Module 5.5: nag_sym_bnd_lin_sys

Symmetric Banded Systems of Linear Equations

`nag_sym_bnd_lin_sys` provides a procedure for solving real symmetric or complex Hermitian *banded* systems of linear equations with one or many right-hand sides:

$Ax = b$ or $AX = B$,

where the matrix A is *positive definite*. It also provides procedures for factorizing A and solving a system of equations when the matrix A has already been factorized.

Contents

Introduction

1 Notation and Background

We use the following notation for a system of linear equations:

$Ax = b$, if there is one right-hand side b;

$AX = B$, if there are many right-hand sides (the columns of the matrix B).

In this module, the matrix A (the *coefficient matrix*) is assumed to be *real symmetric* or *complex Hermitian*, *positive definite*, and *banded*. The procedures take advantage of these properties in order to economize on the work and storage required.

If the matrix A is real symmetric or complex Hermitian but *not* positive definite, it is not possible to preserve the bandwidth while maintaining numerical stability. The system must be treated either as a general banded system (see module `nag_gen_bnd_lin_sys`) or as a full symmetric or Hermitian system (see module `nag_sym_lin_sys`).

The module provides options to return *forward* or *backward error bounds* on the computed solution. It also provides options to evaluate the *determinant* of A and to estimate the *condition number* of A, which is a measure of the sensitivity of the computed solution to perturbations of the original data or to rounding errors in the computation. For more details on error analysis, see the Chapter Introduction.

To solve the system of equations the first step is to factorize A, using the *Cholesky* factorization. The system of equations can then be solved by forward and backward substitution.

2 Choice of Procedures

The procedure `nag_sym_bnd_lin_sol` should be suitable for most purposes; it performs the factorization of A and solves the system of equations in a single call. It also has options to estimate the condition number of A, and to return forward and backward error bounds on the computed solution.

The module also provides lower-level procedures which perform the two computational steps in the solution process:

`nag_sym_bnd_lin_fac` computes a factorization of A, with options to evaluate the determinant and to estimate the condition number;

`nag_sym_bnd_lin_sol_fac` solves the system of equations, assuming that A has already been factorized by a call to `nag_sym_bnd_lin_fac`. It has options to return forward and backward error bounds on the solution.

These lower-level procedures are intended for more experienced users. For example, they enable a factorization computed by `nag_sym_bnd_lin_fac` to be reused several times in repeated calls to `nag_sym_bnd_lin_sol_fac`.

3 Storage of Matrices

The procedures in this module use the following storage scheme for the symmetric or Hermitian band matrix A with k super-diagonals or sub-diagonals:

- If `uplo` = `'u'` or `'U'`, a_{ij} is stored in `a`$(k+i-j+1,j)$, for $\max(j-k,1) \leq i \leq j$.
- If `uplo` = `'l'` or `'L'`, a_{ij} is stored in `a`$(i-j+1,j)$, for $j \leq i \leq \min(j+k,n)$.

For example

`uplo`	Hermitian band matrix A	Band storage in array `a`
`'u'` or `'U'`	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & & \\ \overline{a}_{12} & a_{22} & a_{23} & a_{24} & \\ \overline{a}_{13} & \overline{a}_{23} & a_{33} & a_{34} & a_{35} \\ & \overline{a}_{24} & \overline{a}_{34} & a_{44} & a_{45} \\ & & \overline{a}_{35} & \overline{a}_{45} & a_{55} \end{pmatrix}$	$\begin{matrix} * & * & a_{13} & a_{24} & a_{35} \\ * & a_{12} & a_{23} & a_{34} & a_{45} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \end{matrix}$
`'l'` or `'L'`	$\begin{pmatrix} a_{11} & \overline{a}_{21} & \overline{a}_{31} & & \\ a_{21} & a_{22} & \overline{a}_{32} & \overline{a}_{42} & \\ a_{31} & a_{32} & a_{33} & \overline{a}_{43} & \overline{a}_{53} \\ & a_{42} & a_{43} & a_{44} & \overline{a}_{54} \\ & & a_{53} & a_{54} & a_{55} \end{pmatrix}$	$\begin{matrix} a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{21} & a_{32} & a_{43} & a_{54} & * \\ a_{31} & a_{42} & a_{53} & * & * \end{matrix}$

Procedure: nag_sym_bnd_lin_sol

1 Description

`nag_sym_bnd_lin_sol` is a generic procedure which computes the solution of a system of linear equations, with one or many right-hand sides, where the matrix of coefficients is *banded*, and

real symmetric positive definite, or

complex Hermitian positive definite.

We write:

$Ax = b$, if there is one right-hand side b;

$AX = B$, if there are many right-hand sides (the columns of the matrix B).

The procedure also has options to return an estimate of the *condition number* of A, and *forward* and *backward error bounds* for the computed solution or solutions. See the Chapter Introduction for an explanation of these terms. If error bounds are requested, the procedure performs iterative refinement of the computed solution in order to guarantee a small backward error.

2 Usage

```
USE nag_sym_bnd_lin_sys
CALL nag_sym_bnd_lin_sol(uplo, a, b  [, optional arguments])
```

2.1 Interfaces

Distinct interfaces are provided for each of the 4 combinations of the following cases:

Real / complex data

Real data: `a` and `b` are of type real(kind=*wp*).

Complex data: `a` and `b` are of type complex(kind=*wp*).

One / many right-hand sides

One r.h.s.: `b` is a rank-1 array, and the optional arguments `bwd_err` and `fwd_err` are scalars.

Many r.h.s.: `b` is a rank-2 array, and the optional arguments `bwd_err` and `fwd_err` are rank-1 arrays.

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '`x`(n)' is used in the argument descriptions to specify that the array `x` must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

n — the order of the band matrix A

$k \geq 0$ — the number of super-diagonals or sub-diagonals in the band matrix A

r — the number of right-hand sides

3.1 Mandatory Arguments

uplo — character(len=1), intent(in)

Input: specifies whether the upper or lower triangle of A is supplied, and whether the factorization involves an upper triangular matrix U or a lower triangular matrix L.

If `uplo` = `'u'` or `'U'`, the upper triangle is supplied, and is overwritten by an upper triangular factor U;

if `uplo` = `'l'` or `'L'`, the lower triangle is supplied, and is overwritten by a lower triangular factor L.

Constraints: `uplo` = `'u'`, `'U'`, `'l'` or `'L'`.

a($k+1, n$) — real(kind=*wp*) / complex(kind=*wp*), intent(inout)

Input: the band matrix A.

If `uplo` = `'u'`, the elements of the upper triangle of A within the band must be stored, with a_{ij} in `a`$(k+i-j+1, j)$ for $\max(j-k, 1) \le i \le j$;

if `uplo` = `'l'`, the elements of the lower triangle of A within the band must be stored, with a_{ij} in `a`$(i-j+1, j)$ for $j \le i \le \min(j+k, n)$.

Output: the supplied triangle of A is overwritten by the Cholesky factor U or L as specified by `uplo`, using the same storage format as described above.

Constraints: if A is complex Hermitian, its diagonal elements must have zero imaginary parts.

b(n) / **b**(n, r) — real(kind=*wp*) / complex(kind=*wp*), intent(inout)

Input: the right-hand side vector b or matrix B.

Output: overwritten on exit by the solution vector x or matrix X.

Constraints: `b` must be of the same type as `a`.

Note: if optional error bounds are requested then the solution returned is that computed by iterative refinement.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

bwd_err / **bwd_err**(r) — real(kind=*wp*), intent(out), optional

Output: if `bwd_err` is a scalar, it returns the component-wise backward error bound for the single solution vector x. Otherwise, `bwd_err`(i) returns the component-wise backward error bound for the ith solution vector, returned in the ith column of `b`, for $i = 1, 2, \ldots, r$.

Constraints: if `b` has rank 1, `bwd_err` must be a scalar; if `b` has rank 2, `bwd_err` must be a rank-1 array.

fwd_err / **fwd_err**(r) — real(kind=*wp*), intent(out), optional

Output: if `fwd_err` is a scalar, it returns an estimated bound for the forward error in the single solution vector x. Otherwise, `fwd_err`(i) returns an estimated bound for the forward error in the ith solution vector, returned in the ith column of `b`, for $i = 1, 2, \ldots, r$.

Constraints: if `b` has rank 1, `fwd_err` must be a scalar; if `b` has rank 2, `fwd_err` must be a rank-1 array.

rcond — real(kind=*wp*), intent(out), optional

Output: an estimate of the reciprocal of the condition number of A, $\kappa_\infty(A)(= \kappa_1(A)$ for A symmetric or Hermitian). `rcond` is set to zero if exact singularity is detected or the estimate underflows. If `rcond` is less than `EPSILON(1.0_`*wp*`)`, then A is singular to working precision.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
302	An array argument has an invalid shape.
303	Array arguments have inconsistent shapes.
320	The procedure was unable to allocate enough memory.

Failures (error%level = 2):

error%code	Description
201	Matrix not positive definite. The Cholesky factorization cannot be completed. Either A is close to singularity, or it has at least one negative eigenvalue. No solutions or error bounds are computed.

Warnings (error%level = 1):

error%code	Description
101	Approximately singular matrix. The estimate of the reciprocal condition number (returned in `rcond` if present) is less than `EPSILON(1.0_`*wp*`)`. The matrix is singular to working precision, and it is likely that the computed solution returned in `b` has no accuracy at all. You should examine the forward error bounds returned in `fwd_err`, if present.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 1 of this module document.

6 Further Comments

6.1 Algorithmic Detail

The procedure first calls `nag_sym_bnd_lin_fac` to factorize A, and to estimate the condition number. It then calls `nag_sym_bnd_lin_sol_fac` to compute the solution to the system of equations, and, if required, the error bounds. See the documents for those procedures for more details, and Chapter 4 of Golub and Van Loan [2] for background. The algorithms are derived from LAPACK (see Anderson *et al.* [1]).

6.2 Accuracy

The accuracy of the computed solution is given by the forward and backward error bounds which are returned in the optional arguments `fwd_err` and `bwd_err`.

The backward error bound `bwd_err` is rigorous; the forward error bound `fwd_err` is an estimate, but is almost always satisfied.

The condition number $\kappa_\infty(A)$ gives a general measure of the *sensitivity* of the solution of $Ax = b$, either to uncertainties in the data or to rounding errors in the computation. An estimate of the reciprocal of $\kappa_\infty(A)$ is returned in the optional argument `rcond`. However, forward error bounds derived using this condition number may be more pessimistic than the bounds returned in `fwd_err`, if present.

6.3 Timing

The time taken is roughly proportional to $n(k+1)^2$, assuming $n \gg k$ and there are only a few right-hand sides. The time taken for complex data is about 4 times as long as that for real data.

Procedure: nag_sym_bnd_lin_fac

1 Description

`nag_sym_bnd_lin_fac` is a generic procedure which factorizes a real symmetric or complex Hermitian positive definite band matrix A of order n. The factorization is written as:

$A = U^T U$ or $A = LL^T$, if A is real symmetric;

$A = U^H U$ or $A = LL^H$, if A is complex Hermitian;

where U is upper triangular, L is lower triangular, and both are banded, with the same number of super-diagonals or sub-diagonals as A.

This procedure can also return the determinant of A and an estimate of the condition number $\kappa_\infty(A)$ $(= \kappa_1(A))$.

2 Usage

```
USE nag_sym_bnd_lin_sys
```

`CALL nag_sym_bnd_lin_fac(uplo, a [,` *optional arguments*`])`

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '$\mathbf{x}(n)$' is used in the argument descriptions to specify that the array `x` must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

n — the order of the band matrix A

$k \geq 0$ — the number of super-diagonals or sub-diagonals in the band matrix A

3.1 Mandatory Arguments

uplo — character(len=1), intent(in)

Input: specifies whether the upper or lower triangle of A is supplied, and whether the factorization involves an upper triangular matrix U or a lower triangular matrix L.

If `uplo` = `'u'` or `'U'`, the upper triangle is supplied, and is overwritten by an upper triangular factor U;

if `uplo` = `'l'` or `'L'`, the lower triangle is supplied, and is overwritten by a lower triangular factor L.

Constraints: `uplo` = `'u'`, `'U'`, `'l'` or `'L'`.

a$(k+1, n)$ — real(kind=*wp*) / complex(kind=*wp*), intent(inout)

Input: the band matrix A.

If `uplo` = `'u'`, the elements of the upper triangle of A within the band must be stored, with a_{ij} in $\mathtt{a}(k+i-j+1, j)$ for $\max(j-k, 1) \leq i \leq j$;

if `uplo` = `'l'`, the elements of the lower triangle of A within the band must be stored, with a_{ij} in $\mathtt{a}(i-j+1, j)$ for $j \leq i \leq \min(j+k, n)$.

Output: the supplied triangle of A is overwritten by the Cholesky factor U or L as specified by `uplo`, using the same storage format as described above.

Constraints: if A is complex Hermitian, its diagonal elements must have zero imaginary parts.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

rcond — real(kind=*wp*), intent(out), optional

Output: an estimate of the reciprocal of the condition number of A, $\kappa_\infty(A)(=\kappa_1(A)$ for A symmetric or Hermitian). `rcond` is set to zero if exact singularity is detected or the estimate underflows. If `rcond` is less than `EPSILON(1.0_`*wp*`)`, then A is singular to working precision.

det_frac — real(kind=*wp*), intent(out), optional
det_exp — integer, intent(out), optional

Output: `det_frac` returns the fractional part f, and `det_exp` returns the exponent e, of the determinant of A expressed as $f.b^e$, where b is the base of the representation of the floating point numbers (given by `RADIX(1.0_`*wp*`)`), or as `SCALE` (`det_frac`,`det_exp`). The determinant is returned in this form to avoid the risk of overflow or underflow.

Constraints: if either `det_frac` or `det_exp` is present the other must also be present.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
302	An array argument has an invalid shape.
305	Invalid absence of an optional argument.
320	The procedure was unable to allocate enough memory.

Failures (error%level = 2):

error%code	Description
201	Matrix not positive definite. This error can only occur if the Cholesky factorization cannot be completed. Either A is close to singularity, or it has at least one negative eigenvalue. If the factorization is used to solve a system of linear equations, an error will occur.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 2 of this module document.

6 Further Comments

6.1 Algorithmic Detail

The procedure performs a banded Cholesky factorization of A:

$A = U^H U$, with U upper triangular and banded, if `uplo` = `'u'`;

$A = LL^H$, with L lower triangular and banded, if `uplo` = `'l'`.

See Section 4.3.6 of Golub and Van Loan [2].

To estimate the condition number $\kappa_\infty(A)$ $(= \kappa_1(A) = \|A\|_1\|A^{-1}\|_1)$ the procedure first computes $\|A\|_1$ directly, and then uses Higham's modification of Hager's method (see Higham [3]) to estimate $\|A^{-1}\|_1$. The procedure returns the reciprocal $\rho = 1/\kappa_\infty(A)$, rather than $\kappa_\infty(A)$ itself.

The algorithms are derived from LAPACK (see Anderson *et al.* [1]).

6.2 Accuracy

If `uplo` = `'u'`, the computed factor U is the exact factor of a perturbed matrix $A + E$, such that

$$|E| \le c(k+1)\epsilon|U^H|\,|U|,$$

where $c(k+1)$ is a modest linear function of $k+1$, and ϵ = `EPSILON(1.0_wp)`. If `uplo` = `'l'`, a similar statement holds for the computed factor L. It follows that in both cases $|e_{ij}| \le c(k+1)\epsilon\sqrt{a_{ii}a_{jj}}$.

The computed estimate **rcond** is never less than the true value ρ, and in practice is nearly always less than 10ρ (although examples can be constructed where the computed estimate is much larger).

Since $\rho = 1/\kappa(A)$, this means that the procedure never overestimates the condition number, and hardly ever underestimates it by more than a factor of 10.

6.3 Timing

The total number of floating-point operations required is roughly $n(k+1)^2$ for real A, and $4n(k+1)^2$ for complex A, assuming $n \gg k$.

Estimating the condition number involves solving a number of systems of linear equations with A or A^T as the coefficient matrix; the number is usually 4 or 5 and never more than 11. Each solution involves approximately $4nk$ floating-point operations if A is real, or $16nk$ if A is complex.

Procedure: nag_sym_bnd_lin_sol_fac

1 Description

nag_sym_bnd_lin_sol_fac is a generic procedure which computes the solution of a real symmetric or complex Hermitian positive definite banded system of linear equations, with one or many right-hand sides, assuming that the coefficient matrix has already been factorized by nag_sym_bnd_lin_fac.

We write:

$Ax = b$, if there is one right-hand side b;

$AX = B$, if there are many right-hand sides (the columns of the matrix B).

The procedure also has options to return *forward* and *backward error bounds* for the computed solution or solutions.

2 Usage

```
USE nag_sym_bnd_lin_sys
CALL nag_sym_bnd_lin_sol_fac(uplo, a_fac, b  [, optional arguments])
```

2.1 Interfaces

Distinct interfaces are provided for each of the 4 combinations of the following cases:

Real / complex data

Real data: a_fac, b and the optional argument a are of type real(kind=*wp*).

Complex data: a_fac, b and the optional argument a are of type complex(kind=*wp*).

One / many right-hand sides

One r.h.s.: b is a rank-1 array, and the optional arguments bwd_err and fwd_err are scalars.

Many r.h.s.: b is a rank-2 array, and the optional arguments bwd_err and fwd_err are rank-1 arrays.

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '$\mathbf{x}(n)$' is used in the argument descriptions to specify that the array x must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

n — the order of the matrix A

$k \geq 0$ — the number of super-diagonals or sub-diagonals in the band matrix A

r — the number of right-hand sides

3.1 Mandatory Arguments

uplo — character(len=1), intent(in)

Input: specifies whether the upper or lower triangle of A was supplied to nag_sym_bnd_lin_fac, and whether the factorization involves an upper triangular matrix U or a lower triangular matrix L.

If uplo = 'u' or 'U', the upper triangle was supplied, and was overwritten by an upper triangular factor U;

if uplo = 'l' or 'L', the lower triangle was supplied, and was overwritten by a lower triangular factor L.

Constraints: uplo = 'u', 'U', 'l' or 'L'.

Note: the value of uplo must be the same as in the preceding call to nag_sym_bnd_lin_fac.

a_fac($k+1, n$) — real(kind=*wp*) / complex(kind=*wp*), intent(in)

Input: the factorisation of A, as returned by nag_sym_bnd_lin_fac.

b(n) / **b**(n, r) — real(kind=*wp*) / complex(kind=*wp*), intent(inout)

Input: the right-hand side vector b or matrix B.

Output: overwritten on exit by the solution vector x or matrix X.

Constraints: b must be of the same type as a_fac.

Note: if optional error bounds are requested then the solution returned is that computed by iterative refinement.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

bwd_err / **bwd_err**(r) — real(kind=*wp*), intent(out), optional

Output: if bwd_err is a scalar, it returns the component-wise backward error bound for the single solution vector x. Otherwise, bwd_err(i) returns the component-wise backward error bound for the ith solution vector, returned in the ith column of b, for $i = 1, 2, \ldots, r$.

Constraints: if bwd_err is present, the original matrix A must be supplied in a; if b has rank 1, bwd_err must be a scalar; if b has rank 2, bwd_err must be a rank-1 array.

fwd_err / **fwd_err**(r) — real(kind=*wp*), intent(out), optional

Output: if fwd_err is a scalar, it returns an estimated bound for the forward error in the single solution vector x. Otherwise, fwd_err(i) returns an estimated bound for the forward error in the ith solution vector, returned in the ith column of b, for $i = 1, 2, \ldots, r$.

Constraints: if fwd_err is present, the original matrix A must be supplied in a; if b has rank 1, fwd_err must be a scalar; if b has rank 2, fwd_err must be a rank-1 array.

a($k+1, n$) — real(kind=*wp*) / complex(kind=*wp*), intent(in), optional

Input: the original coefficient matrix A, as supplied to nag_sym_bnd_lin_fac.

Constraints: a must be present if either bwd_err or fwd_err is present; a must be of the same type and rank as a_fac.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document nag_error_handling (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to nag_set_error before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
302	An array argument has an invalid shape.
303	Array arguments have inconsistent shapes.
305	Invalid absence of an optional argument.
320	The procedure was unable to allocate enough memory.

Failures (error%level = 2):

error%code	Description
201	Matrix not positive definite. The supplied array `a_fac` does not contain a valid Cholesky factorization, indicating that the original matrix A was not positive definite. No solutions or error bounds are computed.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 2 of this module document.

6 Further Comments

6.1 Algorithmic Detail

The solution x is computed by forward and backward substitution. If `uplo` = `'u'`, $U^H y = b$ is solved for y, and then $Ux = b$ is solved for x. A similar method is used if `uplo` = `'l'`.

If error bounds are requested (that is, `fwd_err` or `bwd_err` is present), iterative refinement of the solution is performed (in working precision), to reduce the backward error as far as possible.

The algorithms are derived from LAPACK (see Anderson *et al.* [1]).

6.2 Accuracy

The accuracy of the computed solution is given by the forward and backward error bounds which are returned in the optional arguments `fwd_err` and `bwd_err`.

The backward error bound `bwd_err` is rigorous; the forward error bound `fwd_err` is an estimate, but is almost always satisfied.

For each right-hand side b, the computed solution $\hat{x}$ is the exact solution of a perturbed system of equations $(A + E)\hat{x} = b$. Assuming `uplo` = `'u'`:

$$|E| \leq c(k+1)\epsilon|U^H|\,|U|$$

where $c(k+1)$ is a modest linear function of $k+1$, and ϵ = `EPSILON(1.0_`*wp*`)`. This assumes $k \ll n$.

The condition number $\kappa_\infty(A)$ gives a general measure of the *sensitivity* of the solution of $Ax = b$, either to uncertainties in the data or to rounding errors in the computation. An estimate of the reciprocal of $\kappa_\infty(A)$ is returned by `nag_sym_bnd_lin_fac` in its optional argument `rcond`. However, forward error bounds derived using this condition number may be more pessimistic than the bounds returned in `fwd_err`, if present.

If the reciprocal of the condition number is less than `EPSILON(1.0_`*wp*`)`, then A is singular to working precision; if the factorization is used to solve a system of linear equations, the computed solution may have no meaningful accuracy and should be treated with great caution.

6.3 Timing

The number of real floating-point operations required to compute the solutions is roughly $4nkr$ if A is real, and $16nkr$ if A is complex, assuming $n \gg k$.

To compute the error bounds fwd_err and bwd_err usually requires about 5 times as much work.

Example 1: Solution of a Real Symmetric Positive Definite Banded System of Linear Equations

Solve a real symmetric positive definite banded system of linear equations, with one right-hand side $Ax = b$. Estimate the condition number of A, and forward and backward error bounds on the computed solutions. This example calls the procedure `nag_sym_bnd_lin_sol`.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_sym_bnd_lin_sys_ex01

  ! Example Program Text for nag_sym_bnd_lin_sys
  ! NAG fl90, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_sym_bnd_lin_sys, ONLY : nag_sym_bnd_lin_sol
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND, MAX, MIN
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, j, k, n
  REAL (wp) :: bwd_err, fwd_err, rcond
  CHARACTER (1) :: uplo
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: a(:,:), b(:)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) &
   'Example Program Results for nag_sym_bnd_lin_sys_ex01'

  READ (nag_std_in,*)          ! Skip heading in data file
  READ (nag_std_in,*) n, k
  READ (nag_std_in,*) uplo

  ALLOCATE (a(k+1,n),b(n))     ! Allocate storage

  SELECT CASE (uplo)
  CASE ('L','l')
    DO i = 1, n
      READ (nag_std_in,*) (a(1+i-j,j),j=MAX(1,i-k),i)
    END DO
  CASE ('U','u')
    DO i = 1, n
      READ (nag_std_in,*) (a(k+1+i-j,j),j=i,MIN(n,i+k))
    END DO
  END SELECT

  READ (nag_std_in,*) b

  ! Solve the system of equations

  CALL nag_sym_bnd_lin_sol(uplo,a,b,bwd_err=bwd_err,fwd_err=fwd_err, &
   rcond=rcond)

  WRITE (nag_std_out,*)
  WRITE (nag_std_out,'(1X,''kappa(A) (1/rcond)''/2X,ES11.2)') 1/rcond
```

```
      WRITE (nag_std_out,*)
      WRITE (nag_std_out,'(a,100(/f12.4:))') ' Solution', b
      WRITE (nag_std_out,*)
      WRITE (nag_std_out,*) 'Backward error bound'
      WRITE (nag_std_out,'(2X,4ES11.2)') bwd_err
      WRITE (nag_std_out,*)
      WRITE (nag_std_out,*) 'Forward error bound (estimate)'
      WRITE (nag_std_out,'(2X,4ES11.2)') fwd_err

      DEALLOCATE (a,b)             ! Deallocate storage

    END PROGRAM nag_sym_bnd_lin_sys_ex01
```

2 Program Data

```
Example Program Data for nag_sym_bnd_lin_sys_ex01
 4 1                    :Value of n and k
 'L'                    :Value of uplo
 5.49
 2.68  5.63
      -2.39  2.60
            -2.22  5.17  :End of Matrix A
 22.09
  9.31
 -5.24
 11.83                   :End of right-hand side vector b
```

3 Program Results

```
 Example Program Results for nag_sym_bnd_lin_sys_ex01

 kappa(A) (1/rcond)
     7.42E+01

 Solution
      5.0000
     -2.0000
     -3.0000
      1.0000

 Backward error bound
     4.43E-17

 Forward error bound (estimate)
     3.84E-14
```

Example 2: Factorization of a Hermitian Positive Definite Band Matrix and Solution of a Related System of Linear Equations

Solve a complex Hermitian positive definite banded system of linear equations, with many right-hand sides $AX = B$. Estimate forward and backward error bounds on the computed solution. This example calls `nag_sym_bnd_lin_fac` to factorize A, then `nag_sym_bnd_lin_sol_fac` to solve the equations using the factorization.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_sym_bnd_lin_sys_ex02

  ! Example Program Text for nag_sym_bnd_lin_sys
  ! NAG fl90, Release 4. NAG Copyright 2000.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_sym_bnd_lin_sys, ONLY : nag_sym_bnd_lin_fac, &
   nag_sym_bnd_lin_sol_fac
  USE nag_write_mat, ONLY : nag_write_gen_mat, nag_write_bnd_mat
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC EPSILON, KIND, MAX, MIN, SCALE
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: det_exp, i, j, k, ku, n, nrhs
  REAL (wp) :: det_frac, rcond
  CHARACTER (1) :: uplo
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: bwd_err(:), fwd_err(:)
  COMPLEX (wp), ALLOCATABLE :: a(:,:), a_fac(:,:), b(:,:)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) 'Example Program Results for nag_sym_lin_sys_ex02'

  READ (nag_std_in,*)          ! Skip heading in data file
  READ (nag_std_in,*) n, k, nrhs
  READ (nag_std_in,*) uplo

  ALLOCATE (a(k+1,n),b(n,nrhs),a_fac(k+1,n),bwd_err(nrhs),fwd_err(nrhs))
  ! Allocate storage

  a = 0.0_wp
  SELECT CASE (uplo)
  CASE ('L','l')
    ku = 0
    DO i = 1, n
      READ (nag_std_in,*) (a(1+i-j,j),j=MAX(1,i-k),i)
    END DO
  CASE ('U','u')
    ku = k
    DO i = 1, n
      READ (nag_std_in,*) (a(k+1+i-j,j),j=i,MIN(n,i+k))
    END DO
  END SELECT
```

```
    READ (nag_std_in,*) (b(i,:),i=1,n)

    ! Carry out the Cholesky factorisation
    a_fac = a

    CALL nag_sym_bnd_lin_fac(uplo,a_fac,rcond=rcond,det_frac=det_frac, &
     det_exp=det_exp)

    WRITE (nag_std_out,*)

    CALL nag_write_bnd_mat(ku,a_fac,format='(f7.4)', &
     title='Details of Cholesky factorisation')

    WRITE (nag_std_out,*)
    WRITE (nag_std_out, &
     '(1X,''determinant = SCALE(det_frac,det_exp) ='',2X,ES11.3)') &
     SCALE(det_frac,det_exp)

    WRITE (nag_std_out,*)
    WRITE (nag_std_out,'(1X,''kappa(A) (1/rcond)''/2X,ES11.2)') 1/rcond
    WRITE (nag_std_out,*)
    IF (rcond<EPSILON(1.0_wp)) THEN
      WRITE (nag_std_out,*)
      WRITE (nag_std_out,*) ' ** WARNING ** '
      WRITE (nag_std_out,*) &
       'The matrix is almost singular: the solution may have no accuracy.'
      WRITE (nag_std_out,*) &
       'Examine the forward error bounds estimate returned in fwd_err.'
    END IF

    ! Solve the system of equations

    CALL nag_sym_bnd_lin_sol_fac(uplo,a_fac,b,a=a,bwd_err=bwd_err, &
     fwd_err=fwd_err)

    WRITE (nag_std_out,*)
    WRITE (nag_std_out,*) &
     'Result of the solution of the simultaneous equations'
    WRITE (nag_std_out,*)

    CALL nag_write_gen_mat(b,format='(F7.4)',int_col_labels=.TRUE., &
     title='Solutions (one per column)')

    WRITE (nag_std_out,*)
    WRITE (nag_std_out,*) 'Backward error bounds'
    WRITE (nag_std_out,'(2X,4(7X,ES11.2:))') bwd_err
    WRITE (nag_std_out,*)
    WRITE (nag_std_out,*) 'Forward error bounds (estimates)'
    WRITE (nag_std_out,'(2X,4(7X,ES11.2:))') fwd_err

    DEALLOCATE (a,b,a_fac,bwd_err,fwd_err) ! Deallocate storage

  END PROGRAM nag_sym_bnd_lin_sys_ex02
```

2 Program Data

```
Example Program Data for nag_sym_bnd_lin_sys_ex02
 4 1 2                                               :Value of n,k,nrhs
 'U'                                                 :Value of uplo
 (9.39,0.00) (1.08,-1.73)
             (1.69, 0.00) (-0.04,0.29)
                          ( 2.65,0.00) (-0.33,2.24)
                                       ( 2.17,0.00)  :End of Matrix A
 (-12.42,68.42) (54.30,-56.56)
 ( -9.93, 0.88) (18.32,  4.76)
 (-27.30,-0.01) (-4.40,  9.97)
 (  5.31,23.63) ( 9.43,  1.41)    :End of right-hand sides (one per column)
```

3 Program Results

```
 Example Program Results for nag_sym_lin_sys_ex02

 Details of Cholesky factorisation
   ( 3.0643, 0.0000) ( 0.3524,-0.5646)
                     ( 1.1167, 0.0000) (-0.0358, 0.2597)
                                       ( 1.6066, 0.0000) (-0.2054, 1.3942)
                                                         ( 0.4289, 0.0000)

 determinant = SCALE(det_frac,det_exp) =    5.561E+00

 kappa(A) (1/rcond)
     1.22E+02


 Result of the solution of the simultaneous equations

 Solutions (one per column)
                  1                 2
   (-1.0000, 8.0000) ( 5.0000,-6.0000)
   ( 2.0000,-3.0000) ( 2.0000, 3.0000)
   (-4.0000,-5.0000) (-8.0000, 4.0000)
   ( 7.0000, 6.0000) (-1.0000,-7.0000)

 Backward error bounds
             1.84E-16          3.28E-16

 Forward error bounds (estimates)
             3.63E-14          2.20E-14
```

Additional Examples

Not all example programs supplied with NAG *fl*90 appear in full in this module document. The following additional examples, associated with this module, are available.

`nag_sym_bnd_lin_sys_ex03`

Solution of a real symmetric positive definite banded system of linear equations, with many right-hand sides.

`nag_sym_bnd_lin_sys_ex04`

Solution of a complex Hermitian positive definite banded system of linear equations, with one right-hand side.

`nag_sym_bnd_lin_sys_ex05`

Factorization of a real symmetric positive definite band matrix, and use of the factorization to solve a system of linear equations with many right-hand sides.

`nag_sym_bnd_lin_sys_ex06`

Solution of a complex Hermitian positive definite banded system of linear equations, with many right-hand sides.

References

[1] Anderson E, Bai Z, Bischof C, Demmel J, Dongarra J J, Du Croz J J, Greenbaum A, Hammarling S, McKenney A, Blackford S and Sorensen D (1999) *LAPACK Users' Guide* (3rd Edition) SIAM, Philadelphia

[2] Golub G H and Van Loan C F (1989) *Matrix Computations* Johns Hopkins University Press (2nd Edition)

[3] Higham N J (1988) Algorithm 674: 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