Chapter 4

Matrix and Vector Operations

1 Scope of the Chapter

This chapter provides procedures for matrix and vector operations.

This chapter (and Chapters 5 and 6) can handle general matrices and matrices with special structure. Using the special structure of matrices offers possibilities for greater efficiency, more economical storage and increased reliability.

All the procedures in this chapter are generic procedures which can handle either real or complex data.

2 Available Modules

Module 4.1: `nag_mat_norm` — **Norms of a matrix**

Provides procedures to compute the 1-norm, the ∞-norm, the Frobenius (Euclidean) norm or the element of largest absolute value, of a real or complex matrix. It caters for different types of matrices and different storage schemes.

3 Storage of Matrices

In this section we assume that A is a matrix and `a` is the corresponding argument. For symmetric or triangular matrices it assumed that the argument `uplo` is used to specify that the elements of either the upper or the lower triangle are referenced.

3.1 Symmetric Matrices

There are two storage schemes for the symmetric or Hermitian matrix A: conventional storage or packed storage. The choice is determined by the rank of the corresponding argument `a`.

Conventional Storage

`a` is a rank-2 array, of shape (n,n). Matrix element a_{ij} is stored in `a`(i,j). Only the elements of either the upper or the lower triangle need be stored, as specified by the argument `uplo`; the remaining elements of `a` need not be set.

This storage scheme is more straightforward and carries less risk of user error than packed storage; on some machines it may result in more efficient execution. It requires almost twice as much memory as packed storage, although the other triangle of `a` may be used to store other data.

Packed storage

`a` is a rank-1 array of shape $(n(n+1)/2)$. The elements of either the upper or the lower triangle of A, as specified by `uplo`, are packed by *columns* into contiguous elements of `a`.

Packed storage is more economical in use of memory than conventional storage, but may result in less efficient execution on some machines.

The details of packed storage are as follows:

- if `uplo` = `'u'` or `'U'`, a_{ij} is stored in `a`$(i+j(j-1)/2)$, for $i \leq j$;
- if `uplo` = `'l'` or `'L'`, a_{ij} is stored in `a`$(i+(2n-j)(j-1)/2)$, for $i \geq j$.

For example

`uplo`	Hermitian Matrix A	Packed storage in array `a`
`'u'` or `'U'`	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ \overline{a}_{12} & a_{22} & a_{23} & a_{24} \\ \overline{a}_{13} & \overline{a}_{23} & a_{33} & a_{34} \\ \overline{a}_{14} & \overline{a}_{24} & \overline{a}_{34} & a_{44} \end{pmatrix}$	$a_{11} \quad \underbrace{a_{12}\, a_{22}} \quad \underbrace{a_{13}\, a_{23}\, a_{33}} \quad \underbrace{a_{14}\, a_{24}\, a_{34}\, a_{44}}$
`'l'` or `'L'`	$\begin{pmatrix} a_{11} & \overline{a}_{21} & \overline{a}_{31} & \overline{a}_{41} \\ a_{21} & a_{22} & \overline{a}_{32} & \overline{a}_{42} \\ a_{31} & a_{32} & a_{33} & \overline{a}_{43} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$	$\underbrace{a_{11}\, a_{21}\, a_{31}\, a_{41}} \quad \underbrace{a_{22}\, a_{32}\, a_{42}} \quad \underbrace{a_{33}\, a_{43}} \quad a_{44}$

Note that for symmetric matrices, packing the upper triangle by columns is equivalent to packing the lower triangle by rows; packing the lower triangle by columns is equivalent to packing the upper triangle by rows. For Hermitian matrices, packing the upper triangle by columns is equivalent to packing the conjugate of the lower triangle by rows; packing the lower triangle by columns is equivalent to packing the conjugate of the upper triangle by rows.

3.2 Triangular Matrices

There are two storage schemes for the triangular matrix A: conventional storage or packed storage. The choice is determined by the rank of the corresponding argument `a`.

Conventional Storage

`a` is a rank-2 array, of shape (n,n). Matrix element a_{ij} is stored in `a(`i,j`)`. If A is upper triangular, only the elements of the upper triangle ($i \leq j$) need be stored; if A is lower triangular, only the elements of the lower triangle ($i \geq j$) need be stored; the remaining elements of `a` need not be set.

This storage scheme is more straightforward and carries less risk of user error than packed storage; on some machines it may result in more efficient execution. It requires almost twice as much memory as packed storage, although the other triangle of `a` may be used to store other data.

Packed storage

`a` is a rank-1 array of shape $(n(n+1)/2)$. The elements of either the upper or the lower triangle of A, as specified by `uplo`, are packed by *columns* into contiguous elements of `a`.

Packed storage is more economical in use of memory than conventional storage, but may result in less efficient execution on some machines.

The details of packed storage are as follows:

- if `uplo` = `'u'` or `'U'`, a_{ij} is stored in `a(`$i + j(j-1)/2$`)`, for $i \leq j$;
- if `uplo` = `'l'` or `'L'`, a_{ij} is stored in `a(`$i + (2n-j)(j-1)/2$`)`, for $i \geq j$.

For example

`uplo`	Triangular Matrix A	Packed storage in array `a`
`'u'` or `'U'`	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22} & a_{23} & a_{24} \\ & & a_{33} & a_{34} \\ & & & a_{44} \end{pmatrix}$	$a_{11} \quad \underbrace{a_{12}\, a_{22}} \quad \underbrace{a_{13}\, a_{23}\, a_{33}} \quad \underbrace{a_{14}\, a_{24}\, a_{34}\, a_{44}}$
`'l'` or `'L'`	$\begin{pmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$	$\underbrace{a_{11}\, a_{21}\, a_{31}\, a_{41}} \quad \underbrace{a_{22}\, a_{32}\, a_{42}} \quad \underbrace{a_{33}\, a_{43}} \quad a_{44}$

Unit triangular matrices

A *unit* triangular matrix is a triangular matrix whose diagonal elements are known to be 1. Some procedures have an optional argument `unit_diag` which can be used to specify that the matrix is unit triangular, and then the diagonal elements do not need to be stored; the storage of the other elements of the matrix is not affected.

3.3 Square Banded Matrices

The following storage scheme is used 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 $\max(j - k_u, 1) \leq i \leq \min(j + k_l, n)$.

For example

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

3.4 Symmetric Banded Matrices

The following storage scheme is used 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'`	$\begin{pmatrix} a_{11} & \overline{a}_{21} & \overline{a}_{31} & & \\ a_{21} & a_{22} & \overline{a}_{32} & \overline{a}_{42} & \\ a_{31} & a_{32} & a_{33} & \overline{a}_{43} & \overline{a}_{53} \\ & a_{42} & a_{43} & a_{44} & \overline{a}_{54} \\ & & a_{53} & a_{54} & a_{55} \end{pmatrix}$	$\begin{matrix} a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{21} & a_{32} & a_{43} & a_{54} & * \\ a_{31} & a_{42} & a_{53} & * & * \end{matrix}$

3.5 Triangular Banded Matrices

The following storage scheme is used for the triangular 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'`	$\begin{pmatrix} a_{11} & a_{12} & a_{13} & & \\ & a_{22} & a_{23} & a_{24} & \\ & & a_{33} & a_{34} & a_{35} \\ & & & a_{44} & a_{45} \\ & & & & a_{55} \end{pmatrix}$	$\begin{matrix} * & * & a_{13} & a_{24} & a_{35} \\ * & a_{12} & a_{23} & a_{34} & a_{45} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \end{matrix}$
`'l'` or `'L'`	$\begin{pmatrix} a_{11} & & & & \\ a_{21} & a_{22} & & & \\ a_{31} & a_{32} & a_{33} & & \\ & a_{42} & a_{43} & a_{44} & \\ & & a_{53} & a_{54} & a_{55} \end{pmatrix}$	$\begin{matrix} a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{21} & a_{32} & a_{43} & a_{54} & * \\ a_{31} & a_{42} & a_{53} & * & * \end{matrix}$

Unit triangular matrices

A *unit* triangular banded matrix is a triangular banded matrix whose diagonal elements are known to be 1. Some procedures have an optional argument `unit_diag` which can be used to specify that the matrix is unit triangular banded, and then the diagonal elements do not need to be stored; the storage of the other elements of the matrix is not affected.