Module 4.2: nag_mat_inv
Matrix Inversion

`nag_mat_inv` provides procedures for matrix inversion.

Contents

Introduction

This module provides procedures to compute the inverse of a matrix.

It is seldom necessary to compute the explicit inverse of a matrix. In particular, do not attempt to solve $Ax = b$ by first computing A^{-1} and then forming the matrix vector product $x = A^{-1}b$. The procedures provided by `nag_gen_lin_sys`, `nag_sym_lin_sys` and `nag_tri_lin_sys` are more efficient and more accurate.

1 Choice of procedures

The following procedures are provided:

`nag_gen_mat_inv` computes the inverse of a general real or complex matrix;

`nag_gen_mat_inv_fac` computes the inverse of a general real or complex matrix, with the matrix previously factorized using `nag_gen_lin_fac`;

`nag_sym_mat_inv` computes the inverse of a real or complex, symmetric or Hermitian matrix;

`nag_sym_mat_inv_fac` computes the inverse of a real or complex, symmetric or Hermitian matrix, with the matrix previously factorized using `nag_sym_lin_fac`;

`nag_tri_mat_inv` computes the inverse of a real or complex triangular matrix.

Procedure: nag_gen_mat_inv

1 Description

nag_gen_mat_inv is a generic procedure which computes the inverse of a general real or complex matrix A.

The matrix is assumed to be a general matrix, without any known special properties such as symmetry.

The procedure also has an option to return an estimate of the *condition number*, $\kappa_\infty(A)$.

2 Usage

```
USE nag_mat_inv
CALL nag_gen_mat_inv(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.

$n \geq 1$ — the order of the matrix A

3.1 Mandatory Argument

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

Input: the general matrix A.

Output: a is overwritten by the inverse A^{-1}.

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, $\kappa_\infty(A)$. 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
302	An array argument has an invalid shape.
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 inverse is computed.

Warnings (error%level = 1):

error%code	Description
101	Approximately singular matrix. 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 inverse has no accuracy at all.

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_lin_fac` (see the module `nag_gen_lin_sys`), which computes the LU factorization of A as $A = PLU$. The inverse of A, X, is computed by a call to `nag_gen_mat_inv_fac`, which first forms U^{-1} then solves $XPL = U^{-1}$ for X.

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

6.2 Accuracy

The computed inverse, X, satisfies

$$|XA - I| \leq c(n)\ \epsilon\ |X|\ P\ |L|\ |U|,$$

where $c(n)$ is a modest linear function of n, and $\epsilon =$ `EPSILON(1.0_`*wp*`)`.

Note that a similar bound for $|AX - I|$ cannot be guaranteed, although it is almost always satisfied. See Du Croz and Higham [2].

6.3 Timing

The time taken is roughly proportional to n^3. The time taken for complex data is about 4 times as long as that for real data.

Procedure: nag_gen_mat_inv_fac

1 Description

nag_gen_mat_inv_fac is a generic procedure which computes the inverse of a general real or complex matrix A, assuming that the coefficient matrix has already been factorized by nag_gen_lin_fac (see the module nag_gen_lin_sys).

The matrix is assumed to be a general matrix, without any known special properties such as symmetry.

2 Usage

```
USE nag_mat_inv
CALL nag_gen_mat_inv_fac(a, 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.

$n \geq 1$ — the order of the matrix A

3.1 Mandatory Arguments

a(n,n) — real(kind=wp) / complex(kind=wp), intent(inout)

Input: the LU factorization of A, as returned by nag_gen_lin_fac.

Output: a is overwritten by the inverse A^{-1}.

pivot(n) — integer, intent(in)

Input: the pivot indices, as returned by nag_gen_lin_fac.

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

3.2 Optional Argument

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.
	In the factorization supplied in `a`, the factor U has a zero diagonal element, and so is exactly singular. No inverse is 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

To use `nag_gen_mat_inv_fac` to compute the inverse of a matrix X, the user must first call `nag_gen_lin_fac` (see the module `nag_gen_lin_sys`), to compute the LU factorization of A as $A = PLU$. `nag_gen_mat_inv_fac` computes the inverse of A, X, by first forming U^{-1} and then solving $XPL = U^{-1}$ for X.

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

6.2 Accuracy

The computed inverse, X, satisfies

$$|XA - I| \leq c(n)\ \epsilon\ |X|\ P\ |L|\ |U|,$$

where $c(n)$ is a modest linear function of n, and $\epsilon =$ `EPSILON(1.0_`*wp*`)`.
Note that a similar bound for $|AX - I|$ cannot be guaranteed, although it is almost always satisfied. See Du Croz and Higham [2].

6.3 Timing

The number of real floating-point operations required to compute the inverse is roughly $(4/3)n^3$ if A is real, and $(16/3)n^3$ if A is complex.

Procedure: nag_sym_mat_inv

1 Description

`nag_sym_mat_inv` is a generic procedure which computes the inverse of a matrix A, where the matrix may be:

real symmetric indefinite,

complex Hermitian indefinite,

complex symmetric,

real symmetric positive definite, or

complex Hermitian positive definite.

Here the term *indefinite* refers to a matrix that is not *known* to be positive definite, although it may in fact be so.

The procedure allows conventional or packed storage for A.

The procedure also has an option to return an estimate of the *condition number*, $\kappa_\infty(A)$.

2 Usage

```
USE nag_mat_inv
CALL nag_sym_mat_inv(nag_key, uplo, a  [, optional arguments])
```

2.1 Interfaces

Distinct interfaces are provided for each of the 12 combinations of the following cases:

Symmetric indefinite / Hermitian indefinite / positive definite matrix

Symmetric indefinite: `nag_key` = `nag_key_sym`.

Hermitian indefinite: `nag_key` = `nag_key_herm`;
for real matrices this is equivalent to `nag_key_sym`.

Positive definite: `nag_key` = `nag_key_pos`.

Real / complex data

Real data: `a` is of type real(kind=*wp*).

Complex data: `a` is of type complex(kind=*wp*).

Conventional / packed storage (see the Chapter Introduction)

Conventional: `a` is a rank-2 array.

Packed: `a` is a rank-1 array.

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.

$n \geq 1$ — the order of the matrix A

3.1 Mandatory Arguments

nag_key — a "key" argument, intent(in)

Input: must have one of the following values (which are named constants, each of a different derived type, defined by the Library, and accessible from this module).

`nag_key_sym`: if A is real symmetric indefinite or complex symmetric;

`nag_key_herm`: if A is real or complex Hermitian indefinite;

`nag_key_pos`: if A is real symmetric positive definite or complex Hermitian positive definite.

For further explanation of "key" arguments, see the Essential Introduction.

Note: for real matrices, `nag_key_herm` is equivalent to `nag_key_sym`.

uplo — character(len=1), intent(in)

Input: specifies whether the upper or lower triangle of A is supplied.

If `uplo` = `'u'` or `'U'`, the upper triangle is supplied, and is overwritten by the upper triangular of A^{-1};

if `uplo` = `'l'` or `'L'`, the lower triangle is supplied, and is overwritten by the lower triangular of A^{-1}.

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

a(n, n) / **a**$(n(n+1)/2)$ — real(kind=*wp*) / complex(kind=*wp*), intent(inout)

Input: the matrix A.

Conventional storage (`a` has shape (n, n))

If `uplo` = `'u'`, the upper triangle of A must be stored, and elements below the diagonal need not be set;

if `uplo` = `'l'`, the lower triangle of A must be stored, and elements above the diagonal need not be set.

Packed storage (`a` has shape $(n(n+1)/2)$)

If `uplo` = `'u'`, the upper triangle of A must be stored, packed by columns, with a_{ij} in `a`$(i + j(j-1)/2)$ for $i \leq j$;

if `uplo` = `'l'`, the lower triangle of A must be stored, packed by columns, with a_{ij} in `a`$(i + (2n-j)(j-1)/2)$ for $i \geq j$.

Output: the supplied triangle of A is overwritten by details of the corresponding triangle of the inverse A^{-1}; the other elements of `a` are unchanged.

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, $\kappa_\infty(A)$ ($= \kappa_1(A)$ if 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.
320	The procedure was unable to allocate enough memory.

Failures (error%level = 2):

error%code	Description
201	Singular matrix. This error can only occur if nag_key = nag_key_sym or nag_key_herm. The Bunch–Kaufman factorization has been completed, but the factor D has a zero diagonal block of order 1, and so is exactly singular. No inverse is computed.
202	Matrix not positive definite. This error can only occur if nag_key = nag_key_pos. The Cholesky factorization cannot be completed. Either A is close to singularity, or it has at least one negative eigenvalue. No inverse is computed.

Warnings (error%level = 1):

error%code	Description
101	Approximately singular matrix. 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 inverse has no accuracy at all.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 3 of this module document.

6 Further Comments

6.1 Algorithmic Detail

The procedure first calls nag_sym_lin_fac (see the module nag_sym_lin_sys) to factorize A, and to estimate the condition number if required. It then calls nag_sym_mat_inv_fac to compute the inverse.

If nag_key = nag_key_pos (A is positive definite), then

> if uplo = 'u', nag_sym_lin_fac computes the upper triangular factor U, where $A = U^HU$, then A^{-1} is computed by first inverting U and then forming $(U^{-1})(U^{-1})^H$;
>
> if uplo = 'l', nag_sym_lin_fac computes the lower triangular factor L, where $A = LL^H$, then A^{-1} is computed by first inverting L and then forming $(L^{-1})^H(L^{-1})$.

Otherwise,

> if uplo = 'u', nag_sym_lin_fac computes a permutation matrix P and the upper triangular factor U, where $A = PUDU^TP^T$ (or $PUDU^HP^T$ if A is Hermitian), and A^{-1} is computed by solving $U^TP^TXPU = D^{-1}$ (or $U^HP^TXPU = D^{-1}$ if A is Hermitian);

if uplo = 'l', nag_sym_lin_fac computes a permutation matrix P and the lower triangular factor L, where $A = PLDL^TP^T$ (or $PLDL^HP^T$ if A is Hermitian), and A^{-1} is computed by solving $L^TP^TXPL = D^{-1}$ (or $L^HP^TXPL = D^{-1}$ if A is Hermitian).

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

6.2 Accuracy

If nag_key = nag_key_pos (A is positive definite), then the computed inverse X satisfies

$$\|XA - I\|_2 \leq c(n)\ \epsilon\ \kappa_2(A) \quad \text{and} \quad \|AX - I\|_2 \leq c(n)\ \epsilon\ \kappa_2(A),$$

where $c(n)$ is a modest linear function of n, ϵ = EPSILON(1.0_*wp*)and $\kappa_2(A)$is the condition number of A defined by

$$\kappa_2(A) = \|A\|_2\ \|A^{-1}\|_2.$$

Otherwise, if uplo = 'u', then the computed inverse X satisfies a bound of the form

$$|DU^TP^TXPU - I| \leq c(n)\ \epsilon\ (|D|\ |U^T|\ P^T\ |X|\ P\ |U| + |D|\ |D^{-1}|)$$

(or $|DU^HP^TXPU - I| \leq c(n)\ \epsilon\ (|D|\ |U^H|\ P^T\ |X|\ P\ |U| + |D|\ |D^{-1}|)$ if A is Hermitian); where $c(n)$ is a modest linear function of n, ϵ = EPSILON(1.0_*wp*). If uplo = 'l', similar forms hold for the factors L and D. See Du Croz and Higham [2].

6.3 Timing

The time taken is roughly proportional to n^3, and, is roughly half that taken by the procedure nag_gen_mat_inv which does not take advantage of symmetry. The time taken for complex data is about 4 times as long as that for real data.

The procedure is somewhat faster, especially on high-performance computers, when nag_key is set to nag_key_pos (assuming that A is indeed positive definite).

Procedure: nag_sym_mat_inv_fac

1 Description

nag_sym_mat_inv_fac is a generic procedure which computes the inverse of a real or complex, symmetric or Hermitian matrix A, assuming that the matrix has already been factorized by nag_sym_lin_fac (see the module nag_sym_lin_sys).

The matrix may be:

real symmetric indefinite,

complex Hermitian indefinite,

complex symmetric,

real symmetric positive definite, or

complex Hermitian positive definite.

Here the term *indefinite* refers to a matrix that is not *known* to be positive definite, although it may in fact be so.

The procedure allows conventional or packed storage for A.

2 Usage

```
USE nag_mat_inv
```

```
CALL nag_sym_mat_inv_fac(nag_key, uplo, a, pivot  [, optional arguments])
```

or for positive definite matrices only:

```
CALL nag_sym_mat_inv_fac(nag_key, uplo, a  [, optional arguments])
```

2.1 Interfaces

Distinct interfaces are provided for each of the 16 combinations of the following cases:

Symmetric indefinite / Hermitian indefinite / positive definite matrix

For positive definite matrices, two forms of the interface are provided: the first *includes* pivot as a mandatory argument for compatibility with the interface for indefinite matrices; the second *omits* pivot since it is not needed for Cholesky factorization.

Symmetric indefinite: nag_key = nag_key_sym.

Hermitian indefinite: nag_key = nag_key_herm; for real matrices this is equivalent to nag_key_sym.

positive definite (1): nag_key = nag_key_pos, with pivot as a mandatory argument.

positive definite (2): nag_key = nag_key_pos, with pivot not in the argument list.

Real / complex data

Real data: a is of type real(kind=*wp*).

Complex data: a is of type complex(kind=*wp*).

Conventional / packed storage (see the Chapter Introduction)

Conventional: a is a rank-2 array.

Packed: a is a rank-1 array.

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 '$\mathtt{x}(n)$' 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.

$n \geq 1$ — the order of the matrix A

3.1 Mandatory Arguments

nag_key — a "key" argument, intent(in)

Input: must have one of the following values (which are named constants, each of a different derived type, defined by the Library, and accessible from this module).

nag_key_sym: if the matrix A is real symmetric indefinite or complex symmetric;

nag_key_herm: if the matrix A is real or complex Hermitian indefinite;

nag_key_pos: if the matrix A is real symmetric positive definite or complex Hermitian positive definite.

For further explanation of "key" arguments, see the Essential Introduction.

Note: for real matrices, nag_key_herm is equivalent to nag_key_sym.

uplo — character(len=1), intent(in)

Input: specifies whether the upper or lower triangle of A was supplied to nag_sym_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_lin_fac that returns values used for the next two arguments a and pivot.

a(n, n) / **a**$(n(n+1)/2)$ — real(kind=*wp*) / complex(kind=*wp*), intent(inout)

Input: the factorization of A, as returned by nag_sym_lin_fac.

Output: the supplied triangle of a as defined by uplo is overwritten by details of the corresponding triangle of the inverse A^{-1}; the other elements of a are unchanged.

pivot(n) — integer, intent(in)

Input: the pivot indices, as returned by nag_sym_lin_fac.

Note: if nag_key = nag_key_pos, pivot need not be included in the argument list.

3.2 Optional Argument

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.
	This error can only occur if nag_key = nag_key_sym or nag_key_herm. In the Bunch–Kaufman factorization supplied in a, the factor D has a zero diagonal block of order 1, and so is exactly singular. No inverse is computed.
202	Matrix not positive definite.
	This error can only occur if nag_key = nag_key_pos. The supplied array a does not contain a valid Cholesky factorization, indicating that the original matrix A was not positive definite. No inverse is computed.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 4 of this module document.

6 Further Comments

6.1 Algorithmic Detail

If nag_key = nag_key_pos (A is positive definite), then

if uplo = 'u', the upper triangular factor U is supplied, where $A = U^H U$, and A^{-1} is computed by first inverting U and then forming $(U^{-1})(U^{-1})^H$;

if uplo = 'l', the lower triangular factor L is supplied, where $A = LL^H$, and A^{-1} is computed by first inverting L and then forming $(L^{-1})^H(L^{-1})$.

Otherwise,

if uplo = 'u', a permutation matrix P and the upper triangular factor U are supplied, where $A = PUDU^TP^T$ (or $PUDU^HP^T$ if A is Hermitian), and A^{-1} is computed by solving $U^TP^TXPU = D^{-1}$ (or $U^HP^TXPU = D^{-1}$ if A is Hermitian);

if uplo = 'l', a permutation matrix P and the lower triangular factor L are supplied, where $A = PLDL^TP^T$ (or $PLDL^HP^T$ if A is Hermitian), and A^{-1} is computed by solving $L^TP^TXPL = D^{-1}$ (or $L^HP^TXPL = D^{-1}$ if A is Hermitian).

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

6.2 Accuracy

If nag_key = nag_key_pos (A is positive definite), then the computed inverse X satisfies

$$\|XA - I\|_2 \leq c(n)\ \epsilon\ \kappa_2(A) \quad \text{and} \quad \|AX - I\|_2 \leq c(n)\ \epsilon\ \kappa_2(A),$$

where $c(n)$ is a modest linear function of n, ϵ = EPSILON(1.0_*wp*)and $\kappa_2(A)$is the condition number of A defined by

$$\kappa_2(A) = \|A\|_2\ \|A^{-1}\|_2.$$

Otherwise If uplo = 'u', then the computed inverse X satisfies a bound of the form

$$|DU^T P^T XPU - I| \leq c(n)\ \epsilon\ (|D|\ |U^T|\ P^T\ |X|\ P\ |U| + |D|\ |D^{-1}|);$$

(or $|DU^H P^T XPU - I| \leq c(n)\ \epsilon\ (|D|\ |U^H|\ P^T\ |X|\ P\ |U| + |D|\ |D^{-1}|)$ if A is Hermitian); where $c(n)$ is a modest linear function of n, ϵ = EPSILON(1.0_*wp*). If uplo = 'l', similar forms hold for the factors L and D. See Du Croz and Higham [2].

6.3 Timing

The number of real floating-point operations required to compute the inverse is roughly $(2/3)n^3$ if A is real, and $(8/3)n^3$ if A is complex.

Procedure: nag_tri_mat_inv

1 Description

nag_tri_mat_inv is a generic procedure which computes the inverse of a real or complex triangular matrix A.

The procedure allows conventional or packed storage for A.

The procedure also has an option to return an estimate of the *condition number*, $\kappa_\infty(A)$.

2 Usage

```
USE nag_mat_inv

CALL nag_tri_mat_inv(uplo, a  [, 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 is of type real(kind=wp).

Complex data: a is of type complex(kind=wp).

Conventional / packed storage (see the Chapter Introduction)

Conventional: a is a rank-2 array.

Packed: a is a rank-1 array.

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.

$n \geq 1$ — the order of the matrix A

3.1 Mandatory Arguments

uplo — character(len=1), intent(in)

Input: specifies whether A is upper or lower triangular.

If uplo = 'u' or 'U', A is upper triangular;

if uplo = 'l' or 'L', A is lower triangular.

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

a(n, n) / **a**$(n(n+1)/2)$ — real(kind=wp) / complex(kind=wp), intent(inout)

Input: the triangular matrix A.

Conventional storage (a has shape (n, n))

If uplo = 'u', A is upper triangular, and elements below the diagonal need not be set;

if uplo = 'l', A is lower triangular, and elements above the diagonal need not be set.

Packed storage (a has shape $(n(n+1)/2)$)

If uplo = 'u', A is upper triangular, and its upper triangle must be stored, packed by columns, with a_{ij} in $\mathtt{a}(i+j(j-1)/2)$ for $i \leq j$;

if uplo = 'l', A is lower triangular, and its lower triangle must be stored, packed by columns, with a_{ij} in $\mathtt{a}(i+(2n-j)(j-1)/2)$ for $i \geq j$.

If the optional argument unit_diag = .true., the diagonal elements of A are assumed to be 1; they need not be stored, and are not referenced by the procedure.

Output: a is overwritten by the inverse A^{-1}, using the same storage format described above; the other elements of a are unchanged.

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.

unit_diag — logical, intent(in), optional

Input: specifies whether A has unit diagonal elements.

If unit_diag = .false., the diagonal elements of A must be explicitly stored;

if unit_diag = .true., A has unit diagonal elements: they need not be stored and are assumed to be 1.

Default: unit_diag = .false..

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

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

Failures (error%level = 2):

error%code	Description
201	Singular matrix. A has a zero diagonal element, and so is exactly singular. No inverse is computed.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 5 of this module document.

6 Further Comments

6.1 Algorithmic Detail

The algorithm is derived from LAPACK (see Anderson *et al.* [1]).

6.2 Accuracy

The computed inverse, X, satisfies

$$|XA - I| \leq c(n)\ \epsilon\ |X|\ |A|,$$

where $c(n)$ is a modest linear function of n, and $\epsilon =$ `EPSILON(1.0_`wp`)`.
Note that a similar bound for $|AX - I|$ cannot be guaranteed, although it is almost always satisfied.
The computed inverse satisfies the forward error bound

$$|X - A^{-1}| \leq c(n)\ \epsilon\ |A^{-1}|\ |A|\ |X|.$$

See Du Croz and Higham [2].

6.3 Timing

The number of real floating-point operations required to compute the inverse is roughly $(1/3)n^3$ if A is real, and $(4/3)n^3$ if A is complex.

Example 1: Calculation of the Inverse of a General Matrix

This example program shows how `nag_gen_mat_inv` is used to calculate the inverse of a general real matrix. It also shows how `nag_gen_mat_inv` is used to estimate the condition number by using the optional argument `rcond`.

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_mat_inv_ex01

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

  ! .. Use Statements ..
  USE nag_mat_inv, ONLY : nag_gen_mat_inv
  USE nag_examples_io, ONLY : nag_std_out, nag_std_in
  USE nag_write_mat, ONLY : nag_write_gen_mat
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, n
  REAL (wp) :: rcond
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: a(:,:)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) 'Example Program Results for nag_mat_inv_ex01'
  WRITE (nag_std_out,*)

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

  ALLOCATE (a(n,n))            ! Allocate storage

  READ (nag_std_in,*) (a(i,:),i=1,n)

  CALL nag_gen_mat_inv(a,rcond=rcond)

  CALL nag_write_gen_mat(a,format='F10.4',title='Inverse matrix')

  WRITE (nag_std_out,*)
  WRITE (nag_std_out,'(1X,A,1PE10.2)') 'Estimated condition number = ', &
   1.0_wp/rcond

  DEALLOCATE (a)               ! Deallocate storage

END PROGRAM nag_mat_inv_ex01
```

2 Program Data

```
Example Program Data for nag_mat_inv_ex01
  4                           :Value of n
  1.80   2.88   2.05  -0.89
  5.25  -2.95  -0.95  -3.80
  1.58  -2.69  -2.90  -1.04
 -1.11  -0.66  -0.59   0.80   :End of matrix A
```

3 Program Results

```
 Example Program Results for nag_mat_inv_ex01

 Inverse matrix
      1.7720    0.5757    0.0843    4.8155
     -0.1175   -0.4456    0.4114   -1.7126
      0.1799    0.4527   -0.6676    1.4824
      2.4944    0.7650   -0.0360    7.6119

 Estimated condition number =   1.41E+02
```

Example 2: Calculation of the Inverse of a General Matrix Previously Factorized

This example program shows how `nag_gen_mat_inv_fac` is used to calculate the inverse of a general complex matrix, with the matrix previously factorized using `nag_gen_lin_fac`.

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_mat_inv_ex02

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

  ! .. Use Statements ..
  USE nag_mat_inv, ONLY : nag_gen_mat_inv_fac
  USE nag_gen_lin_sys, ONLY : nag_gen_lin_fac
  USE nag_examples_io, ONLY : nag_std_out, nag_std_in
  USE nag_write_mat, ONLY : nag_write_gen_mat
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, n
  ! .. Local Arrays ..
  INTEGER, ALLOCATABLE :: pivot(:)
  COMPLEX (wp), ALLOCATABLE :: a(:,:)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) 'Example Program Results for nag_mat_inv_ex02'
  WRITE (nag_std_out,*)

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

  ALLOCATE (a(n,n),pivot(n))    ! Allocate storage

  READ (nag_std_in,*) (a(i,:),i=1,n)

  CALL nag_gen_lin_fac(a,pivot)

  CALL nag_write_gen_mat(a,format='F7.4',title='Factorized matrix')

  WRITE (nag_std_out,*)
  WRITE (nag_std_out,*) 'Pivotal sequence (pivot)'
  WRITE (nag_std_out,'(2X,10I4:)') pivot
  WRITE (nag_std_out,*)

  CALL nag_gen_mat_inv_fac(a,pivot)

  CALL nag_write_gen_mat(a,format='F7.4',title= &
   'The inverse - using the factorized matrix')

  DEALLOCATE (a,pivot)          ! Deallocate storage

END PROGRAM nag_mat_inv_ex02
```

2 Program Data

```
Example Program Data for nag_mat_inv_ex02
  4                           :Value of n
 (-1.34, 2.55) ( 0.28, 3.17) (-6.39,-2.20) ( 0.72,-0.92)
 (-0.17,-1.41) ( 3.31,-0.15) (-0.15, 1.34) ( 1.29, 1.38)
 (-3.29,-2.39) (-1.91, 4.42) (-0.14,-1.35) ( 1.72, 1.35)
 ( 2.41, 0.39) (-0.56, 1.47) (-0.83,-0.69) (-1.96, 0.67)  :End of matrix A
```

3 Program Results

```
 Example Program Results for nag_mat_inv_ex02

 Factorized matrix
   (-3.2900,-2.3900) (-1.9100, 4.4200) (-0.1400,-1.3500) ( 1.7200, 1.3500)
   ( 0.2376, 0.2560) ( 4.8952,-0.7114) (-0.4623, 1.6966) ( 1.2269, 0.6190)
   (-0.1020,-0.7010) (-0.6691, 0.3689) (-5.1414,-1.1300) ( 0.9983, 0.3850)
   (-0.5359, 0.2707) (-0.2040, 0.8601) ( 0.0082, 0.1211) ( 0.1482,-0.1252)

 Pivotal sequence (pivot)
     3   2   3   4

 The inverse - using the factorized matrix
   ( 0.0757,-0.4324) ( 1.6512,-3.1342) ( 1.2663, 0.0418) ( 3.8181, 1.1195)
   (-0.1942, 0.0798) (-1.1900,-0.1426) (-0.2401,-0.5889) (-0.0101,-1.4969)
   (-0.0957,-0.0491) ( 0.7371,-0.4290) ( 0.3224, 0.0776) ( 0.6887, 0.7891)
   ( 0.3702,-0.5040) ( 3.7253,-3.1813) ( 1.7014, 0.7267) ( 3.9367, 3.3255)
```

Example 3: Calculation of the Inverse of a Symmetric Positive Definite Matrix

This example program shows how `nag_sym_mat_inv` is used to calculate the inverse of a real symmetric positive definite matrix. It also shows how `nag_sym_mat_inv` is used to estimate the condition number by using the optional argument `rcond`.

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_mat_inv_ex03

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

  ! .. Use Statements ..
  USE nag_mat_inv, ONLY : nag_key_pos, nag_sym_mat_inv
  USE nag_examples_io, ONLY : nag_std_out, nag_std_in
  USE nag_write_mat, ONLY : nag_write_tri_mat
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, n
  REAL (wp) :: rcond
  CHARACTER (1) :: uplo
  ! .. Local Arrays ..
  REAL (wp), ALLOCATABLE :: a(:,:)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) 'Example Program Results for nag_mat_inv_ex03'
  WRITE (nag_std_out,*)

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

  ALLOCATE (a(n,n))            ! Allocate storage

  SELECT CASE (uplo)
  CASE ('L','l')
    DO i = 1, n
      READ (nag_std_in,*) a(i,:i)
    END DO
  CASE ('U','u')
    DO i = 1, n
      READ (nag_std_in,*) a(i,i:)
    END DO
  END SELECT

  CALL nag_sym_mat_inv(nag_key_pos,uplo,a,rcond=rcond)

  CALL nag_write_tri_mat(uplo,a,format='F10.4',title='Inverse matrix')

  WRITE (nag_std_out,*)
  WRITE (nag_std_out,'(1X,A,1PE10.2)') 'Estimated condition number = ', &
   1.0_wp/rcond
```

```
    DEALLOCATE (a)                ! Deallocate storage

  END PROGRAM nag_mat_inv_ex03
```

2 Program Data

```
Example Program Data for nag_mat_inv_ex03
  4                         : Value of n
  'L'                       : Value of uplo
  4.16
 -3.12   5.03
  0.56  -0.83   0.76
 -0.10   1.18   0.34   1.18   : End of Matrix A
```

3 Program Results

```
 Example Program Results for nag_mat_inv_ex03

 Inverse matrix
      0.6995
      0.7769    1.4239
      0.7508    1.8255    4.0688
     -0.9340   -1.8841   -2.9342    3.4978

 Estimated condition number =   9.73E+01
```

Example 4: Calculation of the Inverse of a Hermitian Indefinite Matrix Previously Factorized

This example program shows how `nag_sym_mat_inv_fac` is used to calculate the inverse of a complex Hermitian indefinite matrix, using packed storage, with the matrix previously factorized using `nag_sym_lin_fac`.

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_mat_inv_ex04

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

  ! .. Use Statements ..
  USE nag_mat_inv, ONLY : nag_sym_mat_inv_fac
  USE nag_examples_io, ONLY : nag_std_out, nag_std_in
  USE nag_write_mat, ONLY : nag_write_tri_mat
  USE nag_sym_lin_sys, ONLY : nag_key_herm, nag_sym_lin_fac
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, j, n
  CHARACTER (1) :: uplo
  ! .. Local Arrays ..
  INTEGER, ALLOCATABLE :: pivot(:)
  COMPLEX (wp), ALLOCATABLE :: a(:)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) 'Example Program Results for nag_mat_inv_ex04'
  WRITE (nag_std_out,*)

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

  ALLOCATE (a((n*(n+1))/2),pivot(n)) ! Allocate storage

  SELECT CASE (uplo)
  CASE ('L','l')
    DO i = 1, n
      READ (nag_std_in,*) (a(i+((2*n-j)*(j-1))/2),j=1,i)
    END DO
  CASE ('U','u')
    DO i = 1, n
      READ (nag_std_in,*) (a(i+(j*(j-1))/2),j=i,n)
    END DO
  END SELECT

  CALL nag_sym_lin_fac(nag_key_herm,uplo,a,pivot)

  CALL nag_write_tri_mat(uplo,a,format='F7.4',title='Factorized matrix')

  WRITE (nag_std_out,*)
  WRITE (nag_std_out,*) 'Pivotal sequence (pivot)'
  WRITE (nag_std_out,'(2X,10I4:)') pivot
```

```
      WRITE (nag_std_out,*)

      CALL nag_sym_mat_inv_fac(nag_key_herm,uplo,a,pivot)

      CALL nag_write_tri_mat(uplo,a,format='F7.4',title= &
       'The inverse - using the factorized matrix')

      DEALLOCATE (a,pivot)         ! Deallocate storage

    END PROGRAM nag_mat_inv_ex04
```

2 Program Data

```
Example Program Data for nag_mat_inv_ex04
  4                           : Value of n
  'L'                         : Value of uplo
 (-1.36, 0.00)
 ( 1.58,-0.90) (-8.87, 0.00)
 ( 2.21, 0.21) (-1.84, 0.03) (-4.63, 0.00)
 ( 3.91,-1.50) (-1.78,-1.18) ( 0.11,-0.11) (-1.84, 0.00) : End of Matrix A
```

3 Program Results

```
 Example Program Results for nag_mat_inv_ex04

 Factorized matrix
   (-1.3600, 0.0000)
   ( 3.9100,-1.5000) (-1.8400, 0.0000)
   ( 0.3100, 0.0433) ( 0.5637, 0.2850) (-5.4176, 0.0000)
   (-0.1518, 0.3743) ( 0.3397, 0.0303) ( 0.2997, 0.1578) (-7.1028, 0.0000)

 Pivotal sequence (pivot)
    -4  -4   3   4

 The inverse - using the factorized matrix
   ( 0.0826, 0.0000)
   (-0.0335, 0.0440) (-0.1408, 0.0000)
   ( 0.0603,-0.0105) ( 0.0422,-0.0222) (-0.2007, 0.0000)
   ( 0.2391,-0.0926) ( 0.0304, 0.0203) ( 0.0982,-0.0635) ( 0.0073,-0.0000)
```

Example 5: Calculation of the Inverse of a Triangular Matrix

This example program shows how `nag_tri_mat_inv` is used to calculate the inverse of a complex triangular matrix. It also shows how `nag_tri_mat_inv` is used to estimate the condition number by using the optional argument `rcond`.

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_mat_inv_ex05

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

  ! .. Use Statements ..
  USE nag_mat_inv, ONLY : nag_tri_mat_inv
  USE nag_examples_io, ONLY : nag_std_out, nag_std_in
  USE nag_write_mat, ONLY : nag_write_tri_mat
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  ! .. Local Scalars ..
  INTEGER :: i, j, n
  REAL (wp) :: rcond
  CHARACTER (1) :: uplo
  ! .. Local Arrays ..
  COMPLEX (wp), ALLOCATABLE :: a(:,:)
  ! .. Executable Statements ..
  WRITE (nag_std_out,*) 'Example Program Results for nag_mat_inv_ex05'
  WRITE (nag_std_out,*)

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

  ALLOCATE (a(n,n))            ! Allocate storage

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

  CALL nag_tri_mat_inv(uplo,a,rcond=rcond)

  CALL nag_write_tri_mat(uplo,a,format='F7.4',title= &
   'Inverse matrix (triangular)')

  WRITE (nag_std_out,*)
  WRITE (nag_std_out,'(1X,A,1PE10.2)') 'Estimated condition number = ', &
   1.0_wp/rcond
```

```
      DEALLOCATE (a)              ! Deallocate storage

    END PROGRAM nag_mat_inv_ex05
```

2 Program Data

```
Example Program Data for nag_mat_inv_ex05
  4                           : Value of n
  'L'                         : Value of uplo
 ( 4.78, 4.56)
 ( 2.00,-0.30) (-4.11, 1.25)
 ( 2.89,-1.34) ( 2.36,-4.25) ( 4.15, 0.80)
 (-1.89, 1.15) ( 0.04,-3.69) (-0.02, 0.46) ( 0.33,-0.26)  : End of matrix A
```

3 Program Results

```
 Example Program Results for nag_mat_inv_ex05

 Inverse matrix (triangular)
   ( 0.1095,-0.1045)
   ( 0.0582,-0.0411) (-0.2227,-0.0677)
   ( 0.0032, 0.1905) ( 0.1538,-0.2192) ( 0.2323,-0.0448)
   ( 0.7602, 0.2814) ( 1.6184,-1.4346) ( 0.1289,-0.2250) ( 1.8697, 1.4731)

 Estimated condition number =   3.74E+01
```

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_mat_inv_ex06`

Computes the inverse of a complex general matrix.

`nag_mat_inv_ex07`

Computes the inverse of a real general matrix, previously factorized.

`nag_mat_inv_ex08`

Computes the inverse of a complex Hermitian positive definite matrix.

`nag_mat_inv_ex09`

Computes the inverse of a real symmetric positive matrix, using packed storage.

`nag_mat_inv_ex10`

Computes the inverse of a complex Hermitian positive definite matrix, using packed storage.

`nag_mat_inv_ex11`

Computes the inverse of a real symmetric indefinite matrix.

`nag_mat_inv_ex12`

Computes the inverse of a complex Hermitian indefinite matrix.

`nag_mat_inv_ex13`

Computes the inverse of a complex symmetric indefinite matrix.

`nag_mat_inv_ex14`

Computes the inverse of a real symmetric indefinite matrix, using packed storage.

`nag_mat_inv_ex15`

Computes the inverse of a complex Hermitian indefinite matrix, using packed storage.

`nag_mat_inv_ex16`

Computes the inverse of a complex symmetric indefinite matrix, using packed storage.

`nag_mat_inv_ex17`

Computes the inverse of a real symmetric positive definite matrix, previously factorized.

`nag_mat_inv_ex18`

Computes the inverse of a complex Hermitian positive definite matrix, previously factorized.

`nag_mat_inv_ex19`

Computes the inverse of a real symmetric positive definite matrix, previously factorized, using packed storage.

`nag_mat_inv_ex20`

Computes the inverse of a complex Hermitian positive definite matrix, previously factorized, using packed storage.

`nag_mat_inv_ex21`

Computes the inverse of a real symmetric indefinite matrix, previously factorized.

`nag_mat_inv_ex22`

Computes the inverse of a complex Hermitian indefinite matrix, previously factorized.

`nag_mat_inv_ex23`

Computes the inverse of a complex symmetric indefinite matrix, previously factorized.

`nag_mat_inv_ex24`

Computes the inverse of a real symmetric indefinite matrix, previously factorized, using packed storage.

`nag_mat_inv_ex25`

Computes the inverse of a complex symmetric indefinite matrix, previously factorized, using packed storage.

`nag_mat_inv_ex26`

Computes the inverse of a real triangular matrix.

`nag_mat_inv_ex27`

Computes the inverse of a real triangular matrix, using packed storage.

`nag_mat_inv_ex28`

Computes the inverse of a complex triangular matrix, using packed storage.

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] Du Croz J J and Higham N J (1992) Stability of methods for matrix inversion *IMA J. Numer. Anal.* **12** 1–19