NAG Library Function Document

nag_sparse_sym_basic_setup (f11gdc)

1 Purpose

nag_sparse_sym_basic_setup (f11gdc) is a setup function, the first in a suite of three functions for the iterative solution of a symmetric system of simultaneous linear equations. nag_sparse_sym_basic_setup (f11gdc) must be called before the iterative solver, nag_sparse_sym_basic_solver (f11gec). The third function in the suite, nag_sparse_sym_basic_diagnostic (f11gfc), can be used to return additional information about the computation.

These three functions are suitable for the solution of large sparse symmetric systems of equations.

2 Specification

```
#include <nag.h>
#include <nagf11.h>
void nag_sparse_sym_basic_setup (Nag_SparseSym_Method method,
     Nag_SparseSym_PrecType precon, Nag_SparseSym_Bisection sigcmp,
     Nag_NormType norm, Nag_SparseSym_Weight weight, Integer iterm,
     Integer n, double tol, Integer maxitn, double anorm, double sigmax,
     double sigtol, Integer maxits, Integer monit, Integer *lwreq,
     double work[], Integer lwork, NagError *fail)
```

3 Description

The suite consisting of the functions nag_sparse_sym_basic_setup (f11gdc), nag_sparse_sym_basic_solver (f11gec) and nag_sparse_sym_basic_diagnostic (f11gfc) is designed to solve the symmetric system of simultaneous linear equations $Ax = b$ of order n, where n is large and the matrix of the coefficients A is sparse.

nag_sparse_sym_basic_setup (f11gdc) is a setup function which must be called before nag_sparse_sym_basic_solver (f11gec), the iterative solver. The third function in the suite, nag_sparse_sym_basic_diagnostic (f11gfc) can be used to return additional information about the computation. One of the following methods can be used:

1. **Conjugate Gradient Method (CG)**

   For this method (see Hestenes and Stiefel (1952), Golub and Van Loan (1996), Barrett *et al.* (1994) and Dias da Cunha and Hopkins (1994)), the matrix A should ideally be positive definite. The application of the Conjugate Gradient method to indefinite matrices may lead to failure or to lack of convergence.

2. **Lanczos Method (SYMMLQ)**

   This method, based upon the algorithm SYMMLQ (see Paige and Saunders (1975) and Barrett *et al.* (1994)), is suitable for both positive definite and indefinite matrices. It is more robust than the Conjugate Gradient method but less efficient when A is positive definite.

3. **Minimum Residual Method (MINRES)**

   This method may be used when the matrix is indefinite. It seeks to reduce the norm of the residual at each iteration and often takes fewer iterations than the other methods. It does however require slightly more memory.

The CG and SYMMLQ methods start from the residual $r_0 = b - Ax_0$, where x_0 is an initial estimate for the solution (often $x_0 = 0$), and generate an orthogonal basis for the Krylov subspace span$\{A^k r_0\}$, for $k = 0, 1, \ldots$, by means of three-term recurrence relations (see Golub and Van Loan (1996)). A sequence of symmetric tridiagonal matrices $\{T_k\}$ is also generated. Here and in the following, the index k denotes

the iteration count. The resulting symmetric tridiagonal systems of equations are usually more easily solved than the original problem. A sequence of solution iterates $\{x_k\}$ is thus generated such that the sequence of the norms of the residuals $\{\|r_k\|\}$ converges to a required tolerance. Note that, in general, the convergence is not monotonic.

In exact arithmetic, after n iterations, this process is equivalent to an orthogonal reduction of A to symmetric tridiagonal form, $T_n = Q^{\mathrm{T}}AQ$; the solution x_n would thus achieve exact convergence. In finite-precision arithmetic, cancellation and round-off errors accumulate causing loss of orthogonality. These methods must therefore be viewed as genuinely iterative methods, able to converge to a solution **within a prescribed tolerance**.

The orthogonal basis is not formed explicitly in either method. The basic difference between the Conjugate Gradient and Lanczos methods lies in the method of solution of the resulting symmetric tridiagonal systems of equations: the conjugate gradient method is equivalent to carrying out an LDL^{T} (Cholesky) factorization whereas the Lanczos method (SYMMLQ) uses an LQ factorization.

Faster convergence for all the methods 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 matrix of the coefficients 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 preconditioner must be **symmetric and positive definite**, i.e., representable by $M = EE^{\mathrm{T}}$, where M is nonsingular, and such that $\bar{A} = E^{-1}AE^{-\mathrm{T}} \sim I_n$ in (1), where I_n is the identity matrix of order n. Also, we can define $\bar{r} = E^{-1}r$ and $\bar{x} = E^{\mathrm{T}}x$. 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 to solve the preconditioning equations $Mv = u$ are required, that is, explicit information about M, E or their inverses 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 both conjugate gradient and Lanczos (SYMMLQ) methods. In (2), $p = 1, \infty$ 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 matrix of the coefficients $\|A\|_1 = \|A\|_\infty$, when this is not known in advance, used in (2), 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 could be used, but any such criteria would require information about the condition number $\kappa(A)$ which is not easily obtainable.

The second termination criterion

$$\|\bar{r}_k\|_2 \leq \tau \max\big(1.0, \|b\|_2/\|r_0\|_2\big)\big(\|\bar{r}_0\|_2 + \sigma_1\big(\bar{A}\big) \times \|\Delta\bar{x}_k\|_2\big) \tag{3}$$

is available only for the Lanczos method (SYMMLQ). 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). When $\sigma_1\big(\bar{A}\big)$ is not supplied, facilities are provided for its estimation by $\sigma_1\big(\bar{A}\big) \sim \max_k \sigma_1(T_k)$. The interlacing property $\sigma_1(T_{k-1}) \leq \sigma_1(T_k)$ and Gerschgorin's theorem provide lower and upper bounds

from which $\sigma_1(T_k)$ can be easily computed by bisection. Alternatively, the less expensive estimate $\sigma_1(\bar{A}) \sim \max_k \|T_k\|_1$ can be used, where $\sigma_1(\bar{A}) \leq \|T_k\|_1$ by Gerschgorin's theorem. Note that only order of magnitude estimates are required by the termination criterion.

Termination criterion (2) is the recommended choice, despite its (small) additional costs per iteration when using the Lanczos method (SYMMLQ). 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., $\|v\|_p^{(w)} = \left\|v^{(w)}\right\|_p$, where $\left(v^{(w)}\right)_i = w_i v_i$, for $i = 1, 2, \ldots, n$. Note that the use of weights increases the computational costs.

The MINRES algorithm terminates when the norm of the residual of the preconditioned system F, $\|F\|_2 \leq \tau \times \left\|\bar{A}\right\|_2 \times \|x_k\|_2$, where $\bar{A}$ is the preconditioned matrix.

The termination criteria discussed are not robust in the presence of a non-trivial nullspace of A, i.e., when A is singular. It is then possible for $\|x_k\|_p$ to grow without limit, spuriously satisfying the termination criterion. If singularity is suspected, more robust functions can be found in Chapter e04.

The sequence of calls to the functions comprising the suite is enforced: first, the setup function nag_sparse_sym_basic_setup (f11gdc) must be called, followed by the solver nag_sparse_sym_basic_solver (f11gec). The diagnostic function nag_sparse_sym_basic_diagnostic (f11gfc) can be called either when nag_sparse_sym_basic_solver (f11gec) is carrying out a monitoring step or after nag_sparse_sym_basic_solver (f11gec) has completed its tasks. Incorrect sequencing will raise an error condition.

4 References

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

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

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

Hestenes M and Stiefel E (1952) Methods of conjugate gradients for solving linear systems *J. Res. Nat. Bur. Stand.* **49** 409–436

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

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

5 Arguments

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

On entry: the iterative method to be used.

method = Nag_SparseSym_CG
Conjugate gradient method (CG).

method = Nag_SparseSym_SYMMLQ
Lanczos method (SYMMLQ).

method = Nag_SparseSym_MINRES
Minimum residual method (MINRES).

Constraint: **method** = Nag_SparseSym_CG, Nag_SparseSym_SYMMLQ or Nag_SparseSym_MINRES.

2: **precon** – Nag_SparseSym_PrecType *Input*

On entry: determines whether preconditioning is used.

precon = Nag_SparseSym_NoPrec
No preconditioning.

precon = Nag_SparseSym_Prec
Preconditioning.

Constraint: **precon** = Nag_SparseSym_NoPrec or Nag_SparseSym_Prec.

3: **sigcmp** – Nag_SparseSym_Bisection *Input*

On entry: determines whether an estimate of $\sigma_1(\bar{A}) = \left\|E^{-1}AE^{-\mathrm{T}}\right\|_2$, the largest singular value of the preconditioned matrix of the coefficients, is to be computed using the bisection method on the sequence of tridiagonal matrices $\{T_k\}$ generated during the iteration. Note that $\bar{A} = A$ when a preconditioner is not used.

If **sigmax** > 0.0 (see below), i.e., when $\sigma_1(\bar{A})$ is supplied, the value of **sigcmp** is ignored.

sigcmp = Nag_SparseSym_Bisect
$\sigma_1(\bar{A})$ is to be computed using the bisection method.

sigcmp = Nag_SparseSym_NoBisect
The bisection method is not used.

If the termination criterion (3) is used, requiring $\sigma_1(\bar{A})$, an inexpensive estimate is computed and used (see Section 3).

It is not used if **method** = Nag_SparseSym_MINRES.

Suggested value: **sigcmp** = Nag_SparseSym_NoBisect.

Constraint: **sigcmp** = Nag_SparseSym_Bisect or Nag_SparseSym_NoBisect.

4: **norm** – Nag_NormType *Input*

On entry: if **method** = Nag_SparseSym_CG or Nag_SparseSym_SYMMLQ, **norm** defines the matrix and vector norm to be used in the termination criteria.

norm = Nag_OneNorm
Use the l_1 norm.

norm = Nag_InfNorm
Use the l_∞ norm.

norm = Nag_TwoNorm
Use the l_2 norm.

It has no effect if **method** = Nag_SparseSym_MINRES.

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.

5: **weight** – Nag_SparseSym_Weight *Input*

On entry: specifies whether a vector w of user-supplied weights is to be used in the vector norms used in the computation of 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_SparseSym_Weighted
User-supplied weights are to be used and must be supplied on initial entry to nag_sparse_sym_basic_solver (f11gec).

weight = Nag_SparseSym_UnWeighted
All weights are implicitly set equal to one. Weights do not need to be supplied on initial entry to nag_sparse_sym_basic_solver (f11gec).

It has no effect if **method** = Nag_SparseSym_MINRES.

Suggested value: **weight** = Nag_SparseSym_UnWeighted.

Constraints:

if **iterm** $= 1$, **weight** = Nag_SparseSym_Weighted or Nag_SparseSym_UnWeighted;
if **iterm** $= 2$, **weight** = Nag_SparseSym_UnWeighted.

6: **iterm** – Integer *Input*

On entry: defines the termination criterion to be used.

iterm $= 1$
Use the termination criterion defined in (2) (both conjugate gradient and Lanczos (SYMMLQ) methods).

iterm $= 2$
Use the termination criterion defined in (3) (Lanczos method (SYMMLQ) only).

It has no effect if **method** = Nag_SparseSym_MINRES.

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

Constraints:

if **method** = Nag_SparseSym_CG, **iterm** $= 1$;
if **method** = Nag_SparseSym_SYMMLQ, **iterm** $= 1$ or 2.

7: **n** – Integer *Input*

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

Constraint: **n** > 0.

8: **tol** – double *Input*

On entry: the tolerance τ for the termination criterion.

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

Otherwise $\tau = \max(\textbf{tol}, 10\epsilon, \sqrt{n}\epsilon)$ 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_sym_basic_solver (f11gec).

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

It has no effect if **method** $=$ Nag_SparseSym_MINRES.

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

11: **sigmax** – double *Input*

On entry: if **sigmax** > 0.0, the value of $\sigma_1(\bar{A}) = \left\|E^{-1}AE^{-\mathrm{T}}\right\|_2$.

If **sigmax** ≤ 0.0, $\sigma_1(\bar{A})$ is estimated by nag_sparse_sym_basic_solver (f11gec) when either **sigcmp** $=$ Nag_SparseSym_Bisect or termination criterion (3) (**iterm** $= 2$) is employed, though it will be used only in the latter case.

Otherwise, or if **method** $=$ Nag_SparseSym_MINRES, **sigmax** is not referenced.

12: **sigtol** – double *Input*

On entry: the tolerance used in assessing the convergence of the estimate of $\sigma_1(\bar{A}) = \left\|\bar{A}\right\|_2$ when the bisection method is used.

If **sigtol** ≤ 0.0, the default value **sigtol** $= 0.01$ is used. The actual value used is $\max(\textbf{sigtol}, \epsilon)$.

If **sigcmp** $=$ Nag_SparseSym_NoBisect or **sigmax** > 0.0, then **sigtol** is not referenced.

It has no effect if **method** $=$ Nag_SparseSym_MINRES.

Suggested value: **sigtol** $= 0.01$ should be sufficient in most cases.

Constraint: if **sigcmp** $=$ Nag_SparseSym_Bisect and **sigmax** ≤ 0.0, **sigtol** < 1.0.

13: **maxits** – Integer *Input*

On entry: the maximum iteration number $k =$ **maxits** for which $\sigma_1(T_k)$ is computed by bisection (see also Section 3). If **sigcmp** $=$ Nag_SparseSym_NoBisect or **sigmax** > 0.0, or if **method** $=$ Nag_SparseSym_MINRES, then **maxits** is not referenced.

Suggested value: **maxits** $= \min(10, n)$ when **sigtol** is of the order of its default value (0.01).

Constraint: if **sigcmp** $=$ Nag_SparseSym_Bisect and **sigmax** ≤ 0.0, $1 \leq$ **maxits** $\leq$ **maxitn**.

14: **monit** – Integer *Input*

On entry: if **monit** > 0, the frequency at which a monitoring step is executed by nag_sparse_sym_basic_solver (f11gec): the current solution and residual iterates will be returned by nag_sparse_sym_basic_solver (f11gec) and a call to nag_sparse_sym_basic_diagnostic (f11gfc) made possible every **monit** iterations, starting from the (**monit**)th. Otherwise, no monitoring takes place.

There are some additional computational costs involved in monitoring the solution and residual vectors when the Lanczos method (SYMMLQ) is used.

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

15: **lwreq** – Integer * *Output*

On exit: the minimum amount of workspace required by nag_sparse_sym_basic_solver (f11gec). (See also Section 5 in nag_sparse_sym_basic_solver (f11gec).)

16: **work**[**lwork**] – double *Communication Array*

On exit: the array **work** is initialized by nag_sparse_sym_basic_setup (f11gdc). It must **not** be modified before calling the next function in the suite, namely nag_sparse_sym_basic_solver (f11gec).

17: **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_sym_basic_setup (f11gdc), a larger value is required by the other functions in the suite, namely nag_sparse_sym_basic_solver (f11gec) and nag_sparse_sym_basic_diagnostic (f11gfc). The required value is as follows:

Method	**Requirements**
CG	**lwork** $= 120 + 5n + p$.
SYMMLQ	**lwork** $= 120 + 6n + p$,
MINRES	**lwork** $= 120 + 9n$,

where

$p = 2 * (\textbf{maxits} + 1)$, when an estimate of $\sigma_1(A)$ (**sigmax**) is computed;

$p = 0$, otherwise.

18: **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_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 2.3.1.2 in How to Use the NAG Library and its Documentation 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** $= \langle value \rangle$.
Constraint: if **iterm** $= 1$ and **norm** $=$ Nag_TwoNorm, **anorm** > 0.0.

On entry, **sigcmp** $=$ Nag_SparseSym_Bisect, **sigmax** ≤ 0.0 and **maxits** $= \langle value \rangle$.
Constraint: if **sigcmp** $=$ Nag_SparseSym_Bisect and **sigmax** ≤ 0.0, **maxits** ≥ 1.

On entry, **sigcmp** $=$ Nag_SparseSym_Bisect, **sigmax** ≤ 0.0, **maxits** $= \langle value \rangle$ and **maxitn** $= \langle value \rangle$.
Constraint: if **sigcmp** $=$ Nag_SparseSym_Bisect and **sigmax** ≤ 0.0, **maxits** $\leq$ or **maxitn**.

NE_ENUM_INT

On entry, **method** $= \langle value \rangle$, **weight** $= \langle value \rangle$, **norm** $= \langle value \rangle$ and **iterm** $= \langle value \rangle$.
Constraint: if **method** $=$ Nag_SparseSym_CG or **weight** $=$ Nag_SparseSym_Weighted or **norm** $\neq$ Nag_TwoNorm, **iterm** $= 1$. Otherwise, **iterm** $= 1$ or 2.

NE_ENUM_REAL_2

On entry, **sigcmp** = Nag_SparseSym_Bisect, **sigmax** ≤ 0.0 and **sigtol** = ⟨*value*⟩.
Constraint: if **sigcmp** = Nag_SparseSym_Bisect and **sigmax** ≤ 0.0, **sigtol** < 1.0.

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** $\leq$ **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 2.7.6 in How to Use the NAG Library and its Documentation for further information.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

NE_OUT_OF_SEQUENCE

nag_sparse_sym_basic_setup (f11gdc) has been called out of sequence: either nag_sparse_sym_basic_setup (f11gdc) has been called twice or nag_sparse_sym_basic_solver (f11gec) has not terminated its current task.

NE_REAL

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

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_sparse_sym_basic_setup (f11gdc) is not threaded in any implementation.

9 Further Comments

When $\sigma_1(\bar{A})$ is not supplied (**sigmax** ≤ 0.0) but it is required, it is estimated by nag_sparse_sym_basic_solver (f11gec) using either of the two methods described in Section 3, as specified by the argument **sigcmp**. In particular, if **sigcmp** = Nag_SparseSym_Bisect, then the computation of $\sigma_1(\bar{A})$ is deemed to have converged when the differences between three successive values of $\sigma_1(T_k)$ differ, in a relative sense, by less than the tolerance **sigtol**, i.e., when

$$\max\left(\frac{\left|\sigma_1^{(k)}-\sigma_1^{(k-1)}\right|}{\sigma_1^{(k)}},\frac{\left|\sigma_1^{(k)}-\sigma_1^{(k-2)}\right|}{\sigma_1^{(k)}}\right)\leq \textbf{sigtol}.$$

The computation of $\sigma_1\left(\bar{A}\right)$ is also terminated when the iteration count exceeds the maximum value allowed, i.e., $k \geq$ **maxits**.

Bisection is increasingly expensive with increasing iteration count. A reasonably large value of **sigtol**, of the order of the suggested value, is recommended and an excessive value of **maxits** should be avoided. Under these conditions, $\sigma_1\left(\bar{A}\right)$ usually converges within very few iterations.

10 Example

This example solves a symmetric system of simultaneous linear equations using the conjugate gradient method, where the matrix of the coefficients A, has a random sparsity pattern. An incomplete Cholesky preconditioner is used (nag_sparse_sym_chol_fac (f11jac) and nag_sparse_sym_precon_ichol_solve (f11jbc)).

10.1 Program Text

```
/* nag_sparse_sym_basic_setup (f11gdc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagf11.h>
int main(void)
{
  /* Scalars */
  Integer exit_status = 0;
  double anorm, dscale, dtol, sigerr, sigmax, sigtol, stplhs, stprhs, tol;
  Integer i, irevcm, iterm, itn, its, la, lfill,
         lwork, lwreq, maxitn, maxits, monit, n, nnz, nnzc, npivm;
  /* Arrays */
  char nag_enum_arg[100];
  double *a = 0, *b = 0, *wgt = 0, *work = 0, *x = 0;
  Integer *icol = 0, *ipiv = 0, *irow = 0, *istr = 0;
  /* NAG types */
  Nag_SparseSym_Piv pstrat;
  Nag_SparseSym_Method method;
  Nag_SparseSym_PrecType precon;
  Nag_NormType norm;
  Nag_SparseSym_Weight weight;
  Nag_SparseSym_Fact mic;
  Nag_Sparse_Comm comm;
  Nag_SparseSym_Bisection sigcmp;
  NagError fail, fail1;

  INIT_FAIL(fail);
  INIT_FAIL(fail1);

  printf("nag_sparse_sym_basic_setup (f11gdc) 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
  /* Read or initialize the parameters for the iterative solver */
  /* nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
#ifdef _WIN32
  scanf_s("%99s%*[^\n]", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
  scanf("%99s%*[^\n]", nag_enum_arg);
#endif
  method = (Nag_SparseSym_Method) nag_enum_name_to_value(nag_enum_arg);

#ifdef _WIN32
  scanf_s("%99s%*[^\n]", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
  scanf("%99s%*[^\n]", nag_enum_arg);
#endif
  precon = (Nag_SparseSym_PrecType) nag_enum_name_to_value(nag_enum_arg);

#ifdef _WIN32
  scanf_s("%99s%*[^\n]", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
  scanf("%99s%*[^\n]", nag_enum_arg);
#endif
  sigcmp = (Nag_SparseSym_Bisection) nag_enum_name_to_value(nag_enum_arg);

#ifdef _WIN32
  scanf_s("%99s%*[^\n]", nag_enum_arg, (unsigned)_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, (unsigned)_countof(nag_enum_arg));
#else
  scanf("%99s%*[^\n]", nag_enum_arg);
#endif
  weight = (Nag_SparseSym_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("%lf%" NAG_IFMT "%*[^\n]", &tol, &maxitn);
#else
  scanf("%lf%" NAG_IFMT "%*[^\n]", &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, (unsigned)_countof(nag_enum_arg));
#else
  scanf("%99s%*[^\n]", nag_enum_arg);
#endif
  mic = (Nag_SparseSym_Fact) nag_enum_name_to_value(nag_enum_arg);
#ifdef _WIN32
  scanf_s("%lf%*[^\n]", &dscale);
#else
  scanf("%lf%*[^\n]", &dscale);
#endif
#ifdef _WIN32
  scanf_s("%99s%*[^\n]", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
  scanf("%99s%*[^\n]", nag_enum_arg);
#endif
  pstrat = (Nag_SparseSym_Piv) nag_enum_name_to_value(nag_enum_arg);

  la = 2 * nnz;
  lwork = 120;
  if (!(a = NAG_ALLOC(la, double)) ||
      !(b = NAG_ALLOC(n, double)) ||
      !(wgt = NAG_ALLOC(n, double)) ||
      !(work = NAG_ALLOC(lwork, double)) ||
      !(x = NAG_ALLOC(n, double)) ||
      !(icol = NAG_ALLOC(la, Integer)) ||
      !(ipiv = 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 the nonzero elements of the matrix A */
  for (i = 0; i < nnz; i++)
#ifdef _WIN32
    scanf_s("%lf%" NAG_IFMT "%" NAG_IFMT "%*[^\n]", &a[i], &irow[i],
            &icol[i]);
#else
    scanf("%lf%" NAG_IFMT "%" NAG_IFMT "%*[^\n]", &a[i], &irow[i], &icol[i]);
#endif

  /* Read right-hand side vector b and initial approximate solution x */
#ifdef _WIN32
  for (i = 0; i < n; i++)
    scanf_s("%lf", &b[i]);
#else
  for (i = 0; i < n; i++)
    scanf("%lf", &b[i]);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n]");
#else
  scanf("%*[^\n]");
#endif
#ifdef _WIN32
  for (i = 0; i < n; i++)
    scanf_s("%lf", &x[i]);
#else
  for (i = 0; i < n; i++)
    scanf("%lf", &x[i]);
#endif

  printf("\nSolve a system of linear equations using the ");
  if (method == Nag_SparseSym_CG)
    printf("conjugate gradient method (CG)\n");
  else if (method == Nag_SparseSym_SYMMLQ)
    printf("Lanczos method (SYMMLQ)\n");
  else if (method == Nag_SparseSym_MINRES)
```

```
  printf("minimum residual method (MINRES)\n");

/* Calculate incomplete Cholesky factorization as preconditioner using
 * nag_sparse_sym_chol_fac (f11jac).
 * Incomplete Cholesky factorization (symmetric)
 */
nag_sparse_sym_chol_fac(n, nnz, &a, &la, &irow, &icol, lfill, dtol, mic,
                        dscale, pstrat, ipiv, istr, &nnzc, &npivm, &comm,
                        &fail);
if (fail.code != NE_NOERROR) {
  printf("Error from nag_sparse_sym_chol_fac (f11jac)\n%s\n", fail.message);
  exit_status = 1;
  goto END;
}

/* Initialize the solver using nag_sparse_sym_basic_setup (f11gdc).
 * Real sparse symmetric linear systems, setup for f11gec
 */
anorm = 0.0;
sigmax = 0.0;
sigtol = 0.01;
maxits = n;
while (1) {
  nag_sparse_sym_basic_setup(method, precon, sigcmp, norm, weight, iterm, n,
                             tol, maxitn, anorm, sigmax, sigtol, maxits,
                             monit, &lwreq, work, lwork, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_sparse_sym_basic_setup (f11gdc)\n%s\n",
           fail.message);
    exit_status = 2;
    goto END;
  }
  if (lwork >= lwreq)
    break;
  else {
    NAG_FREE(work);
    lwork = lwreq;
    work = NAG_ALLOC(lwreq, double);
  }
}

/* Call 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;
/* First call of nag_sparse_sym_basic_solver (f11gec).
 * Real sparse symmetric linear systems, preconditioned conjugate
 * gradient or Lanczos
 */
nag_sparse_sym_basic_solver(&irevcm, x, b, wgt, work, lwork, &fail);
while (irevcm != 4) {
  switch (irevcm) {
  case 1:
    /* nag_sparse_sym_matvec (f11xec)
     * Real sparse symmetric matrix vector multiply
     */
    nag_sparse_sym_matvec(n, nnz, a, irow, icol, Nag_SparseSym_NoCheck,
                          x, b, &fail1);
    break;
  case 2:
    /* nag_sparse_sym_precon_ichol_solve (f11jbc).
     * Solution of linear system involving incomplete Cholesky
     * preconditioning matrix generated by f11jac
     */
    nag_sparse_sym_precon_ichol_solve(n, a, la, irow, icol, ipiv, istr,
                                      Nag_SparseSym_NoCheck, x, b, &fail1);
    break;
  case 3:
    /* nag_sparse_sym_basic_diagnostic (f11gfc).
     * Real sparse symmetric linear systems, diagnostic for f11gec
```

```
       */
      nag_sparse_sym_basic_diagnostic(&itn, &stplhs, &stprhs, &anorm,
                                      &sigmax, &its, &sigerr, work, lwork,
                                      &fail1);
      printf("\nMonitoring at iteration no.%4" NAG_IFMT "\n", itn);
      printf("residual norm: %14.4e\n", stplhs);
      printf("  Solution vector  Residual vector\n");
      for (i = 0; i < n; i++)
        printf("%16.4e%16.4e\n", x[i], b[i]);
      printf("\n");
    }
    if (fail1.code != NE_NOERROR)
      irevcm = 6;
    /* Next call of nag_sparse_sym_basic_solver (f11gec). */
    nag_sparse_sym_basic_solver(&irevcm, x, b, wgt, work, lwork, &fail);
  }
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_sparse_sym_basic_solver (f11gec).\n%s\n",
           fail.message);
    exit_status = 3;
    goto END;
  }

  /* Obtain information about the computation using
   * nag_sparse_sym_basic_diagnostic (f11gfc).
   * Real sparse symmetric linear systems, diagnostic for solver.
   */
  nag_sparse_sym_basic_diagnostic(&itn, &stplhs, &stprhs, &anorm, &sigmax,
                                  &its, &sigerr, work, lwork, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_sparse_sym_basic_diagnostic (f11gfc)\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);
  printf("Largest singular value of A_bar:          %14.4e\n\n", sigmax);
  /* Output x */
  printf("%14s%14s\n", "Solution", "Residual");
  for (i = 0; i < n; i++)
    printf("%14.4e%14.4e\n", x[i], b[i]);
  printf("\n");

END:

  NAG_FREE(a);
  NAG_FREE(b);
  NAG_FREE(wgt);
  NAG_FREE(work);
  NAG_FREE(x);
  NAG_FREE(icol);
  NAG_FREE(ipiv);
  NAG_FREE(irow);
  NAG_FREE(istr);
  return exit_status;
}
```

10.2 Program Data

```
nag_sparse_sym_basic_setup (f11gdc) Example Program Data
  7                         : n
 16                         : nnz
   Nag_SparseSym_CG         : method
   Nag_SparseSym_Prec       : precon
   Nag_SparseSym_Bisect     : sigcmp
```

```
   Nag_OneNorm              : norm
   Nag_SparseSym_UnWeighted : weight
  1                         : iterm
  1.0e-6   20               : tol, maxitn
  2                         : monit
  0  0.0                   : lfill, dtol
   Nag_SparseSym_UnModFact  : mic
  0.0                       : dscale
   Nag_SparseSym_MarkPiv    : pstrat
  4.   1    1
  1.   2    1
  5.   2    2
  2.   3    3
  2.   4    2
  3.   4    4
 -1.   5    1
  1.   5    4
  4.   5    5
  1.   6    2
 -2.   6    5
  3.   6    6
  2.   7    1
 -1.   7    2
 -2.   7    3
  5.   7    7               : a[i], irow[i], icol[i], i=0,...,nnz-1
 15.  18.  -8.  21.
 11.  10.  29.              : b[i], i=0,...,n-1
  0.   0.   0.   0.
  0.   0.   0.              : x[i], i=0,...,n-1
```

10.3 Program Results

```
nag_sparse_sym_basic_setup (f11gdc) Example Program Results

Solve a system of linear equations using the conjugate gradient method (CG)

Monitoring at iteration no.   2
residual norm:     1.9938e+00
  Solution vector  Residual vector
      9.6320e-01     -2.2960e-01
      1.9934e+00      2.2254e-01
      3.0583e+00      9.5827e-02
      4.1453e+00     -2.5155e-01
      4.8289e+00     -1.7160e-01
      5.6630e+00      6.7533e-01
      7.1062e+00     -3.4737e-01


Monitoring at iteration no.   4
residual norm:     6.6574e-03
  Solution vector  Residual vector
      9.9940e-01     -1.0551e-03
      2.0011e+00     -2.4675e-03
      3.0008e+00     -1.7116e-05
      3.9996e+00      4.4929e-05
      4.9991e+00      2.1359e-03
      5.9993e+00     -8.7482e-04
      7.0007e+00      6.2045e-05

Final Results
Number of iterations for convergence:         5
Residual norm:                                2.0428e-14
Right-hand side of termination criterion:     3.9200e-04
1-norm of matrix A:                           1.0000e+01
Largest singular value of A_bar:              1.3596e+00

      Solution      Residual
    1.0000e+00    0.0000e+00
    2.0000e+00    0.0000e+00
    3.0000e+00   -2.6645e-15
```

```
    4.0000e+00   -3.5527e-15
    5.0000e+00   -5.3291e-15
    6.0000e+00    1.7764e-15
    7.0000e+00    7.1054e-15
```