Multi-run script and automation practise

Abstract

The focus of this assignment is to develop the ability to run a FORTRAN program from a PYTHON script, including GNUPLOT functionalities, to automize the plotting and fitting data process. The exercise is performed again over the matrix-matrix multiplication program of the exercise three of the first assignment.

Code development

In this code the MODULE, named DEBUGMOD, developed in the previous assignment to debug a generic code, is used. Another MODULE similar to the one developed in the exercise three of the first assignment is included. The latter is named matrix and contains, above all, three functions that perform the matrix-matrix multiplication in three different ways. These functions are named matrix_multiplication1, matrix_multiplication2 and matrix_multiplication3 and take in input two real double precision matrices and perform the multiplication. To see the details on the construction of this MODULE, one can check the attached code, here I prefer to highlight the requests of this assignment.

In particular this MODULE contains a SUBROUTINE, shown in figure 1, that opens a file and read its content, the dimensions of the matrices. This SUBROUTINE takes in input the name, a CHARACTER, of the aforementioned file and an INTEGER array, in which the dimensions are stored. A LOGICAL flag, named file_exists is used to check if the input file is present in the current folder. If not a warning message is printed. The file in which the dimensions are stored is named dims.txt.

```
SUBROUTINE DIM_FILE(file_name,sizes)
  ! This subroutine opens the file from which the dimensions
  ! of the matrices to be multiplied are read.
  !The dimensions are then stored in an array.

  CHARACTER(*) :: file_name     ! the name of the aforementioned
  file
  INTEGER*4, DIMENSION(4) :: sizes ! integer array in which the
  dimensions arre stored
```

```fortran
   LOGICAL :: file_exists

   inquire (file = file_name, exist=file_exists)  ! test if the
   input file exists
   IF (.not. file_exists) THEN
      PRINT*, 'The file does not exist'
      STOP
   ELSE IF  (file_exists) THEN
      OPEN (12, file = file_name, status = 'old') ! status='old'
   means 'the file should exist'
      DO ii = 1,4
         READ(12,*) sizes(ii)
      ENDDO
      CLOSE(12)
   ENDIF
 END SUBROUTINE DIM_FILE
```

Listing 1: The `SUBROUTINE` to read the input from file

Another `SUBROUTINE`, named `DIMS`, shown in figure 2 asks the user to insert the dimensions of the matrices, cheking if they are positive. If the user inserts negative dimensions, the program asks for them again, through a `DO WHILE` loop.

```fortran
 SUBROUTINE DIMS(r1,c1,r2,c2)
 ! This subroutine asks the user the dimensions of the matrices to
    be
 ! multiplied. It takes in input the integer values that are these
    inserted dimensions (rows&columns).
   INTEGER*4 :: r1,c1,r2,c2
   r1=0
   c1=0      ! all the dimensions are 'inizialized' to zero
   r2=0
   c2=0

   DO WHILE (r1 <= 0)
      PRINT*, "The number of rows of the first matrix is:"
      READ*, r1
      IF (r1 <= 0) PRINT*, "Try with a positive value"
   ENDDO
   DO WHILE (c1 <= 0)
      PRINT*, "The number of columns of the first matrix is:"
      READ*, c1
      IF (c1 <= 0) PRINT*, "Try with a positive value"
   ENDDO
   DO WHILE (r2 <= 0)
      PRINT*, "The number of rows of the second matrix is:"
      READ*, r2
      IF (r2 <= 0) PRINT*, "Try with a positive value"
   ENDDO
   DO WHILE (c2 <= 0)
      PRINT*, "The number of columns of the second matrix is:"
      READ*, c2
      IF (c2 <= 0) PRINT*, "Try with a positive value"
   ENDDO
 END SUBROUTINE DIMS
```

Listing 2: 'DIMS' SUBROUTINE

Moreover a function, called `test_DIMS`, tests if the dimensions of the matrices allow the multiplication.

In the main program there is a logical flag, `manual_insert`, to decide if the input should be read from a file or if it must be inserted manually by the user. The corresponding piece of code is shown in figure 3.

```fortran
PROGRAM EX4
  USE matrix
  USE DEBUGMOD
  IMPLICIT NONE


  INTEGER*4, DIMENSION(4) :: allsize  ! Variable to store the sizes
     of matrices

  LOGICAL :: manual_insert  ! Flag to decide input mode

  turn_debug_on = .TRUE.
  manual_insert = .FALSE.


  open(1,file="time1.txt",status='old',position="append",action='
    write')
  open(2,file="time2.txt",status='old',position="append",action='
    write')
  open(3,file="time3.txt",status='old',position="append",action='
    write')
  open(4,file="time4.txt",status='old',position="append",action='
    write')

  IF (manual_insert) THEN
     CALL DIMS(r1,c1,r2,c2)
  ELSE
     CALL DIM_FILE("dims.txt", allsize)
     r1 = allsize(1)
     c1 = allsize(2)
     r2 = allsize(3)
     c2 = allsize(4)


  END IF
```

Listing 3: main program

Like in the previous assignment the matrices to be multiplied are filled with random numbers, drawn from the uniform distribution in the interval [0,1]. Using the Fortran function `CPUTIME`, the time taken to do the matrix multiplication is printed in `.txt` files. The operation is done in four different ways: through the functions `matrix_multiplication*` (where $*$ stands for 1,2,3) and the Fortran intrinsic function `MATMUL`. The program stores the time taken and the size of the resulting matrix in four different files, one for each multiplication method, named `time*.txt` (where $*$ stands for 1,2,3,4). A `GNUPLOT` script, called `plots.gnu` plots the logarithm of the time taken for the operation vs the size (i.e. number of entries) of the resulting matrix. The results are shown in the next section.

Moreover a python script, `running.py`, asks the user to insert the minimum and the maximum dimension ,N_{min} and N_{max} for the matrix and takes 10 values in this range. The values are the dimensions of the matrices to be multiplied and are collected in the `dims.txt` file from which the `FORTRAN` program reads the input. The python script launches the aforementioned `FORTRAN` code and the `GNUPLOT` script.

Another python script, `fit.py` fixes N_{min} and N_{max} to 1 and 10^3. The request of the assignment was to consider the biggest possible range. Empirically this is more or less the range after which my program crashes or takes a long time to run. However the number 2'147'483'647 is the maximum positive value for a 32 bit signed integer, so I tried with its square root as N_{max} (a root rounding, since the number is not a perfect square), but the program takes much time. This python script launches the `FORTRAN` code and `GNUPLOT` script, `fit.gnu`, that produces a similar plot than the other, but this time fitting the data. The fitting function is

$$f(x) = a + b \cdot x^c \tag{1}$$

The file `fit.log` contains the values of the fitting parameters. I highlight that I had to inizialize the fitting parameters values to $a = 0.1$, $b = 10^{-10}$ and $c = 3$ to obtain resonable fits.

Results

The plot performed through the script `plots.gnu` is showed in figure 1. For each multiplication method, we have ten points. In the case shown in figure N_{min} is 5 and N_{max} is 300. Generally, we can say that as the size increases the time taken increases. The better performance is obtained by the `MATMUL` intrinsic function, while the worst through the direct calculation 'rows times columns'.
Then, as explained in the previous section, the plot of the time taken vs the size of the resulting matrix is fitted, and it is shown in figure 2. In the range between $N_{min} = 1$ and $N_{max} = 10^3$ this time we have 100 points. Despite the fact that the `MATMUL` method, respect to the others, is very efficient in time increasing the resulting matrix size, checking the `fit.log` file one can notice an interesting pattern: the scaling parameter indicated as c in equation 1 is around 1.5 for all the methods.

Self-Evaluation

This exercise was useful to me to refresh some gnuplot's tools about fitting and to learn python's library `subprocess` to run codes from python. Also, I got aware that develop the practise of automized fit and run differents codes from only one script is time saving.

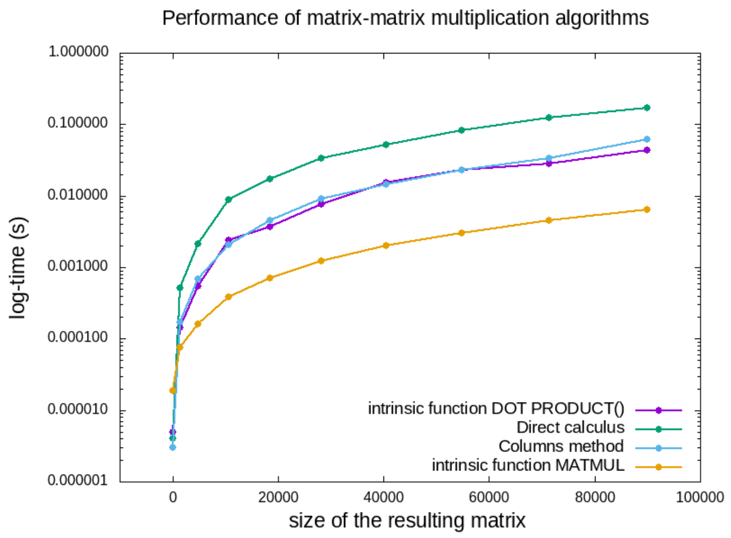

Figure 1: Plot of the logarithm of the time taken in the matrix-matrix multiplication vs the size of the resulting matrix, for four different multiplication methods.

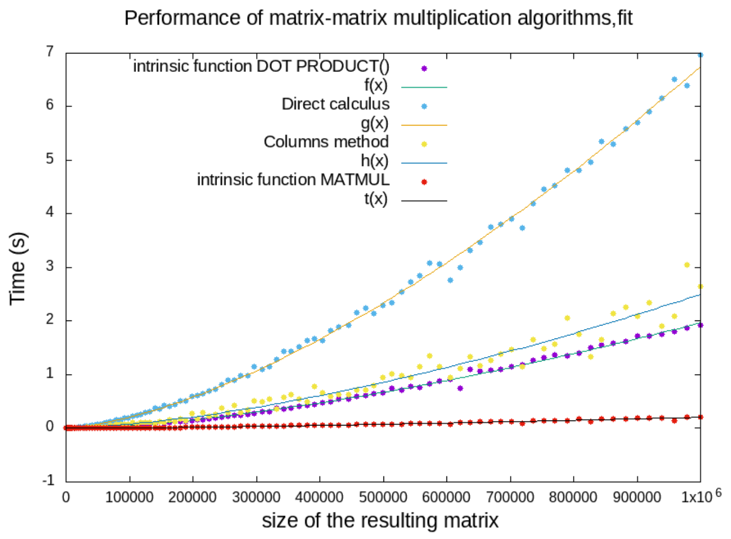

Figure 2: Fitted plot of the time taken in the matrix-matrix multiplication vs the size of the resulting matrix, for four different multiplication methods.