Chapter 5: Arrays

Fortran handles arrays easily compared to other low-level languages. For example it is very flexible when it comes to indexing or accessing elements of arrays. Therefore, it is an ideal choice for coding vector and matrix operations. By default, the initial index of an array is 1, but this can be easily changed.

It is important to note that ***Fortran stores array data by column***, often referred to as ***column-major.*** By default, when writing an array without formatting the leftmost column from top to bottom is written, then the next leftmost from top to bottom, and so on.

5.1 Basics

Some common functions that operate on arrays and their interpretations are

`size(A)`	number of elements in A
`transpose(A)`	the transpose of A
`maxval(A)`	maximum value in A
`minval(A)`	minimum value in A
`matmul(A,B)`	matrix product $A \times B$
`dot_product(a,b)`	the dot product $a \cdot b$
`sum(A)`	sum on elements in A
`product(A)`	product of elements in A

The following program introduces some basics of arrays.

array.f95

```fortran
program array
implicit none
   integer :: i, j, n
   integer, dimension(5) :: A
   integer :: B(5)
   real, dimension(-3:1) :: C
   real :: D(-2:2)
   integer :: E(2, 2), F(-1:1,2), eye(3,3)

   character(1024) :: frmt

   A = (/ 1, 2, 3, 4, 5 /) ! explicit assignment
   write(*,'(a,5(i0,1x))') 'A = ',A ! write as space-delimited row

   B=(/(2*i,i=1,size(B))/) ! implicit do loop in explicit constructor
   write(*,'(a)') '(as a column) B ='
   write(*,'(i0)') B ! write as column

   do i=1,size(B)
      B(i)=2*i
   end do
   write(*,'(a)') '(as a row) B ='
   write(*,'(5(i0,x))') B ! write as row

   write(*,'(a,i0)'), 'A dot B = ',dot_product(A,B) ! dot product

   C = (/ 1., 3., 5., 4., 2. /)
```

```
   write(*,'(a,3(f0.0,1x))') 'C(-2:0) = ',C(-2:0) ! middle 3 elements as space-delimited row

   forall(i=-2:2) D(i)=real(i)**2. ! forall declaration, more concise than do loop
   write(*,'(a,5(f0.0,1x))') 'D(:) = ', D(:)
   write(*,'(a,f0.0)'),'maxval(D) = ',maxval(D)
   write(*,'(a,f0.0)'),'minval(D) = ',minval(D)
   write(*,'(a,i0)'),'lbound(D) = ',lbound(D)
   write(*,'(a,i0)'),'ubound(D) = ',ubound(D)

   E = reshape((/1,2,3,4/),(/2,2/))
   write(*,*) '(unformatted) E = ',E
   write(*,*) '(formatted) E = '
   do i=1,2
      write(*,'(2(i0,1x),a,i0,a)') E(i,:),' (row ',i,')'
   end do
   write(*,*) '(formatted) E = '
   do i=1,2
      write(*,'(2(i0,1x),a,i0,a)') E(:,i),' (col ',i,')'
   end do

   F = reshape((/1,2,3,4,5,6/),(/3,2/))
   write(*,'(a)') 'F = '
   do i=-1,1
      write(*,'(2(i0,1x))') F(i,:)
   end do
   write(*,'(a)') 'F = '
   write(*,'(2(i0,1x))') transpose(F)

   n=3
   forall(i=1:n,j=1:n) eye(i,j)=(i/j)*(j/i) ! trick for creating identity matrix
   write(*,'(a)') 'eye = '
   write(frmt,'(a,i0,a)') '(',n,'(i0,1x))' ! write to frmt string
   write(*,frmt) eye ! write eye with frmt string
end program array
```

- An array may be declared either with the `dimension` attribute following the data type declaration or by appending the array index range(s) to the variable name. For example, `integer, dimension(5) :: A` or `integer :: A(5)` declares an an array of 5 integers; the first integer is in `A(1)` down to the last in `A(5)`. You can specify an index range other than the default. For example, either `integer, dimemsion(-2:2) :: A` or `integer :: A(-2:2)` declare arrays of integers with the first element in `A(-2)` and the last element in `A(2)`. In general, you can declare an array with arbitrary data type, dimension and indexing with `DATATYPE :: ARRAYNAME(MIN1:MAX1,MIN2:MAX2,...,MINN:MAXN)`.

- There are a number of ways of assigning values to an array. To assign values explicitly, use the array constructor `(/ ... /)`; for example, if `A` is an array with size 5, use `A = (/ 1, 2, 3, 4, 5 /)`. The `reshape` command is useful for explicitly assigning values to a multi-dimensional array. Array assignments can also be made one element at a time; for example `B(i)=2*i` assigns a value of `2*i` to the ith element in `B`. This should be used in conjunction with `do` loops. As a concise alternative, `forall` statements can be used to assign values one element at a time; for example, if `eye` is a 3×3 matrix `eye=forall(i=1:3,j=1:3) eye(i,j)=(i/j)*(j/i)` creates the identity matrix. This last example is a bit tricky since it relies on the fact that integer division is rounded down, i.e. the only time that `(i/j)*(j/i)` computes to `1` is if `i=j`, otherwise it computes to `0`.

- By default, `write(*,*)A` will write the elements of `A` in column-major order; that is, if `A` is an $n \times n$ matrix indexed from 1 to n in both dimensions `write(*,*)A` prints the list `A(1,1)`, `A(2,1)`,`...`, A(n,1), A(1,2), A(2,2),. . ., A(n,2),. . ., A(1,n), A(2,n),. . ., A(n,n). For better output, do loops or formatting should be used.

- Blocks of arrays can be accessed directly by specifying the desired indices. For example, If A is a 3×3 array indexed from 1 to n in both dimensions, the 2×2 minor matrix in the upper left of A is A(1:2,1:2) or the last column of A is A(:,3).

array - commands and output

```
 gfortran array.f95 -o array
 ./array
A = 1 2 3 4 5
(as a column) B =
2
4
6
8
10
(as a row) B =
2 4 6 8 10
A dot B = 110
C(-2:0) = 3. 5. 4.
D(:) = 4. 1. 0. 1. 4.
maxval(D) = 4.
minval(D) = 0.
lbound(D) = -2
ubound(D) = 2
 (unformatted) E =            1            2            3            4
 (formatted) E =
1 3  (row 1)
2 4  (row 2)
 (formatted) E =
1 2  (col 1)
3 4  (col 2)
F =
1 4
2 5
3 6
F =
1 4
2 5
3 6
eye =
1 0 0
0 1 0
0 0 1
```

Sometimes you will not know the dimensions of an array at declaration. For this, you can declare the array with the attribute allocatble and a deferred shape and later allocate memory for the array. After you no longer need an allocated arrray, you can use deallocate to free the memory that it is using.

The following program demonstrates how to allocate arrays.

allocate.f95

```
program allocation
```

```
implicit none
   real, allocatable :: A(:), B(:,:)
   integer :: i, j

   allocate(A(1:5),B(3,3))
   A = (/ 1., 2., 3., 4., 5. /)
   write(*,*) 'A=',A

   forall (i=1:3,j=1:3) B(i,j)=i+j
   write(*,*) 'B='
   do i=1,3
      write(*,*) B(i,:)
   end do
   deallocate(A,B)

end program allocation
```

- To declare an array with a deferred shape, use only commas and semi-colons to assign indices. For example, to declare a 1-dimensional allocatable array `A` of integers, use `integer, allocatable :: A(:)` or for a 2-dimensional array `B` of integers, use `integer, allocatable :: B(:,:)`.
- With the allocate function, array indices can be specified as usual.

allocate - commands and output

```
gfortran allocate.f95 -o allocate
./allocate
A=    1.00000000        2.00000000        3.00000000        4.00000000        5.00000000
B=
   2.00000000        3.00000000        4.00000000
   3.00000000        4.00000000        5.00000000
   4.00000000        5.00000000        6.00000000
```

Exercise

1. Consider solving the linear system $Ax = b$ for x where

$$A = \left(\frac{1}{i+j-1}\right)_{i,j=1}^{5} = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} \\ \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} \end{bmatrix} \quad \text{and} \quad b = \begin{bmatrix} \frac{137}{60} \\ \frac{29}{20} \\ \frac{153}{140} \\ \frac{360}{407} \\ \frac{299}{401} \end{bmatrix}.$$

Since A is nonsingular, we could use the formula $x = A^{-1}b$ to solve for x. In practice, this is never done because there are more efficient methods. One method relies on factoring A into the product of a lower triangular matrix L and upper triangular matrix U. For A above, the LU-factorization of A is $A = LU$ where

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 1 & 0 & 0 & 0 \\ \frac{1}{3} & 1 & 1 & 0 & 0 \\ \frac{1}{4} & \frac{9}{10} & \frac{3}{2} & 1 & 0 \\ \frac{1}{5} & \frac{4}{5} & \frac{12}{7} & \frac{2}{1} & 1 \end{bmatrix} \quad \text{and} \quad U = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ 0 & \frac{1}{12} & \frac{1}{12} & \frac{3}{40} & \frac{1}{15} \\ 0 & 0 & \frac{1}{180} & \frac{1}{120} & \frac{1}{105} \\ 0 & 0 & 0 & \frac{1}{2800} & \frac{1}{1400} \\ 0 & 0 & 0 & 0 & \frac{1}{44100} \end{bmatrix}.$$

Your task is to write a program that solves for x above using the LU-factorization of A. Report the solution x that you obtain as well as number of basic floating point operations that your program requires to solve for x and the number of storage locations required by your program.