Checkpoint and matrix-matrix multiplication

1 Abstract

The aim of this exercise is to implement checkpoint function in order to visualize possible errors during program execution.

So i define a module named ***checkpoint debug*** that the main program recalls every time it finish to evaluate a function result.

In order to understand better how every function works,at the beginning,i write its documentation and in order to check if they work properly,in module , named ***matrix mul***, some if statements are fixed.

2 Theory

The ***debugging procedure*** is fundamental to discover if our code that seems works properly at first sight,indeed,it doesn't work as we think.

A programmer could discover bugs or errors both during programming procedure (initialization error,type variable error,error calling function and so on) and ,as happens frequently, when tests a program in such different situations.

In this exercise,for example, i applied some different checks in order to be sure if a matrix-matrix multiplication works properly.
Matrix-matrix multiplication is defined as :

"Be K a ring . Be A and B matrices with m and n rows and n and l columns respectively . Be a_{ij} and b_{ij} A's elements and B's elements respectively that laid in K.
It will be defined C=AB with m rows and l columns as matrix product if named c_{ij} C's elements it will have to :

$$c_{ij} = \sum_{r=1}^{n} a_{ir} b_{rj} \tag{1}$$

for each row i and column j."

3 Code development

My code is divided in three program units : two modules and one main program.

- MODULE ***checkpoint debug*** : here i define three functions as shown in interface part and for each one it checks that two integer/real/complex double matrices ,passed as argues, have same dimensions and the same elements.Otherwise if same conditions is not satisfied ,using STOP function, the program will be arrested.

- MODULE ***matrix mul*** : here i define two functions named ***by rows*** and ***by columns***.
  The first one computes matrix-matrix multiplication moving on same row changing column at every inner iteration .
  The second one do the same but this time fixing column and the index moves along rows.
  Before computation begins the code checks if matrices have right dimensions and,in particular, they are not null!

- MAIN PROGRAM ***test modules***: In order to evaluate time consuming related to functions defined in the previous point, i used ***cpu time*** before and after calling that functions .
  Then, using MODULE ***checkpoint debug*** , the code researches differences and errors in matrix computed by ***by rows***, ***by columns*** and ***matmul*** function .

```
if (count==0) then              !debug variable

debug=.true.

WRITE(*,*) "Matrices are almost the same?" ,debug

else

debug =.false.

 WRITE(*,*) "Matrices are almost the same?" ,debug

 STOP

 end if
```

Checkpoint code example

4 Results

When i try to bad initialize matrices or to introduce bugs during computations, as shown below,the checks react displaying error strings.

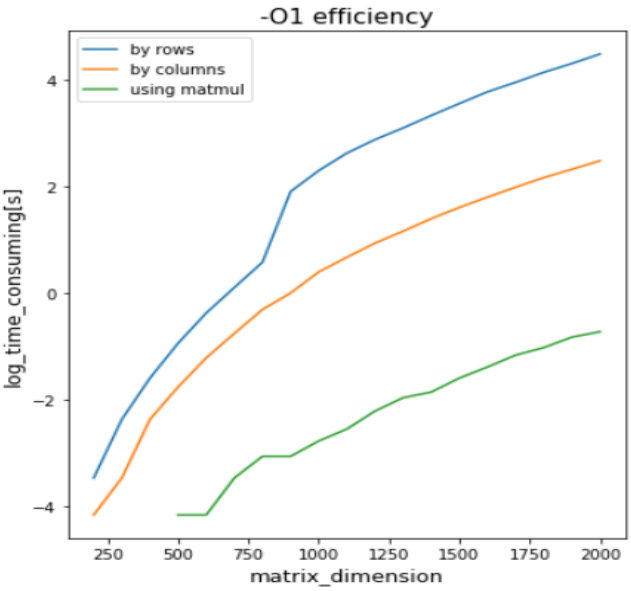
```
The matrices will have dimensions          0        110  and        110        120
Some dimension is 0!
```

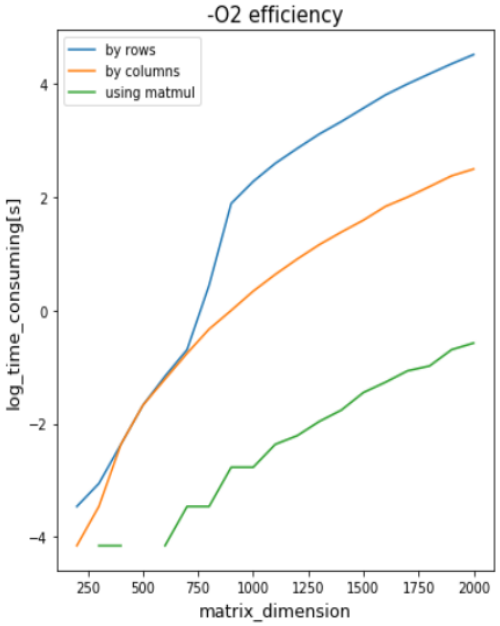
```
Matrices elements are almost the same? F
```

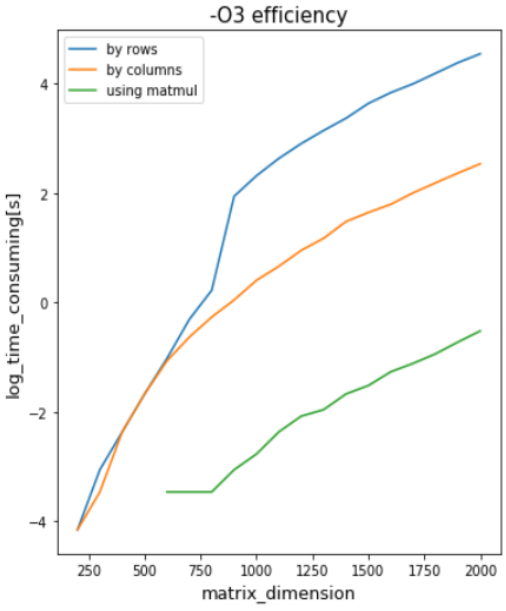
```
DEBUGGING TEST :
For by_rows and by_columns :
Matrices have different dimensions!        100        120 and        50        240
```

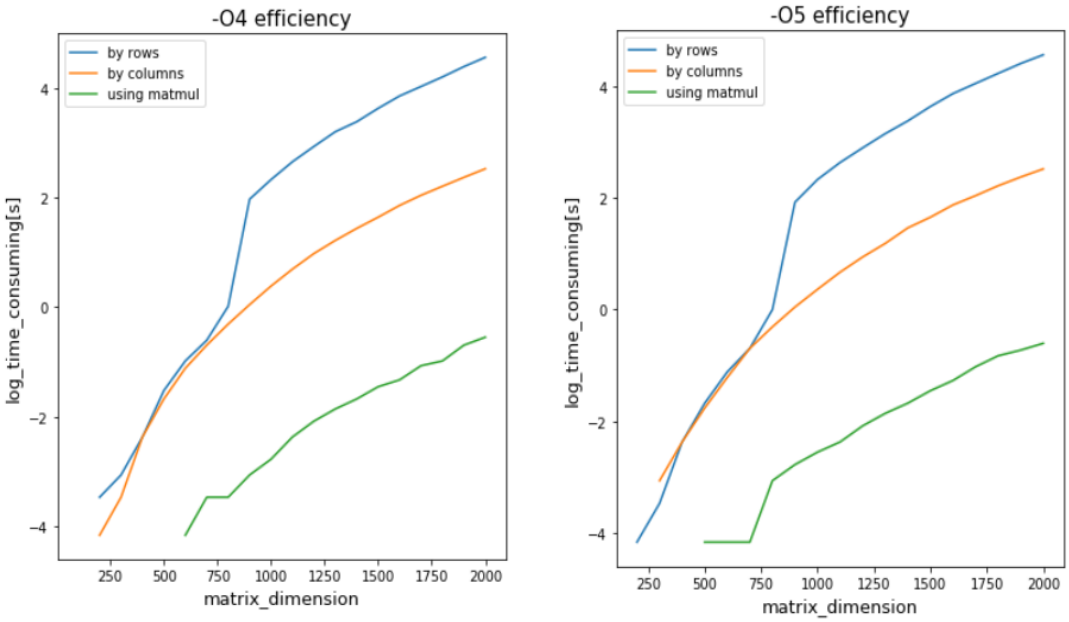
```
The matrices will have dimensions          100        110  and        220        120
Impossible to do matrices multiplication :
dim2's matrix1 its different than dim1's matrix2
```

Farther,plotting log scale time consuming on y axis and matrix dimension on x axis,i find clear more efficiency using ***matmul*** function and ***by columns*** than ***by rows***.
This because matrix's elements in the same column are stored near.


-O1 efficiency
by rows
by columns
using matmul
log_time_consuming[s]
matrix_dimension
4
2
0
−2
−4
250
500
750
1000
1250
1500
1750
2000


-O2 efficiency
by rows
by columns
using matmul
log_time_consuming[s]
matrix_dimension
4
2
0
−2
−4
250
500
750
1000
1250
1500
1750
2000


-O3 efficiency
by rows
by columns
using matmul
log_time_consuming[s]
matrix_dimension
4
2
0
−2
−4
250
500
750
1000
1250
1500
1750
2000



5 Self-evaluation

- Improving checks is very useful :

  for example i can define two matrices A and B such that A's 2° dimension is different then B's 1° dimension and probably,using simply do loops, it will be never shown as an error during compilation and execution time.
  So,it's a good routine not limiting ourself at only syntactic errors.

- Another note is for different flag options used to optimize the code :

  it seems that $-O < n >$ doesn't change drastically cpu time consuming.