Module 5.4: nag_gen_bnd_lin_sys
General Banded Systems of Linear Equations

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

$Ax = b$ or $AX = B$.

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).

There are options to solve alternative forms of the equations:

$A^T x = b$, $A^T X = B$, $A^H x = b$ or $A^H X = B$.

(If A is real, then $A^H = A^T$.)

In this module, the square matrix A (the *coefficient matrix*) is assumed to be a general real or complex *band* matrix. The procedures take advantage of bandedness in order to economize on the work and storage required. If A is symmetric or Hermitian and positive definite, as well as banded, see the module `nag_sym_bnd_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 compute an *LU factorization* of A (with row interchanges to ensure numerical stability). The system of equations can then be solved by forward and backward substitution.

2 Choice of Procedures

The procedure `nag_gen_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_gen_bnd_lin_fac` computes a factorization of A, with options to evaluate the determinant and to estimate the condition number;

`nag_gen_bnd_lin_sol_fac` solves the system of equations, assuming that A has already been factorized by a call to `nag_gen_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_gen_bnd_lin_fac` to be reused several times in repeated calls to `nag_gen_bnd_lin_sol_fac`.

3 Storage of Matrices

The procedures in this module use the following storage scheme for the general band matrix A with k_l sub-diagonals and k_u super-diagonals:

- a_{ij} is stored in `a`$(k_u + i - j + 1, j)$, for $\max(j - k_u, 1) \le i \le \min(j + k_l, n)$.

For example

General band matrix A	Band storage in array `a`
$\begin{pmatrix} a_{11} & a_{12} & & & \\ a_{21} & a_{22} & a_{23} & & \\ a_{31} & a_{32} & a_{33} & a_{34} & \\ & a_{42} & a_{43} & a_{44} & a_{45} \\ & & a_{53} & a_{54} & a_{55} \end{pmatrix}$	$\begin{matrix} * & a_{12} & a_{23} & a_{34} & a_{45} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{21} & a_{32} & a_{43} & a_{54} & * \\ a_{31} & a_{42} & a_{53} & * & * \end{matrix}$

Procedure: nag_gen_bnd_lin_sol

1 Description

`nag_gen_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.*

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).

Optionally, the procedure can solve alternative forms of the system of equations:

$$A^T x = b, \quad A^T X = B, \quad A^H x = b \quad \text{or} \quad A^H X = B.$$

(If A is real, then $A^H = A^T$.)

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_gen_bnd_lin_sys
CALL nag_gen_bnd_lin_sol(ku, a, b  [, optional arguments])
```

2.1 Interfaces

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

Real / complex data

Real data:	`a`, `b` and the optional argument `a_fac` are of type real(kind=*wp*).
Complex data:	`a`, `b` and the optional argument `a_fac` 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 band matrix A
$k_l \geq 0$	— the number of sub-diagonals in the band matrix A
r	— the number of right-hand sides

3.1 Mandatory Arguments

ku — integer, intent(in)

Input: the number k_u of super-diagonals in the band matrix A.

Constraints: `ku` ≥ 0.

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

Input: the general band matrix A; element a_{ij} must be stored in `a`$(k_u + i - j + 1, j)$ for $\max(j - k_u, 1) \leq i \leq \min(j + k_l, n)$.

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.

trans — character(len=1), intent(in), optional

Input: specifies whether the equations involve A or its transpose A^T or its conjugate-transpose A^H ($= A^T$ if A is real).

If `trans` = `'n'` or `'N'`, the equations involve A (i.e., $Ax = b$);

if `trans` = `'t'` or `'T'`, the equations involve A^T (i.e., $A^T x = b$);

if `trans` = `'c'` or `'C'`, the equations involve A^H (i.e., $A^H x = b$).

Default: `trans` = `'n'`.

Constraints: `trans` = `'n'`, `'N'`, `'t'`, `'T'`, `'c'` or `'C'`.

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 ($\kappa_\infty(A)$, if `trans` = `'n'` or `'N'`; $\kappa_1(A)$ otherwise). `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.

a_fac$(2k_l + k_u + 1, n)$ — real(kind=*wp*) / complex(kind=*wp*), intent(out), optional

Output: details of the factorization of A: the upper triangular band matrix U with $k_l + k_u$ superdiagonals is stored in rows 1 to $k_l + k_u + 1$, and the multipliers used to form the matrix L are stored in rows $k_l + k_u + 2$ to $2k_l + k_u + 1$.

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

pivot(n) — integer, intent(out), optional

Output: the pivot indices used in the LU factorization; row i was interchanged with row `pivot(`i`)` for $i = 1, 2, \ldots, n$.

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	Singular matrix. The matrix A has been factorized, but the factor U has a zero diagonal element, and so is exactly singular. 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 or equal to `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_gen_bnd_lin_fac` to factorize A, and to estimate the condition number. It then calls `nag_gen_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 Chapters 3 and 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. If the system has one of the alternative forms $A^T x = b$ or $A^H x = b$, the appropriate condition number is $\kappa_1(A)$ $(= \kappa_\infty(A^T) = \kappa_\infty(A^H))$. An estimate of the reciprocal of $\kappa_\infty(A)$ or $\kappa_1(A)$ 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 $n \times k_l \times (k_l + k_u + 1)$, assuming $n \gg k_l, k_u$ 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_gen_bnd_lin_fac

1 Description

nag_gen_bnd_lin_fac is a generic procedure which factorizes a general real or complex band matrix A of order n, with k_l sub-diagonals and k_u super-diagonals, using partial pivoting with row interchanges. The factorization is written as:

$$A = PLU$$

where

P is a permutation matrix,

L is lower triangular with unit diagonal elements, and at most k_l non-zero elements in each column;

U is upper triangular and banded with $k_l + k_u$ super-diagonals.

This procedure can also return the determinant of A and estimates of the condition numbers numbers $\kappa_1(A)$ and $\kappa_\infty(A)$.

2 Usage

```
USE nag_gen_bnd_lin_sys
CALL nag_gen_bnd_lin_fac(ku, a, a_fac, pivot  [, 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
$k_l \geq 0$	— the number of sub-diagonals in the band matrix A

3.1 Mandatory Arguments

ku — integer, intent(in)

Input: the number k_u of super-diagonals in the band matrix A.

Constraints: ku ≥ 0.

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

Input: the general band matrix A; element a_{ij} must be stored in a$(k_u + i - j + 1, j)$ for $\max(j - k_u, 1) \leq i \leq \min(j + k_l, n)$.

a_fac$(2k_l + k_u + 1, n)$ — real(kind=*wp*) / complex(kind=*wp*), intent(out)

Output: details of the factorization of A: the upper triangular band matrix U with $k_l + k_u$ super-diagonals is stored in rows 1 to $k_l + k_u + 1$, and the multipliers used to form the matrix L are stored in rows $k_l + k_u + 2$ to $2k_l + k_u + 1$.

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

pivot(n) — integer, intent(out)

Output: the pivot indices; row i was interchanged with row `pivot(`i`)` for $i = 1, 2, \ldots, n$.

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

Output: an estimate of the reciprocal of the condition number of A in the ∞-norm, $\kappa_\infty(A)$.

rcond_1 — real(kind=*wp*), intent(out), optional

Output: an estimate of the reciprocal of the condition number of A in the 1-norm, $\kappa_1(A)$.

det_frac — real(kind=*wp*) / complex(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: `det_frac` must be of the same type as `a` and 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.
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	Singular matrix. The matrix A has been factorized, but the factor U has a zero diagonal element, and so is exactly singular. 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 *LU* factorization is computed using partial pivoting with row interchanges. See Section 4.3.4 of Golub and Van Loan [2]. The row interchanges may cause fill-in outside the band of A, and therefore the number of nonzero superdiagonals in the upper triangular factor U is $k_l + k_u$.

To estimate the condition number $\kappa_1(A)$ $(= \|A\|_1 \|A^{-1}\|_1)$, 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 $\rho = 1/\kappa_1(A)$, rather than $\kappa_1(A)$ itself.

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

6.2 Accuracy

The computed factors L and U are the exact factors of a perturbed matrix $A + E$, such that

$$|E| \leq c(k)\epsilon P|L||U|,$$

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

The computed estimate `rcond_inf` or `rcond_1` 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 for the *LU* factorization is roughly $2n \times k_l \times (k_l + k_u + 1)$ for real A, or $8n \times k_l \times (k_l + k_u + 1)$ for complex A, assuming $n \gg k_l, k_u$.

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 $2n(2k_l + k_u)$ floating-point operations if A is real, or $8n(2k_l + k_u)$ if A is complex. Thus, for large n, the cost is much less than that of directly computing A^{-1} and its norm, which would require $O(n^2(2k_l + k_u))$ operations.

Procedure: nag_gen_bnd_lin_sol_fac

1 Description

`nag_gen_bnd_lin_sol_fac` is a generic procedure which computes the solution of a general real or complex banded system of linear equations with one or many right-hand sides, assuming that the coefficient matrix has already been factorized by `nag_gen_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 matrix A (the *coefficient matrix*) is assumed to be a general band matrix, which has already been factorized by a call to `nag_gen_bnd_lin_fac`.

Optionally, the procedure can solve alternative forms of the system of equations:

$$A^Tx = b, \quad A^TX = B, \quad A^Hx = b \quad \text{or} \quad A^HX = B.$$

(If A is real, then $A^H = A^T$.)

The procedure also has options to return *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 solution in order to guarantee a small backward error in the computed solution; in this case, a copy of the original matrix A must be supplied in the optional argument `a`, as well as the factorized form in the argument `a_fac`.

2 Usage

```
USE nag_gen_bnd_lin_sys
CALL nag_gen_bnd_lin_sol_fac(ku, a_fac, pivot, 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 '`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
$k_l \geq 0$	— the number of sub-diagonals in the band matrix A
r	— the number of right-hand sides

3.1 Mandatory Arguments

ku — integer, intent(in)

Input: the number k_u of super-diagonals in the band matrix A.

Constraints: `ku` ≥ 0.

a_fac$(2k_l + k_u + 1, n)$ — real(kind=*wp*) / complex(kind=*wp*), intent(in)

Input: the *LU* factorisation of A, as returned by `nag_gen_bnd_lin_fac`.

pivot(n) — integer, intent(in)

Input: the pivot indices, as returned by `nag_gen_bnd_lin_fac`.

Constraints: $i \leq$ `pivot`(i) $\leq n$, for $i = 1, 2, \ldots, n$.

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.

trans — character(len=1), intent(in), optional

Input: specifies whether the equations involve A or its transpose A^T or its conjugate-transpose A^H ($= A^T$ if A is real).

If `trans` = `'n'` or `'N'`, the equations involve A (i.e., $Ax = b$);

if `trans` = `'t'` or `'T'`, the equations involve A^T (i.e., $A^T x = b$);

if `trans` = `'c'` or `'C'`, the equations involve A^H (i.e., $A^H x = b$).

Default: `trans` = `'n'`.

Constraints: `trans` = `'n'`, `'N'`, `'t'`, `'T'`, `'c'` or `'C'`.

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

Input: the original coefficient matrix A, as supplied to `nag_gen_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.
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	Singular matrix. In the factorization supplied in `a_fac`, the factor U has a zero diagonal element, and so is exactly singular. 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:

to solve $Ax = b$ (in factorized form, $PLUx = b$), $PLy = b$ is solved for y, and then $Ux = y$ is solved for x;

to solve $A^T x = b$ (in factorized form, $U^T L^T P^T x = b$), $U^T y = b$ is solved for y, and then $L^T P^T x = y$ is solved for x. A similar approach is used to solve $A^H x = b$.

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 $(A + E)\hat{x} = b$, such that

$$|E| \leq c(k)\epsilon P|L||U|,$$

where $c(k)$ is a modest linear function of $k = k_l + k_u + 1$, and $\epsilon =$ `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. If the system has one of the alternative forms $A^T x = b$ or $A^H x = b$, the appropriate condition number is $\kappa_1(A)$ $(= \kappa_\infty(A^T) = \kappa_\infty(A^H))$. Estimates of the reciprocals of $\kappa_\infty(A)$ and $\kappa_1(A)$ are returned by `nag_gen_bnd_lin_fac` in its optional arguments `rcond_inf` and `rcond_1`. However, forward error bounds derived using these condition numbers 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 $2nr(2k_l + k_u)$ if A is real, and $16nr(2k_l + k_u)$ if A is complex, assuming $n \gg k_l, k_u$.

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

Example 1: Solution of a general real banded system of linear equations

Solve a general real 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 solution. This example calls the procedure `nag_gen_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_gen_bnd_lin_sys_ex01

  ! Example Program Text for nag_gen_bnd_lin_sys
  ! NAG fl90, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_gen_bnd_lin_sys, ONLY : nag_gen_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, kl, ku, n
  REAL (wp) :: bwd_err, fwd_err, rcond
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: a(:,:), b(:)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) &
   'Example Program Results for nag_gen_bnd_lin_sys_ex01'

  READ (nag_std_in,*)          ! Skip heading in data file
  READ (nag_std_in,*) n, kl, ku

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

  DO i = 1, n
    READ (nag_std_in,*) (a(ku+1+i-j,j),j=MAX(1,i-kl),MIN(i+ku,n))
  END DO
  READ (nag_std_in,*) b

  ! Solve the system of equations

  CALL nag_gen_bnd_lin_sol(ku,a,b,rcond=rcond,bwd_err=bwd_err, &
   fwd_err=fwd_err)

  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,*) 'Solution'
  WRITE (nag_std_out,'(4X,F9.4)') 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_gen_bnd_lin_sys_ex01
```

2 Program Data

```
Example Program Data for nag_gen_bnd_lin_sys_ex01
  4 1 2                       :Values of n, kl and ku
-0.23  2.54  -3.66
-6.98  2.46  -2.73  -2.13
       2.56   2.46   4.07
             -4.78  -3.82     :End of matrix A
 4.42
27.13
-6.14
10.50                         :End of right-hand side vector b
```

3 Program Results

```
 Example Program Results for nag_gen_bnd_lin_sys_ex01

 kappa(A) (1/rcond)
     5.13E+01

 Solution
      -2.0000
       3.0000
       1.0000
      -4.0000

 Backward error bound
     1.37E-16

 Forward error bound (estimate)
     2.96E-14
```

Example 2: Factorization of a general complex band matrix and use of the factorization to solve a system of linear equations

Solve a general complex banded system of linear equations with many right-hand sides $AX = B$, also returning forward and backward error bounds on the computed solution. This example calls `nag_gen_bnd_lin_fac` to factorize A, and then `nag_gen_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_gen_bnd_lin_sys_ex02

  ! Example Program Text for nag_sym_lin_sys
  ! NAG fl90, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_gen_bnd_lin_sys, ONLY : nag_gen_bnd_lin_fac, &
   nag_gen_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, kl, ku, n, nrhs
  REAL (wp) :: rcond
  COMPLEX (wp) :: det_frac
  CHARACTER (1) :: trans
  ! .. Local Arrays ..
  INTEGER, ALLOCATABLE :: pivot(:)
  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_gen_lin_sys_ex02'

  READ (nag_std_in,*)          ! Skip heading in data file
  READ (nag_std_in,*) n, kl, ku, nrhs
  READ (nag_std_in,*) trans

  ALLOCATE (a(kl+ku+1,n),b(n,nrhs),a_fac(2*kl+ku+1,n),bwd_err(nrhs), &
   fwd_err(nrhs),pivot(n))     ! Allocate storage

  DO i = 1, n
    READ (nag_std_in,*) (a(ku+1+i-j,j),j=MAX(1,i-kl),MIN(i+ku,n))
  END DO
  READ (nag_std_in,*) (b(i,:),i=1,n)

  ! Carry out LU factorisation

  CALL nag_gen_bnd_lin_fac(ku,a,a_fac,pivot,rcond_1=rcond, &
   det_frac=det_frac,det_exp=det_exp)

  WRITE (nag_std_out,*)
```

```
      WRITE (nag_std_out,*) 'Result of LU factorisation '
      WRITE (nag_std_out,*)

      CALL nag_write_bnd_mat(ku+kl,a_fac,format='(F7.4)',title='Array a_fac')

      WRITE (nag_std_out,*)
      WRITE (nag_std_out,*) 'Pivotal sequence (pivot)'
      WRITE (nag_std_out,'(2X,10I4:)') pivot
      WRITE (nag_std_out,*)
      WRITE (nag_std_out,'(1X,''determinant = det_frac*SCALE(1.0_wp,det_exp) &
       &='',2X,"(",ES11.3,",",ES11.3,")")') det_frac*SCALE(1.0_wp,det_exp)
      WRITE (nag_std_out,*)
      WRITE (nag_std_out,'(1X,''kappa(A) (1/rcond)''/2X,ES11.2)') 1/rcond
      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_gen_bnd_lin_sol_fac(ku,a_fac,pivot,b,a=a,trans=trans, &
       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,pivot,bwd_err,fwd_err) ! Deallocate storage

    END PROGRAM nag_gen_bnd_lin_sys_ex02
```

2 Program Data

```
Example Program Data for nag_gen_bnd_lin_sys_ex02
  4 1 2 2                                    :Values of n, kl, ku and nrhs
   'N'                                       :Value of trans
(-1.65,2.26) (-2.05,-0.85) ( 0.97,-2.84)
( 0.00,6.30) (-1.48,-1.75) (-3.99, 4.01) ( 0.59,-0.48)
             (-0.77, 2.83) (-1.06, 1.94) ( 3.33,-1.04)
                           ( 4.48,-1.09) (-0.46,-1.72)    :End of matrix A
( -1.06, 21.50) ( 12.85, 2.84)
(-22.72,-53.90) (-70.22,21.57)
( 28.24,-38.60) (-20.73,-1.23)
(-34.56, 16.73) ( 26.01,31.97) :End of right-hand sides (one rhs per column)
```

3 Program Results

```
Example Program Results for nag_gen_lin_sys_ex02

Result of LU factorisation

Array a_fac
  ( 0.0000, 6.3000) (-1.4800,-1.7500) (-3.9900, 4.0100) ( 0.5900,-0.4800)
  ( 0.3587, 0.2619) (-0.7700, 2.8300) (-1.0600, 1.9400) ( 3.3300,-1.0400)
                    ( 0.2314, 0.6358) ( 4.9303,-3.0086) (-1.7692,-1.8587)
                                      ( 0.7604, 0.2429) ( 0.4338, 0.1233)

Pivotal sequence (pivot)
    2   3   3   4

determinant = det_frac*SCALE(1.0_wp,det_exp) =  ( -4.812E+01,  2.601E-01)

kappa(A) (1/rcond)
    1.04E+02

Result of the solution of the simultaneous equations

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

Backward error bounds
           9.81E-17          6.74E-17

Forward error bounds (estimates)
           7.04E-14          8.44E-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_gen_bnd_lin_sys_ex03`

Solution of a general real banded system of linear equations with many right-hand sides.

`nag_gen_bnd_lin_sys_ex04`

Solution of a general complex banded system of linear equations with one right-hand side.

`nag_gen_bnd_lin_sys_ex05`

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

`nag_gen_bnd_lin_sys_ex06`

Solution of a general complex 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, Ostrouchov S and Sorensen D (1995) *LAPACK Users' Guide* (2nd 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