

IISEE lecture for group training

Fortran programming for beginner seismologists

Lesson 3

Lecturer

Tatsuhiko Hara

Reference

Introduction to FORTRAN90/95 by S. J. Chapman (New York: McGraw-Hill, 1998)

Let's make a travel time table

- In Lesson 2, we have developed a program to calculate a travel time for a given pair of a focal depth and an epicentral distance.
- It is necessary to calculate travel times for a given focal depth for a set of epicentral distances to obtain a travel time table.
- In Lesson 3, we are going to extend our program to make a travel time table.

How can we avoid tiring task?

- It is a quite tiring task to input a certain value among a set of epicentral distances to the program developed in Lesson 2, and run the program many times in order to get a travel time table.
- To avoid this tiring task, we use an iterative *DO loop* (a counting loop).

The Iterative DO LOOP

- The iterative *DO* loop construct has the form

```
do index=(initial value), (final value), increment
    (calculation)
end do
```

where the value of the variable "index" (the loop counter/index) changes from initial value to final value by a step of increment.

- When you do not specify increment, "1" is used.
- It is desirable to use INTEGER type variable for index, although REAL type variable is allowed in some compilers.

Example (1)

The followings are examples of *DO ... END DO* loop:

Program 1

```
program ex3_1
implicit none
integer :: I
  do i=1, 10
    write(*,*) i
  end do
stop
end program ex3_1
```

Program2

```
program ex3_1a
implicit none
integer :: i
  do i=1, 10
    write(*,*) i
  end do
stop
end program ex3_1a
```

Note that “write(*,*) i” is indented to make it easy to understand the structure in Program 1. We highly recommend this kind of indentation, although the performance of Program 1 is the same of that of Program 2.

Example (2)

Example of *increment*

```
program ex3_2
implicit none
integer :: i
    do i=1, 10, 2
        write(*,*) i
    end do
stop
end program ex3_2
```

Another example

```
program ex3_2a
implicit none
integer :: i
    do i=1, 10, 3
        write(*,*) i
    end do
stop
end program ex3_2a
```

Example (3)

You can use variables for initial, final values, and increment:

```
program ex3_3
implicit none
integer :: i, istart=1, iend=10, inc=3
  do i=istart, iend, inc
    write(*,*) i
  end do
stop
end program ex3_3
```

Example (4)

The following program tries to calculate square roots:

```
program ex3_4
implicit none
integer :: i
  do i=1, 10
    write(*,*) i, sqrt(i)
  end do
stop
end program ex3_4
```

But it does not work.

This does work:

```
program ex3_4
implicit none
integer :: i
real :: x
  do i=1, 10
    x = i
    write(*,*) i, sqrt(x)
  end do
stop
end program ex3_4
```

The argument for `sqrt` should be real type.

Exercise 3-1

- (a) Compile and run the programs given in the examples (1)-(4).
- (b) Modify the program for calculating a travel time to calculate travel times for a set of numbers of epicentral distances following the suggestions below:
 - Do not change the part for z .
 - Use an iterative DO loop to change the value of `delta` from 0 to 100 with increment of 5.
 - Use the following statement in the body of the iterative DO loop to output the results:

```
write(*,*) delta, tp
```

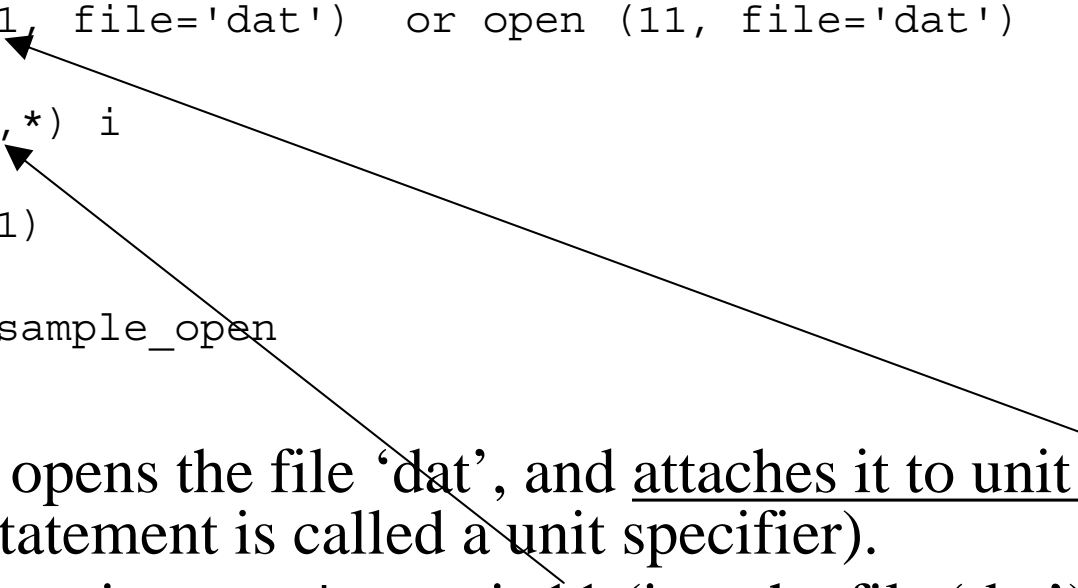
Saving results

- Now we have made a program to print out a travel time table. As a next step, let's try to plot this table.
- To do this job, first we save the travel time table to a certain file.

OPEN statement

The following is an example of OPEN and CLOSE statements:

```
Program sample_open
implicit none
integer :: i
open (unit=11, file='dat') or open (11, file='dat')
  do i=1, 10
    write(11,*) i
  end do
close(unit=11)
stop
end program sample_open
```



where

- OPEN statement opens the file 'dat', and attaches it to unit 11 ("11" in OPEN statement is called a unit specifier).
- WRITE statement prints out *i* to unit 11 (i.e., the file 'dat')
- CLOSE statement closes the file.

Exercise 3-2

- Modify the program to calculate travel times to save the results to a file “tt.dat”

Hints:

- use OPEN and CLOSE statements
- use the following statement

```
write(20,*) delta, tp, ts, ts-tp
```

Let's plot a travel time table

- Now we have obtained the file which contains the travel time table.
- Let's plot this table using *gnuplot*.

GNU PLOT

- Gnuplot is a portable command-line driven interactive data and function plotting utility for UNIX, Linux, MS Windows family, etc (<http://www.gnuplot.info/>).
- Gnuplot is included in software packages of Cygwin.
- You can start this software by the following command:

```
$ gnuplot
```

Plotting travel times

Exercise 3-3

First, we use Gnuplot interactively. Try the following commands:

```
Gnuplot> plot 'tt.dat'
```

```
Gnuplot> plot 'tt.dat' using 1:3 with lines
```

```
Gnuplot> plot 'tt.dat' using 1:4
```

```
Gnuplot> plot 'tt.dat', 'tt.dat' using 1:3
```

Gnuplot commands

- **Title**

```
gnuplot> set title "Travel time"
```

- **Axis**

```
gnuplot> set xlabel "Epicentral distance (km)"
```

```
gnuplot> set ylabel "Time (sec)"
```

- **Legend**

```
gnuplot> plot 'tt.dat' title "P-wave"
```

```
gnuplot> plot 'tt.dat' title "P-wave", ¥
```

```
> 'tt.dat' using 1:3 title "S-wave"
```

Automatically displayed, not necessary to type



Making a command file

- It is a tiring job to type the commands in the previous slides each time.
- Let's create a file "plotcom" which contains the following commands:

```
set title "Travel time"  
set xlabel "Epicentral distane (km)"  
set ylabel "Time (sec)"  
plot 'tt.dat' title "P wave", ¥  
'tt.dat' using 1:3 title "S wave"
```

How to use a command file? (1)

- You can “load” a file “plotcom” in the interactive mode.

Exercise

Try the following command after starting gnuplot:

```
Gnuplot> load 'plotcom'
```

How to use a command file? (2)

- You can use a command file as an argument of gnuplot.

Exercise

Add the following two lines at the beginning of “plotcom.”

```
set terminal postscript  
set output "tt.ps"
```

Then, try the following:

```
$ gnuplot plotcom  
$ ls
```

You will find that the postscript file “tt.ps” is created.