Programming and Modelling (week 39)

C. Thieulot

Institute of Earth Sciences

September 2017

Arrays

We have seen how to define arrays in Fortran:

```
real, dimension(1000) :: tab

integer, dimension(1000,100) :: bigtab
```

Arrays

We have seen how to define arrays in Fortran:

```
real, dimension(1000) :: tab

integer, dimension(1000,100) :: bigtab
```

→ recurrent problem: the size of arrays has to be defined at compilation, but often their size is provided by the user at runtime.

Arrays

We have seen how to define arrays in Fortran:

```
real, dimension(1000) :: tab

integer, dimension(1000,100) :: bigtab
```

→ recurrent problem: the size of arrays has to be defined at compilation, but often their size is provided by the user at runtime.

Concretely:

```
program opla
implicit none
real(8), dimension(123) :: xcoordinates

write(6,*) 'Enter nb of points'
read(5,x) npts
```

Allocatable arrays

There is a solution : **allocatable arrays**

Allocatable arrays

There is a solution : **allocatable arrays**

Example

It is a three-step process:

1. declare an array, without specifying its size
2. compute/read its size
3. allocate the array to the correct size
   (book the memory to store the array)

Example

It is a three-step process:

1. declare an array, without specifying its size
2. compute/read its size
3. allocate the array to the correct size
   (book the memory to store the array)

```
integer n
real(8), dimension(:), allocatable :: array

write(6,*) 'Enter size of array'
read(5,*) n

allocate(array(n))
```

Example

It is a three-step process:

1. declare an array, without specifying its size
2. compute/read its size
3. allocate the array to the correct size
   (book the memory to store the array)

```
integer n
real(8), dimension(:), allocatable :: array

write(6,*) 'Enter size of array'
read(5,*) n

allocate(array(n))
```

```
integer n,m
real(8), dimension(:,:), allocatable :: largearray

write(6,*) 'Enter nb of lines of array'
read(5,*) n

write(6,*) 'Enter nb of columns of array'
read(5,*) m

allocate(largearray(n,m))
```

To be clear:

- to declare a scalar:
```
integer :: npts
real :: x0
```
- to declare a fixed-size array:
```
integer, dimension(100) :: mmm
real, dimension(33) :: xcoords
```
- to declare a variable-size array:
```
integer, dimension(:), allocatable :: mnp
real, dimension(:), allocatable :: values
```

Example 2

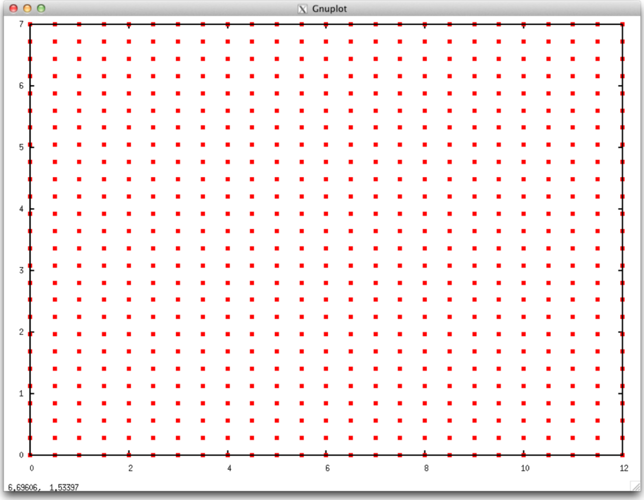
```
program example
implicit none
integer :: n
real,dimension(:),allocatable :: xcoords
real,dimension(:),allocatable :: ycoords

write(6,*) 'enter number of points'
read(5,*) n

allocate(xcoords(n))
allocate(ycoords(n))

call random_number(xcoords)
call random_number(ycoords)


!
! do something will the coordinates
!


deallocate(xcoords)
deallocate(ycoords)

end program
```

Example 3

Generate a regular grid of nnx $\times$ nny points in $[0 : 12] \times [0 : 7]$

Example 3

Generate a regular grid of nnx × nny points in $[0:12] \times [0:7]$

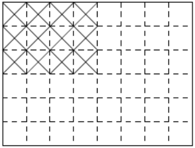


Example 3

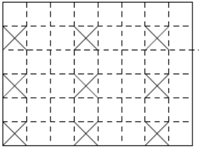
```
program example
implicit none
integer :: i,j,ip,counter
integer :: nnx,nny,np
real :: Lx,Ly,dx,dy
real,dimension(:),allocatable :: gridx
real,dimension(:),allocatable :: gridy

Lx=12
Ly=7

write(6,*) 'how many points in the x direction (nnx) ?'
read(5,*) nnx
write(6,*) 'how many points in the y direction (nny) ?'
read(5,*) nny

np=nnx*nny

allocate(gridx(np))
allocate(gridy(np))

write(6,*) 'total number of points: ',np

dx=Lx/real(nnx-1)
dy=Ly/real(nny-1)

counter=0
do i=1,nnx
   do j=1,nny
      counter=counter+1
      gridx(counter)=(i-1)*dx
      gridy(counter)=(j-1)*dy
   end do
end do

open(unit=345,file='points.dat',action='write')
do ip=1,np
   write(345,*) gridx(ip),gridy(ip)
end do
close(345)

deallocate(gridx)
deallocate(gridy)

end program
```

Manipulating arrays

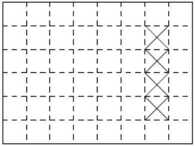
```
REAL, DIMENSION(1:6,1:8) :: P
```



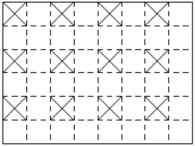
P(1:3,1:4)



P(2:6:2,1:7:3)



P(2:5,7), P(2:5,7:7)



P(1:6:2,1:8:2)

Intrinsic array functions

- minval,maxval
- shape
- size
- sum
- product

Intrinsic array functions - Example

```fortran
program example
implicit none

real, dimension(53) :: tab

call random_number(tab)

write(6,*) 'minimum value in tab ',minval(tab)
write(6,*) 'maximum value in tab ',maxval(tab)
write(6,*) 'shape of tab         ',shape(tab)
write(6,*) 'size of tab          ',size(tab)
write(6,*) 'sum of  all numbers in tab   ',sum(tab)
write(6,*) 'product of all numbers in tab',product(tab)

end program
```

Intrinsic array functions - Example

```fortran
program example
implicit none

real, dimension(53) :: tab

call random_number(tab)

write(6,*) 'minimum value in tab ',minval(tab)
write(6,*) 'maximum value in tab ',maxval(tab)
write(6,*) 'shape of tab         ',shape(tab)
write(6,*) 'size of tab          ',size(tab)
write(6,*) 'sum of  all numbers in tab   ',sum(tab)
write(6,*) 'product of all numbers in tab',product(tab)

end program
```

```
geogarfield ~/Desktop/UD/onderwijs/1320_PROGMOD/LECTURES (master *) $ ./a.out
 minimum value in tab    1.07835531E-02
 maximum value in tab   0.964987576
 shape of tab                   53
 size of tab                    53
 sum of  all numbers in tab      28.6039162
 product of all numbers in tab   1.31217255E-22
```

Intrinsic array functions - Example(2)

```
program example
implicit none

real, dimension(11,13) :: tab

call random_number(tab)

write(6,*) 'minimum value in tab ',minval(tab)
write(6,*) 'maximum value in tab ',maxval(tab)
write(6,*) 'shape of tab         ',shape(tab)
write(6,*) 'size of tab          ',size(tab)
write(6,*) 'sum of  all numbers in tab   ',sum(tab)
write(6,*) 'product of all numbers in tab',product(tab)

end program
```

Intrinsic array functions - Example(2)

```
program example
implicit none

real, dimension(11,13) :: tab

call random_number(tab)

write(6,*) 'minimum value in tab ',minval(tab)
write(6,*) 'maximum value in tab ',maxval(tab)
write(6,*) 'shape of tab         ',shape(tab)
write(6,*) 'size of tab          ',size(tab)
write(6,*) 'sum of  all numbers in tab    ',sum(tab)
write(6,*) 'product of all numbers in tab',product(tab)

end program
```

```
geogarfield ~/Desktop/UD/onderwijs/1320_PROGMOD/LECTURES (master *) $ ./a.out
 minimum value in tab    1.77552104E-02
 maximum value in tab   0.993430555
 shape of tab                   11          13
 size of tab                   143
 sum of  all numbers in tab      73.0963593
 product of all numbers in tab   0.00000000
```

Intrinsic array functions - Example(3)

```
program example
implicit none

real, dimension(11,13) :: tab

call random_number(tab)

write(6,*) 'average of values in in tab ',sum(tab)/size(tab)

end program
```

More intrinsic array functions

Fortran also offers very useful functions in linear algebra:

- dot_product
- matmul
- transpose

More intrinsic array functions - Example 1

Let us define $\boldsymbol{v}_1 = (2, -3, -1)$ and $\boldsymbol{v}_2 = (6, 3, 3)$
We then have the scalar product of these vectors: $\boldsymbol{v}_1 \cdot \boldsymbol{v}_2 = 0$.

More intrinsic array functions - Example 1

Let us define $\mathbf{v}_1 = (2, -3, -1)$ and $\mathbf{v}_2 = (6, 3, 3)$
We then have the scalar product of these vectors: $\mathbf{v}_1 \cdot \mathbf{v}_2 = 0$.

```
program example
implicit none

real, dimension(3) :: vect1
real, dimension(3) :: vect2
real :: prod_scal

vect1=(/2,-3,-1/)
vect2=(/6,3,3/)

prod_scal=vect1(1)*vect2(1)&
         +vect1(2)*vect2(2)&
         +vect1(3)*vect2(3)

write(6,*) 'scalar product is ',prod_scal

write(6,*) 'scalar product is ',dot_product(vect1,vect2)

end program
```

```
thebeast:progmod geogarfield$ ./a.out
 scalar product is    0.0000000
 scalar product is   0.0000000
```

More intrinsic array functions - Example 2

Let us consider two small matrices:

$$\boldsymbol{A} = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} \qquad \boldsymbol{B} = \begin{pmatrix} -1 & -2 \\ 4 & 1 \end{pmatrix}$$

We wish to compute $\boldsymbol{C} = \boldsymbol{A} \cdot \boldsymbol{B}$.

More intrinsic array functions - Example 2

Let us consider two small matrices:

$$\boldsymbol{A} = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} \qquad \boldsymbol{B} = \begin{pmatrix} -1 & -2 \\ 4 & 1 \end{pmatrix}$$

We wish to compute $\boldsymbol{C} = \boldsymbol{A} \cdot \boldsymbol{B}$.

```
program example
implicit none

real,dimension(2,2) :: matA
real,dimension(2,2) :: matB
real,dimension(2,2) :: matC

matA(1,1)=1.   ; matA(1,2)=3.
matA(2,1)=2.   ; matA(2,2)=4.

matB(1,1)=-1.  ; matB(1,2)=-2.
matB(2,1)=4.   ; matB(2,2)=1.

matC=matmul(matA,matB)

write(6,*) matC(1,1),matC(1,2)
write(6,*) matC(2,1),matC(2,2)

end program
```

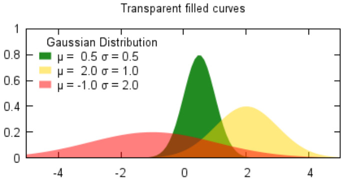
```
thebeast:progmod geogarfield$ ./a.out
   11.000000       1.0000000
   14.000000       0.0000000
```

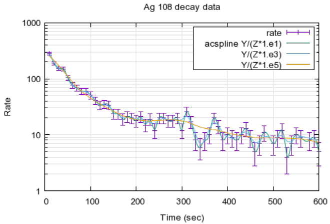
Gnuplot (1)

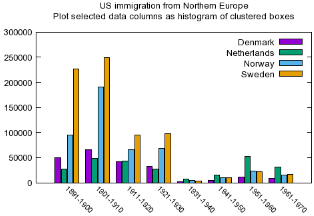
- Gnuplot is a portable command-line driven graphing utility for Linux, MS Windows, OSX, and many other platforms.
- The source code is copyrighted but freely distributed (i.e., you don't have to pay for it).
- It was originally created to allow scientists and students to visualize mathematical functions
- The official website is at this address: `http://gnuplot.info/`

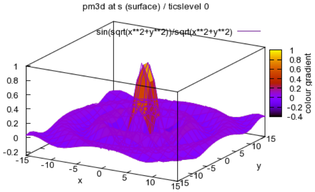
Gnuplot (2)

Here are a few examples of what can be done with gnuplot:









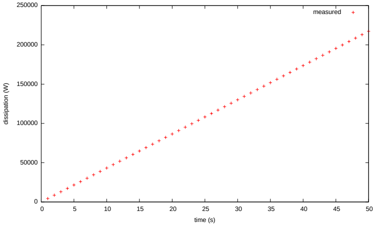
Gnuplot (3)

Let us create the following gnuplot script: *script1*.

`set term postscript eps color`	→ output is set to be a color posts
`set xlabel 'time (s)'`	→ set the label of x-axis
`set ylabel 'dissipation (W)'`	→ set the label of y-axis
`set output 'plot1.eps'`	→ set the name of graphics file
`plot 'datas.dat' title 'measured'`	→ plot the data

We then run gnuplot on this script as follows: `>gnuplot script1`

The following file *plot1.eps* is then generated:



Gnuplot (4)

Physics of the Earth and Planetary Interiors 188 (2011) 47–68

Contents lists available at ScienceDirect

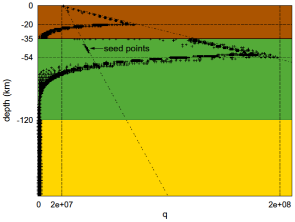
Physics of the Earth and Planetary Interiors

journal homepage: www.elsevier.com/locate/pepi

ELSEVIER

FANTOM: Two- and three-dimensional numerical modelling of creeping flows for the solution of geological problems

Cedric Thieulot *



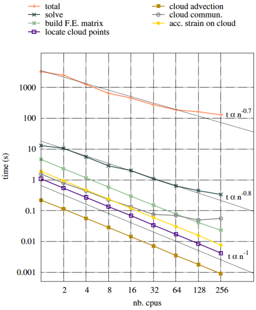


Fig. 17. Average timings for various routines calls.