nherm_basic_setup (f11brc), nag_sparse_nherm_basic_solver (f11bsc) and nag_sparse_nherm_basic_diagnostic (f11btc) is designed to solve the general (non-Hermitian) system of simultaneous linear equations $Ax = b$ of order n, where n is large and the coefficient matrix A is sparse.

nag_sparse_nherm_basic_setup (f11brc) is a setup function which must be called before nag_sparse_nherm_basic_solver (f11bsc), the iterative solver. The third function in the suite, nag_sparse_nherm_basic_diagnostic (f11btc), can be used to return additional information about the computation. A choice of methods is available:

restarted generalized minimum residual method (RGMRES);

conjugate gradient squared method (CGS);

bi-conjugate gradient stabilized (ℓ) method (Bi-CGSTAB(ℓ));

transpose-free quasi-minimal residual method (TFQMR).

3.1 Restarted Generalized Minimum Residual Method (RGMRES)

The restarted generalized minimum residual method (RGMRES) (see Saad and Schultz (1986), Barrett *et al.* (1994) and Dias da Cunha and Hopkins (1994)) starts from the residual $r_0 = b - Ax_0$, where x_0 is an initial estimate for the solution (often $x_0 = 0$). An orthogonal basis for the Krylov subspace span$\{A^k r_0\}$, for $k = 0, 1, \ldots$, is generated explicitly: this is referred to as Arnoldi's method (see Arnoldi (1951)). The solution is then expanded onto the orthogonal basis so as to minimize the residual norm $\|b - Ax\|_2$. The lack of symmetry of A implies that the orthogonal basis is generated by applying a 'long' recurrence relation, whose length increases linearly with the iteration count. For all but the most trivial problems, computational and storage costs can quickly become prohibitive as the iteration count increases. RGMRES limits these costs by employing a restart strategy: every m iterations at most, the Arnoldi process is restarted from $r_l = b - Ax_l$, where the subscript l denotes the last available iterate. Each group of m iterations is referred to as a 'super-iteration'. The value of m is chosen in advance and is fixed throughout the computation. Unfortunately, an optimum value of m cannot easily be predicted.

3.2 Conjugate Gradient Squared Method (CGS)

The conjugate gradient squared method (CGS) (see Sonneveld (1989), Barrett *et al.* (1994) and Dias da Cunha and Hopkins (1994)) is a development of the bi-conjugate gradient method where the nonsymmetric Lanczos method is applied to reduce the coefficients matrix to tridiagonal form: two bi-orthogonal sequences of vectors are generated starting from the residual $r_0 = b - Ax_0$, where x_0 is an initial estimate for the solution (often $x_0 = 0$) and from the *shadow residual* $\hat{r}_0$ corresponding to the arbitrary problem $A^{\mathrm{H}}\hat{x} = \hat{b}$, where $\hat{b}$ can be any vector, but in practice is chosen so that $r_0 = \hat{r}_0$. In the course of the iteration, the residual and shadow residual $r_i = P_i(A)r_0$ and $\hat{r}_i = P_i(A^{\mathrm{H}})\hat{r}_0$ are generated, where P_i is a polynomial of order i, and bi-orthogonality is exploited by computing the vector product $\rho_i = (\hat{r}_i, r_i) = (P_i(A^{\mathrm{H}})\hat{r}_0,$
$P_i(A)r_0) = \left(\hat{r}_0, P_i^2(A)r_0\right)$. Applying the 'contraction' operator $P_i(A)$ twice, the iteration coefficients can still be recovered without advancing the solution of the shadow problem, which is of no interest. The CGS method often provides fast convergence; however, there is no reason why the contraction operator should also reduce the once reduced vector $P_i(A)r_0$: this may well lead to a highly irregular convergence which may result in large cancellation errors.

3.3 Bi-Conjugate Gradient Stabilized (ℓ) Method (Bi-CGSTAB(ℓ))

The bi-conjugate gradient stabilized (ℓ) method (Bi-CGSTAB(ℓ)) (see Van der Vorst (1989), Sleijpen and Fokkema (1993) and Dias da Cunha and Hopkins (1994)) is similar to the CGS method above. However, instead of generating the sequence $\left\{P_i^2(A)r_0\right\}$, it generates the sequence $\{Q_i(A)P_i(A)r_0\}$, where the $Q_i(A)$ are polynomials chosen to minimize the residual *after* the application of the contraction operator $P_i(A)$. Two main steps can be identified for each iteration: an OR (Orthogonal Residuals) step where a basis of order ℓ is generated by a Bi-CG iteration and an MR (Minimum Residuals) step where the residual is minimized over the basis generated, by a method akin to GMRES. For $\ell = 1$, the method corresponds to the Bi-CGSTAB method of Van der Vorst (1989). For $\ell > 1$, more information about complex eigenvalues of the iteration matrix can be taken into account, and this may lead to improved convergence and robustness. However, as ℓ increases, numerical instabilities may arise. For this reason, a maximum value of $\ell = 10$ is imposed, but probably $\ell = 4$ is sufficient in most cases.

3.4 Transpose-free Quasi-minimal Residual Method (TFQMR)

The transpose-free quasi-minimal residual method (TFQMR) (see Freund and Nachtigal (1991) and Freund (1993)) is conceptually derived from the CGS method. The residual is minimized over the space of the residual vectors generated by the CGS iterations under the simplifying assumption that residuals are almost orthogonal. In practice, this is not the case but theoretical analysis has proved the validity of the method. This has the effect of remedying the rather irregular convergence behaviour with wild oscillations in the residual norm that can degrade the numerical performance and robustness of the CGS method. In general, the TFQMR method can be expected to converge at least as fast as the CGS method, in terms of number of iterations, although each iteration involves a higher operation count. When the CGS method exhibits irregular convergence, the TFQMR method can produce much smoother, almost monotonic convergence curves. However, the close relationship between the CGS and TFQMR method implies that the *overall* speed of convergence is similar for both methods. In some cases, the TFQMR method may converge faster than the CGS method.

3.5 General Considerations

For each method, a sequence of solution iterates $\{x_i\}$ is generated such that, hopefully, the sequence of the residual norms $\{\|r_i\|\}$ converges to a required tolerance. Note that, in general, convergence, when it occurs, is not monotonic.

In the RGMRES and Bi-CGSTAB(ℓ) methods above, your program must provide the **maximum** number of basis vectors used, m or ℓ, respectively; however, a **smaller** number of basis vectors may be generated and used when the stability of the solution process requires this (see Section 9).

Faster convergence can be achieved using a **preconditioner** (see Golub and Van Loan (1996) and Barrett *et al.* (1994)). A preconditioner maps the original system of equations onto a different system, say

$$\bar{A}\bar{x} = \bar{b}, \tag{1}$$

with, hopefully, better characteristics with respect to its speed of convergence: for example, the condition number of the coefficients matrix can be improved or eigenvalues in its spectrum can be made to coalesce. An orthogonal basis for the Krylov subspace span$\{\bar{A}^k\bar{r}_0\}$, for $k = 0, 1, \ldots$, is generated and the solution proceeds as outlined above. The algorithms used are such that the solution and residual iterates of the original system are produced, not their preconditioned counterparts. Note that an unsuitable preconditioner or no preconditioning at all may result in a very slow rate, or lack, of convergence. However, preconditioning involves a trade-off between the reduction in the number of iterations required for convergence and the additional computational costs per iteration. Also, setting up a preconditioner may involve non-negligible overheads.

A *left* preconditioner M^{-1} can be used by the RGMRES, CGS and TFQMR methods, such that $\bar{A} = M^{-1}A \sim I_n$ in (1), where I_n is the identity matrix of order n; a *right* preconditioner M^{-1} can be used by the Bi-CGSTAB(ℓ) method, such that $\bar{A} = AM^{-1} \sim I_n$. These are formal definitions, used only in the design of the algorithms; in practice, only the means to compute the matrix–vector products $v = Au$ and $v = A^{\mathrm{H}}u$ (the latter only being required when an estimate of $\|A\|_1$ or $\|A\|_\infty$ is computed internally), and to solve the preconditioning equations $Mv = u$ are required, i.e., explicit information about M, or its inverse is not required at any stage.

The first termination criterion

$$\|r_k\|_p \leq \tau\Big(\|b\|_p + \|A\|_p \times \|x_k\|_p\Big) \tag{2}$$

is available for all four methods. In (2), $p = 1$, ∞ or 2 and τ denotes a user-specified tolerance subject to $\max(10, \sqrt{n})$, $\epsilon \leq \tau < 1$, where ϵ is the ***machine precision***. Facilities are provided for the estimation of the norm of the coefficients matrix $\|A\|_1$ or $\|A\|_\infty$, when this is not known in advance, by applying Higham's method (see Higham (1988)). Note that $\|A\|_2$ cannot be estimated internally. This criterion uses an error bound derived from **backward** error analysis to ensure that the computed solution is the exact solution of a problem as close to the original as the termination tolerance requires. Termination criteria employing bounds derived from **forward** error analysis are not used because any such criteria would require information about the condition number $\kappa(A)$ which is not easily obtainable.

The second termination criterion

$$\|\bar{r}_k\|_2 \leq \tau\big(\|\bar{r}_0\|_2 + \sigma_1\big(\bar{A}\big) \times \|\Delta\bar{x}_k\|_2\big) \tag{3}$$

is available for all methods except TFQMR. In (3), $\sigma_1\big(\bar{A}\big) = \big\|\bar{A}\big\|_2$ is the largest singular value of the (preconditioned) iteration matrix $\bar{A}$. This termination criterion monitors the progress of the solution of the preconditioned system of equations and is less expensive to apply than criterion (2) for the Bi-CGSTAB(ℓ) method with $\ell > 1$. Only the RGMRES method provides facilities to estimate $\sigma_1\big(\bar{A}\big)$ internally, when this is not supplied (see Section 9).

Termination criterion (2) is the recommended choice, despite its additional costs per iteration when using the Bi-CGSTAB(ℓ) method with $\ell > 1$. Also, if the norm of the initial estimate is much larger than the norm of the solution, that is, if $\|x_0\| \gg \|x\|$, a dramatic loss of significant digits could result in complete lack of convergence. The use of criterion (2) will enable the detection of such a situation, and the iteration will be restarted at a suitable point. No such restart facilities are provided for criterion (3).

Optionally, a vector w of user-specified weights can be used in the computation of the vector norms in termination criterion (2), i.e., $\|v\|_p^{(w)} = \big\|v^{(w)}\big\|_p$, where $\big(v^{(w)}\big)_i = w_i v_i$, for $i = 1, 2, \ldots, n$. Note that the use of weights increases the computational costs.

The sequence of calls to the functions comprising the suite is enforced: first, the setup function nag_sparse_nherm_basic_setup (f11brc) must be called, followed by the solver nag_sparse_nherm_basic_solver (f11bsc). nag_sparse_nherm_basic_diagnostic (f11btc) can be called either when nag_sparse_nherm_basic_solver (f11bsc) is carrying out a monitoring step or after nag_sparse_nherm_basic_solver (f11bsc) has completed its tasks. Incorrect sequencing will raise an error condition.

In general, it is not possible to recommend one method in preference to another. RGMRES is often used in the solution of systems arising from PDEs. On the other hand, it can easily stagnate when the size m of the orthogonal basis is too small, or the preconditioner is not good enough. CGS can be the fastest

method, but the computed residuals can exhibit instability which may greatly affect the convergence and quality of the solution. Bi-CGSTAB(ℓ) seems robust and reliable, but it can be slower than the other methods: if a preconditioner is used and $\ell > 1$, Bi-CGSTAB(ℓ) computes the solution of the preconditioned system $\bar{x}_k = Mx_k$: the preconditioning equations must be solved to obtain the required solution. The algorithm employed limits to 10% or less, when no intermediate monitoring is requested, the number of times the preconditioner has to be thus applied compared with the total number of applications of the preconditioner. TFQMR can be viewed as a more robust variant of the CGS method: it shares the CGS method speed but avoids the CGS fluctuations in the residual, which may give rise to instability. Also, when the termination criterion (2) is used, the CGS, Bi-CGSTAB(ℓ) and TFQMR methods will restart the iteration automatically when necessary in order to solve the given problem.

4 References

Arnoldi W (1951) The principle of minimized iterations in the solution of the matrix eigenvalue problem *Quart. Appl. Math.* **9** 17–29

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

Dias da Cunha R and Hopkins T (1994) PIM 1.1 — the parallel iterative method package for systems of linear equations user's guide — Fortran 77 version *Technical Report* Computing Laboratory, University of Kent at Canterbury, Kent, UK

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

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

Higham N J (1988) FORTRAN codes for estimating the one-norm of a real or complex matrix, with applications to condition estimation *ACM Trans. Math. Software* **14** 381–396

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: 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: determines whether preconditioning is used.

precon = Nag_SparseNsym_NoPrec
No preconditioning.

precon = Nag_SparseNsym_Prec
Preconditioning.

Constraint: **precon** = Nag_SparseNsym_NoPrec or Nag_SparseNsym_Prec.

3: **norm** – Nag_NormType *Input*

On entry: defines the matrix and vector norm to be used in the termination criteria.

norm = Nag_OneNorm
l_1 norm.

norm = Nag_InfNorm
l_∞ norm.

norm = Nag_TwoNorm
l_2 norm.

Suggested value:

if **iterm** = 1, **norm** = Nag_InfNorm;
if **iterm** = 2, **norm** = Nag_TwoNorm.

Constraints:

if **iterm** = 1, **norm** = Nag_OneNorm, Nag_InfNorm or Nag_TwoNorm;
if **iterm** = 2, **norm** = Nag_TwoNorm.

4: **weight** – Nag_SparseNsym_Weight *Input*

On entry: specifies whether a vector w of user-supplied weights is to be used in the computation of the vector norms required in termination criterion (2) (**iterm** = 1): $\|v\|_p^{(w)} = \left\|v^{(w)}\right\|_p$, where $v_i^{(w)} = w_i v_i$, for $i = 1, 2, \ldots, n$. The suffix $p = 1, 2, \infty$ denotes the vector norm used, as specified by the argument **norm**. Note that weights cannot be used when **iterm** = 2, i.e., when criterion (3) is used.

weight = Nag_SparseNsym_Weighted
User-supplied weights are to be used and must be supplied on initial entry to nag_sparse_nherm_basic_solver (f11bsc).

weight = Nag_SparseNsym_UnWeighted
All weights are implicitly set equal to one. Weights do not need to be supplied on initial entry to nag_sparse_nherm_basic_solver (f11bsc).

Suggested value: **weight** = Nag_SparseNsym_UnWeighted.

Constraints:

if **iterm** = 1, **weight** = Nag_SparseNsym_Weighted or Nag_SparseNsym_UnWeighted;
if **iterm** = 2, **weight** = Nag_SparseNsym_UnWeighted.

5: **iterm** – Integer *Input*

On entry: defines the termination criterion to be used.

iterm $= 1$
Use the termination criterion defined in (2).

iterm $= 2$
Use the termination criterion defined in (3).

Suggested value: **iterm** $= 1$.

Constraints:

if **method** $=$ Nag_SparseNsym_TFQMR or **weight** $=$ Nag_SparseNsym_Weighted or **norm** $\neq$ Nag_TwoNorm, **iterm** $= 1$;
otherwise **iterm** $= 1$ or 2.

Note: **iterm** $= 2$ is only appropriate for a restricted set of choices for **method**, **norm** and **weight**; that is **norm** $=$ Nag_TwoNorm, **weight** $=$ Nag_SparseNsym_UnWeighted and **method** $\neq$ Nag_SparseNsym_TFQMR.

6: **n** – Integer *Input*

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

Constraint: **n** > 0.

7: **m** – Integer *Input*

On entry: if **method** $=$ Nag_SparseNsym_RGMRES, **m** is the dimension m 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**$, 10)$.

8: **tol** – double *Input*

On entry: the tolerance τ for the termination criterion. If **tol** $\leq 0.0, \tau = \max(\sqrt{\epsilon}, \sqrt{n}\epsilon)$ is used, where ϵ is the ***machine precision***. Otherwise $\tau = \max($**tol**$, 10\epsilon, \sqrt{n}\epsilon)$ is used.

Constraint: **tol** < 1.0.

9: **maxitn** – Integer *Input*

On entry: the maximum number of iterations.

Constraint: **maxitn** > 0.

10: **anorm** – double *Input*

On entry: if **anorm** > 0.0, the value of $\|A\|_p$ to be used in the termination criterion (2) (**iterm** $= 1$).

If **anorm** ≤ 0.0, **iterm** $= 1$ and **norm** $=$ Nag_OneNorm or Nag_InfNorm, then $\|A\|_1 = \|A\|_\infty$ is estimated internally by nag_sparse_nherm_basic_solver (f11bsc).

If **iterm** $= 2$, **anorm** is not referenced.

Constraint: if **iterm** $= 1$ and **norm** $=$ Nag_TwoNorm, **anorm** > 0.0.

11: **sigmax** – double *Input*

On entry: if **iterm** $= 2$, the largest singular value σ_1 of the preconditioned iteration matrix; otherwise, **sigmax** is not referenced.

If **sigmax** ≤ 0.0, **iterm** $= 2$ and **method** $=$ Nag_SparseNsym_RGMRES, then the value of σ_1 will be estimated internally.

Constraint: if **method** $=$ Nag_SparseNsym_CGS or Nag_SparseNsym_BiCGSTAB and **iterm** $= 2$, **sigmax** > 0.0.

12: **monit** – Integer *Input*

On entry: if **monit** > 0, the frequency at which a monitoring step is executed by nag_sparse_nherm_basic_solver (f11bsc): if **method** $=$ Nag_SparseNsym_CGS or Nag_SparseNsym_TFQMR, a monitoring step is executed every **monit** iterations; otherwise, a monitoring step is executed every **monit** super-iterations (groups of up to m or ℓ iterations for RGMRES or Bi-CGSTAB(ℓ), respectively).

There are some additional computational costs involved in monitoring the solution and residual vectors when the Bi-CGSTAB(ℓ) method is used with $\ell > 1$.

Constraint: **monit** $\leq$ **maxitn**.

13: **lwreq** – Integer * *Output*

On exit: the minimum amount of workspace required by nag_sparse_nherm_basic_solver (f11bsc). (See also Section 5 in nag_sparse_nherm_basic_solver (f11bsc).)

14: **work**[**lwork**] – Complex *Communication Array*

On exit: the array **work** is initialized by nag_sparse_nherm_basic_setup (f11brc). It must **not** be modified before calling the next function in the suite, namely nag_sparse_nherm_basic_solver (f11bsc).

15: **lwork** – Integer *Input*

On entry: the dimension of the array **work**.

Constraint: **lwork** ≥ 120.

Note: although the minimum value of **lwork** ensures the correct functioning of nag_sparse_nherm_basic_setup (f11brc), a larger value is required by the other functions in the suite, namely nag_sparse_nherm_basic_solver (f11bsc) and nag_sparse_nherm_basic_diagnostic (f11btc). The required value is as follows:

Method	**Requirements**
RGMRES	**lwork** $= 120 + n(m+3) + m(m+5) + 1$, where m is the dimension of the basis.
CGS	**lwork** $= 120 + 7n$.
Bi-CGSTAB(ℓ)	**lwork** $= 120 + (2n + \ell)(\ell + 2) + p$, where ℓ is the order of the method.
TFQMR	**lwork** $= 120 + 10n$,

where

$p = 2n$ if $\ell > 1$ and **iterm** $= 2$ was supplied.

$p = n$ if $\ell > 1$ and a preconditioner is used or **iterm** $= 2$ was supplied.

$p = 0$ otherwise.

16: **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.
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_CONSTRAINT

On entry, **iterm** = 1, **norm** = Nag_TwoNorm and **anorm** = ⟨*value*⟩.
Constraint: if **iterm** = 1 and **norm** = Nag_TwoNorm, **anorm** > 0.0.

On entry, **iterm** = 2, **method** = ⟨*value*⟩ and **sigmax** = ⟨*value*⟩.
Constraint: if **iterm** = 2 and **method** = Nag_SparseNsym_CGS or Nag_SparseNsym_BiCGSTAB, **sigmax** > 0.0.

NE_ENUM_3_INT

On entry, **method** = ⟨*value*⟩, **weight** = ⟨*value*⟩, **norm** = ⟨*value*⟩ and **iterm** = ⟨*value*⟩.
Constraint: if **method** = Nag_SparseNsym_TFQMR or **weight** = Nag_SparseNsym_Weighted or **norm** ≠ Nag_TwoNorm, **iterm** = 1. Otherwise, **iterm** = 1 or 2.

NE_ENUM_INT_2

On entry, **m** = ⟨*value*⟩ and **method** = ⟨*value*⟩.
Constraint: if **method** = Nag_SparseNsym_RGMRES or Nag_SparseNsym_BiCGSTAB, **m** > 0.

On entry, **m** = ⟨*value*⟩, **n** = ⟨*value*⟩ and **method** = ⟨*value*⟩.
Constraint: if **method** = Nag_SparseNsym_RGMRES, **m** ≤ min(**n**, 50). If **method** = Nag_SparseNsym_BiCGSTAB, **m** ≤ min(**n**, 10).

NE_INT

On entry, **lwork** = ⟨*value*⟩.
Constraint: **lwork** ≥ 120.

On entry, **maxitn** = ⟨*value*⟩.
Constraint: **maxitn** > 0.

On entry, **n** = ⟨*value*⟩.
Constraint: **n** > 0.

NE_INT_2

On entry, **monit** = ⟨*value*⟩ and **maxitn** = ⟨*value*⟩.
Constraint: **monit** ≤ **maxitn**.

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.

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_OUT_OF_SEQUENCE

nag_sparse_nherm_basic_setup (f11brc) has been called out of sequence: either nag_sparse_nherm_basic_setup (f11brc) has been called twice or nag_sparse_nherm_basic_solver (f11bsc) has not terminated its current task.

NE_REAL

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

7 Accuracy

Not applicable.

8 Parallelism and Performance

Not applicable.

9 Further Comments

RGMRES can estimate internally the maximum singular value σ_1 of the iteration matrix, using $\sigma_1 \sim \|T\|_1$, where T is the upper triangular matrix obtained by QR factorization of the upper Hessenberg matrix generated by the Arnoldi process. The computational costs of this computation are negligible when compared to the overall costs.

Loss of orthogonality in the RGMRES method, or of bi-orthogonality in the Bi-CGSTAB(ℓ) method may degrade the solution and speed of convergence. For both methods, the algorithms employed include checks on the basis vectors so that the number of basis vectors used for a given super-iteration may be less than the value specified in the input argument **m**. Also, if termination criterion (2) is used, the CGS, Bi-CGSTAB(ℓ) and TFQMR methods will restart automatically the computation from the last available iterates, when the stability of the solution process requires it.

Termination criterion (3), when available, involves only the residual (or norm of the residual) produced directly by the iteration process: this may differ from the norm of the true residual $\tilde{r}_k = b - Ax_k$, particularly when the norm of the residual is very small. Also, if the norm of the initial estimate of the solution is much larger than the norm of the exact solution, convergence can be achieved despite very large errors in the solution. On the other hand, termination criterion (3) is cheaper to use and inspects the progress of the actual iteration. Termination criterion (2) should be preferred in most cases, despite its slightly larger costs.

10 Example

This example solves an 8×8 non-Hermitian system of simultaneous linear equations using the TFQMR method where the coefficients matrix A has a random sparsity pattern. An incomplete LU preconditioner is used (routines nag_sparse_nherm_fac (f11dnc) and nag_sparse_nherm_precon_ilu_solve (f11dpc)).

10.1 Program Text

```
/* nag_sparse_nherm_basic_setup (f11brc) Example Program.
 *
 * Copyright 2014 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, dtol, sigmax, stplhs, stprhs, tol;
  Integer                  i, irevcm, iterm, itn, la, lfill, lwork,
                           lwreq, m, maxitn, monit, n, nnz, nnzc, npivm;
  /* Arrays */
  char                     nag_enum_arg[100];
  Complex                  *a = 0, *b = 0, *work = 0, *x = 0;
  double                   *wgt = 0;
  Integer                  *icol = 0, *idiag = 0, *ipivp = 0, *ipivq = 0,
                           *irow = 0, *istr = 0;
  /* NAG types */
  Nag_SparseNsym_Piv       pstrat;
  Nag_SparseNsym_Fact      milu;
  Nag_SparseNsym_Method    method;
  Nag_SparseNsym_PrecType  precon;
  Nag_NormType             norm;
  Nag_SparseNsym_Weight    weight;
  Nag_TransType            trans;
  Nag_SparseNsym_CheckData check = Nag_SparseNsym_NoCheck;
  NagError                 fail, fail1;

  INIT_FAIL(fail);
  INIT_FAIL(fail1);

  printf("nag_sparse_nherm_basic_setup (f11brc) Example Program Results\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
  la = 2 * nnz;
  lwork = 200;
  if (
      !(a = NAG_ALLOC((la), Complex)) ||
      !(b = NAG_ALLOC((n), Complex)) ||
      !(work = NAG_ALLOC((lwork), Complex)) ||
      !(x = NAG_ALLOC((n), Complex)) ||
      !(wgt = NAG_ALLOC((n), double)) ||
      !(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;
  }

  /* Read or initialize the parameters for the iterative solver */
#ifdef _WIN32
  scanf_s("%99s%*[^\n]", nag_enum_arg, _countof(nag_enum_arg));
#else
  scanf("%99s%*[^\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("%99s%*[^\n]", nag_enum_arg, _countof(nag_enum_arg));
#else
  scanf("%99s%*[^\n]", nag_enum_arg);
#endif
  precon = (Nag_SparseNsym_PrecType) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
  scanf_s("%99s%*[^\n]", nag_enum_arg, _countof(nag_enum_arg));
#else
  scanf("%99s%*[^\n]", nag_enum_arg);
#endif
  norm = (Nag_NormType) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
  scanf_s("%99s%*[^\n]", nag_enum_arg, _countof(nag_enum_arg));
#else
  scanf("%99s%*[^\n]", nag_enum_arg);
#endif
  weight = (Nag_SparseNsym_Weight) nag_enum_name_to_value(nag_enum_arg);

#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%*[^\n]", &iterm);
#else
  scanf("%"NAG_IFMT"%*[^\n]", &iterm);
#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
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%*[^\n]", &monit);
#else
  scanf("%"NAG_IFMT"%*[^\n]", &monit);
#endif

  /* Read the parameters for the preconditioner */
#ifdef _WIN32
  scanf_s("%"NAG_IFMT"%lf%*[^\n]", &lfill, &dtol);
#else
  scanf("%"NAG_IFMT"%lf%*[^\n]", &lfill, &dtol);
#endif
#ifdef _WIN32
  scanf_s("%99s%*[^\n]", nag_enum_arg, _countof(nag_enum_arg));
#else
  scanf("%99s%*[^\n]", nag_enum_arg);
#endif
  milu = (Nag_SparseNsym_Fact) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
  scanf_s("%99s%*[^\n]", nag_enum_arg, _countof(nag_enum_arg));
#else
  scanf("%99s%*[^\n]", nag_enum_arg);
#endif
  pstrat = (Nag_SparseNsym_Piv) nag_enum_name_to_value(nag_enum_arg);

  /* Read the non-zero elements of 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 right-hand side vector B and initial approximate solution */
  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
  for (i = 0; i < n; i++)
#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_nherm_fac (f11dnc)
   * Incomplete LU factorization (non-hermitian)
   */
  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;
    }

  /* Initialize the solver using nag_sparse_nherm_basic_setup (f11brc)
   * Complex sparse non-Hermitian linear systems, setup routine
   */
  anorm = 0.0;
  sigmax = 0.0;
  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 = 2;
      goto END;
    }

  /* Call solver repeatedly to solve the equations.
   * Note that the arrays B and X are overwritten on final exit,
   * X will contain the solution and B the residual vector
   */
  irevcm = 0;
  lwreq = lwork;
  /* First call to nag_sparse_nherm_basic_solver (f11bsc)
   * Complex sparse non-Hermitian linear systems, solver routine
   * preconditioned RGMRES, CGS, Bi-CGSTAB or TFQMR method
   */
  nag_sparse_nherm_basic_solver(&irevcm, x, b, wgt, work, lwreq, &fail);
  while (irevcm != 4) {
    switch (irevcm) {
    case -1:
      /* nag_sparse_nherm_matvec (f11xnc)
       * Complex sparse non-Hermitian matrix vector multiply
       */
      trans = Nag_ConjTrans;
      nag_sparse_nherm_matvec(trans, n, nnz, a, irow, icol, check, x, b,
                              &fail1);
      break;
    case 1:
      trans = Nag_NoTrans;
      nag_sparse_nherm_matvec(trans, n, nnz, a, irow, icol, check, x, b,
                              &fail1);
```

```
      break;
    case 2:
      /* nag_sparse_nherm_precon_ilu_solve (f11dpc)
       * Solution of complex linear system involving incomplete LU
       * preconditioning matrix
       */
      trans = Nag_NoTrans;
      nag_sparse_nherm_precon_ilu_solve(trans, n, a, la, irow, icol, ipivp,
                                        ipivq, istr, idiag, check, x, b,
                                        &fail1);
      break;
    case 3:
      /* nag_sparse_nherm_basic_diagnostic (f11btc)
       * Complex sparse non-Hermitian linear systems, diagnostic routine
       */
      nag_sparse_nherm_basic_diagnostic(&itn, &stplhs, &stprhs, &anorm,
                                        &sigmax, work, lwreq, &fail1);
      printf("\nMonitoring at iteration no.%4"NAG_IFMT"\n", itn);
      printf("residual norm%14.4e\n\n", stplhs);
      printf(" Current Solution vector\n");
      for (i = 0; i < n; i++)
        printf(" (%13.4e, %13.4e)\n", x[i].re, x[i].im);
      printf("\n Current Residual vector\n");
      for (i = 0; i < n; i++)
        printf(" (%13.4e, %13.4e)\n", b[i].re, b[i].im);
      printf("\n");
    }
    if (fail1.code != NE_NOERROR) irevcm = 6;
    /* Next call to nag_sparse_nherm_basic_solver (f11bsc) */
    nag_sparse_nherm_basic_solver(&irevcm, x, b, wgt, work, lwreq, &fail);
  }
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_sparse_nherm_basic_solver (f11bsc)\n%s\n",
             fail.message);
      exit_status = 3;
      goto END;
    }

  /* Obtain information about the computation using
   * nag_sparse_nherm_basic_diagnostic (f11btc).
   */
  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 = 4;
      goto END;
    }
  /* Print the output data */
  printf("Final Results\n");
  printf("Number of iterations for convergence:     %5"NAG_IFMT"\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  Solution vector\n");
  for (i = 0; i < n; i++)
    printf(" (%13.4e, %13.4e)\n", x[i].re, x[i].im);
  printf("\n  Residual vector\n");
  for (i = 0; i < n; i++)
    printf(" (%13.4e, %13.4e)\n", b[i].re, b[i].im);
  printf("\n");
 END:

  NAG_FREE(a);
  NAG_FREE(b);
  NAG_FREE(work);
  NAG_FREE(x);
```

```
  NAG_FREE(wgt);
  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_basic_setup (f11brc) Example Program Data
  8                          : n
  24                         : nnz
  Nag_SparseNsym_TFQMR       : method
  Nag_SparseNsym_Prec        : precon
  Nag_OneNorm                : norm
  Nag_SparseNsym_UnWeighted  : weight
  1                          : iterm
  1    1.0e-8   20           : m, tol, maxitn
  2                          : monit
  0    0.0                   : lfill, dtol
  Nag_SparseNsym_UnModFact   : milu
  Nag_SparseNsym_CompletePiv : pstrat
 (  2.0,  1.0)   1    1
 ( -1.0,  1.0)   1    4
 (  1.0, -3.0)   1    8
 (  4.0,  7.0)   2    1
 ( -3.0,  0.0)   2    2
 (  2.0,  4.0)   2    5
 ( -7.0, -5.0)   3    3
 (  2.0,  1.0)   3    6
 (  3.0,  2.0)   4    1
 ( -4.0,  2.0)   4    3
 (  0.0,  1.0)   4    4
 (  5.0, -3.0)   4    7
 ( -1.0,  2.0)   5    2
 (  8.0,  6.0)   5    5
 ( -3.0, -4.0)   5    7
 ( -6.0, -2.0)   6    1
 (  5.0, -2.0)   6    3
 (  2.0,  0.0)   6    6
 (  0.0, -5.0)   7    3
 ( -1.0,  5.0)   7    5
 (  6.0,  2.0)   7    7
 ( -1.0,  4.0)   8    2
 (  2.0,  0.0)   8    6
 (  3.0,  3.0)   8    8      : a(i), irow(i), icol(i), i=0,...,nnz-1
 (  7.0, 11.0)
 (  1.0, 24.0)
 (-13.0,-18.0)
 (-10.0,  3.0)
 ( 23.0, 14.0)
 ( 17.0, -7.0)
 ( 15.0, -3.0)
 ( -3.0, 20.0)              : b(i), i=0,...,n-1
 (  0.0,  0.0)
 (  0.0,  0.0)
 (  0.0,  0.0)
 (  0.0,  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_nherm_basic_setup (f11brc) Example Program Results

Monitoring at iteration no.   2
residual norm    8.2345e+01

 Current Solution vector
 (   6.9055e-01,    1.4236e+00)
 (   7.3931e-02,   -1.1880e+00)
 (   1.4778e+00,    4.7846e-01)
 (   5.6572e+00,   -3.0786e+00)
 (   1.4243e+00,   -1.1246e+00)
 (   1.0374e-01,    1.9740e+00)
 (   4.4985e-01,   -1.2715e+00)
 (   2.5704e+00,    1.7578e+00)

 Current Residual vector
 (   1.7772e+00,    4.6797e+00)
 (   1.0774e+00,    6.4600e+00)
 (  -3.2812e+00,   -1.1314e+01)
 (  -3.8698e+00,   -1.6438e+00)
 (   8.9912e+00,    1.1100e+01)
 (   9.7428e+00,   -4.6218e-01)
 (   3.1668e+00,    2.8721e+00)
 (  -1.0323e+01,    1.5837e+00)

Final Results
Number of iterations for convergence:         4
Residual norm:                                1.3396e-11
Right-hand side of termination criterion:     8.9100e-06
1-norm of matrix A:                           2.7000e+01

  Solution vector
 (   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)
 (  -4.4172e-13,    3.0000e+00)

  Residual vector
 (  -6.0396e-14,   -7.8160e-13)
 (   1.5508e-12,   -1.3642e-12)
 (   8.2245e-13,    7.0344e-13)
 (   9.8765e-13,   -2.1760e-13)
 (  -1.2541e-12,   -4.1744e-13)
 (  -7.3541e-13,    5.2669e-13)
 (   2.8422e-13,    1.6165e-13)
 (   1.1120e-12,    2.4158e-12)
```