Checkpoints and Documentation

Umberto Maria Tomasini
Quantum Information Course

23/10/2018

Abstract

This report contains two exercises. The first is about an example of a subroutine which serves as a checkpoint, in order to debug the code. In the second exercise we will reconsider the code of the exercise 3 of the first week -which was about matrix- matrix multiplication via three different methods, analyzing their execution time- and we will make it user-friendly with comments, documentation and checkpoints. Moreover we will make it easier to debug adding pre-conditions and post-conditions., error handling and again checkpoints.

Chapter 1

Exercise 1: Checkpoint

Theory

Code Development

The module *checkpoint_debug* contains different subroutines, interfaced via the interface *checkpoint*. All the subroutines contain an if statement, which checks if the logical variable DEBUG is true. If it is, the checks below are done. All the subroutine, in addition to printing "CHECKPOINT-DEBUGGING", print an input string. Moreover, they all print at the end of the checks the cpu_time, in order to have an additional check point. The subroutines each contain a specific additional feature:

- check_none: none
- check_int2: given two integer*2 variables, var and varcheck, checks if they are equal. It should be so, because one variable is the actual one, whilst the other one contain the right value.
- check_real4: given two real*4 variables, var and varcheck, checks if they are equal. It should be so, because one variable is the actual one, whilst the other one contains the right value. The equality test is done taking the absolute value of the difference between the two numbers, divided for the check value. This value is confronted with a bound, which is 10^{-8} This is done in order to not be dependent on the order of magnitude of the variables.
- check_dcomplex: given two double complex variables, var and varcheck, checks if they are equal. It should be so, because one variable is the actual one, whilst the other one contains the right value. The equality test is done taking the absolute value of the difference between the two numbers, divided for the absolute value of the check value. This value is confronted with a bound, which is 10^{-5} This is done in order to not be dependent on the order of magnitude of the variables. Even if the precision of real*4 is 10^{-8}, we decide to widen that bound, taking into accounr possible error propagations.
- check_r4array: given two 2-dimensional array variables, m and mcheck, with their dimensions, checks if they are equal. It should be so, because one variable is the actual

one, whilst the other one contains the right values. The equality test is done entry by entry, taking the absolute value of the difference between the two numbers, divided for the check value. This value is confronted with a bound, which is 10^{-10} This is done in order to not be dependent on the order of magnitude of the variables. For each entry which is not equal, the variable INTEGER*4 accum increseas by 1 (starting from 0). If accum in the end is bigger than 0, a WARNING message states how much entries between the two matrices are different. Otherwise, it is printed "Same matrices".

In any subroutine, except for check_r4array, the actual value and the value that should be, are printed if they are not equal. Otherwise, only the actual value is printed.

```
        module checkpoint_debug
implicit none

interface checkpoint!INTERFACE
module procedure check_none, check_int2,
$                                check_real4,
   check_dcomplex,
$                                check_r4array
end interface

contains

subroutine check_none(debug,stringg)
!When no variables are passed
implicit none
logical debug
character(:), allocatable :: stringg
real ttime

if(debug.eqv..true.) then
call cpu_time(ttime)
print*, " "
print*, "CHECKPOINT-DEBUGGING"
print*, stringg
print*, " "
print*, "Code time: ", ttime
print*, " "
end if

end subroutine check_none

subroutine check_int2(debug,stringg,var,varcheck)
!check on an integer*2 variable
implicit none
logical debug
character(:), allocatable :: stringg
```

```
real ttime
integer*2 var, varcheck

if(debug.eqv..true.) then
call cpu_time(ttime)
print*, "␣"
print*, "CHECKPOINT-DEBUGGING"
print*, stringg
print*, "␣"
if(var==varcheck) then !checking correctness
print*, "The␣value␣of␣the␣variable␣is:", var
else
print*, "The␣actual␣value␣is:", var
print*, "It␣should␣be", varcheck
end if
print*, "␣"
print*, "Code␣time:␣", ttime!code time printed
print*, "␣"
end if

end subroutine check_int2

subroutine check_real4(debug,stringg,var,varcheck)
!check on an real*4 variable
implicit none
logical debug
character(:), allocatable :: stringg
real ttime
real*4 var, varcheck

if(debug.eqv..true.) then
call cpu_time(ttime)
print*, "␣"
print*, "CHECKPOINT-DEBUGGING"
print*, stringg
print*, "␣"
!checking correctness
if(abs(var-varcheck)/varcheck < 10E-5 ) then
print*, "The␣value␣of␣the␣variable␣is:", var
else
print*, "The␣actual␣value␣is:", var
print*, "It␣should␣be", varcheck
print*, "␣"
end if
print*, "Code␣time:␣", ttime!printing code time
```

```
print*, "␣"
end if

end subroutine check_real4

subroutine check_dcomplex(debug,stringg,var,varcheck)
!check on an double complex variable
implicit none
logical debug
character(:), allocatable :: stringg
real ttime
double complex var, varcheck

if(debug.eqv..true.) then
call cpu_time(ttime)
print*, "␣"
print*, "CHECKPOINT-DEBUGGING"
print*, stringg
print*, "␣"
!checking correctness
if(abs(var-varcheck)/abs(varcheck) < 10E-5 ) then
print*, "The␣value␣of␣the␣variable␣is:", var
else
print*, "The␣actual␣value␣is:", var
print*, "It␣should␣be", varcheck
print*, "␣"
print*, "Code␣time:␣", ttime!printing code time
print*, "␣"
end if
end if

end subroutine check_dcomplex

subroutine check_r4array(debug,stringg, m,mcheck,nn,
   mm)
implicit none
!checking if the input arrays are equal
real*4, dimension(nn,mm) :: m, mcheck
integer*2 :: tt,ss,nn,mm
integer*4 accum
logical debug
character(:), allocatable :: stringg
real ttime

if(debug.eqv..true.) then
```

```
call cpu_time(ttime)
print*, "␣"
print*, "CHECKPOINT-DEBUGGING"
print*, stringg
print*, "␣"
accum=0
do tt=1,nn
do ss=1,mm
if(abs(m(tt,ss)-mcheck(tt,ss))/mcheck(tt,ss) > 10E-5)
    then
accum=accum+1
end if
end do
end do
!how many different entries?
if(accum>0) then
print*, "The␣two␣arrays␣have"
$                , accum, "different␣entries"
print*, "␣"
else
print*, "Same␣arrays"
print*, "␣"
end if
end if

print*, "Code␣time:␣", ttime
print*, "␣"

end subroutine check_r4array

end module
```

Results

We check on a test program if the module above works. In the program we define some variables: one integer*2, one real*4, one double complex and one real*4 array 2-dimensional. We also define the sambe number of check variables, which should have in theory the same value. We insert two bugs: the value of the real variable is different from the check real variable and we change one entry of the array. We can infer from what is printed on terminal that the check subroutine worked.

```
CHECKPOINT-DEBUGGING
Checking none

Code time:    1.15000003E-03
```

```
CHECKPOINT-DEBUGGING
Checking int2

The value of the variable is:      8

Code time:    1.18999998E-03


CHECKPOINT-DEBUGGING
Checking real4

The actual value is:   5.55550003
It should be   5.55600023

Code time:    1.20299996E-03


CHECKPOINT-DEBUGGING
Checking double complex

The value of the variable is:  (3.0000000000000000,6.0000000000000000)

CHECKPOINT-DEBUGGING
Checking array 2-dim, real4

The two arrays have           1 different entries

Code time:    1.22600002E-03
```

The actual test program:

```
program testing
use checkpoint_debug

implicit none
logical debug
character(:), allocatable :: stringg
integer*2 intvar, intvarcheck
real*4 realvar, realvarcheck
double complex dcvar, dcvarcheck
integer*2 :: ii, jj, nn, mm
real*4, dimension(:,:), allocatable :: arr,arrcheck
```

```
debug=.true.
nn=4
mm=5
allocate(arr(nn,mm))
allocate(arrcheck(nn,mm))
do ii=1,nn
do jj=1,mm
arr(ii,jj)=ii
arrcheck(ii,jj)=arr(ii,jj)
end do
end do
dcvar=(3,6)
dcvarcheck=dcvar

realvar=5.5555
realvarcheck=5.556 !bug

intvar=8
intvarcheck=intvar

arr(3,3)=5

stringg="Checking␣none"
call checkpoint(debug,stringg)

stringg="Checking␣int2"
call checkpoint(debug,stringg,intvar,intvarcheck)

stringg="Checking␣real4"
call checkpoint(debug,stringg,realvar,realvarcheck)

stringg="Checking␣double␣complex"
call checkpoint(debug,stringg,dcvar,dcvarcheck)

stringg="Checking␣array␣2-dim,␣real4"
call checkpoint(debug,stringg,arr,arrcheck,nn, mm)

end program testing
```

Self-Evaluation

The main goal achieved: writing an as general as possible subroutine useful to checkpoints, according to different types of variable.

Chapter 2

Exercise 2: Documentation

Theory

Code Development

This program implements matrix-matrix multiplication in three different method: row by row, column by column and by intrinsic function. For each method an apposite subroutine is called: ROWROWMATMUL, COLCOLMATMUL and MATMULINTRINSIC. For each method the CPU_TIME is computed. This is done for squared matrix, from !100x100 to 500x500, with a step=100. The first matrix is such that at the entry (i,j) there is the value i+j, whilst the second matrix is such that at the at the entry (i,j) there is the value i*j. The multiplications are done in a do cycle. In each cycle, at the beginning, the matrices are allocated and in the ending are deallocated. The results are printed on a file, called "Results", at unit=40.

DEBUGGING of the program. This is controlled with a CHARACTER*1 variable called CHOICE. The programmer can choose to give the choice to do or not the debugging to the user keeping or changing the value of CHOICE. Keeping its default value "X", the user can choose at the beginning of the program to do the debug, via a printed choice on the terminal. If he says no (printing "n"), the logical variable DEBUG turns .FALSE. and all the debugging routines do not work.
If he says yes (printing "y"), the logical variable DEBUG turns .TRUE. and therefore all the debugging procedures below are executed. If one of the them reports an error, a warning message is printed. If he fails inserting the character, the program stops. On the other hand, if the programmer changes the value of CHOICE in "n", the user has not this choice and the debugging is not done (DEBUG is .FALSE.). Otherwise, changing the value of CHOICE in "y", the user has not the choice and the debugging is done (DEBUG is .TRUE.).
The debugging is done printing various checkpoints throughout the program, in specific points: at the beginning of each cycle, where there is the initialization of the input matrices, at the calling the subroutines of the three methods, checking the resulting matrices.
Moreover the subroutines CHECKDIM and CHECKMAT are used a lot, interfaced via the operator CHECK. CHECKDIM is used every time a subroutine is called and it checks if the dimension of the matrix is what it should be. CHECKMAT checks after the calling of the

three methods that the imput matrices are still the same (after each one). Moreover, it checks if the resulting matrices mris1, mris and mris3 are the same at the end of each cycle.
Another check is printing the istant cpu_time before and after each calling to a function which exploits one method. This is done in order to check immediately if there are some problems with time computation. Lastly, error handling commands are added for the functions OPEN, WRITE, ALLOCATE and DEALLOCATE. They print which error the functions reports and its relative error message.
If one of the these procedures reports an error, a warning message is printed. At the end of the program, the total time spent on debugging and the total time spent executing the program are printed. Furthermore, the percentage of time spent on debugging respect to the total time is printed. The information on each subroutine are contained in the below documentation of the program. The subroutines are:

- rowrowmatmul, matrix matrix multiplication row by row
- colcolmatmul, matrix matrix multiplication column by column
- matmulintrinsic, matrix matrix multiplication via intrisic function matmul
- checkdim, checks if the dimension of the matrix is what it should be
- checkmat, checks if two matrices are equal and says how many entries are different

The first three matrices are contained in the module *matmultiplication*, the last two in the module *debugging*.

The results are contained in the last chapter.

```
! \brief \b Multiplication matrix-matrix and its
  computational time
!
!  =========== DOCUMENTATION ===========
!  Definition:
! ===========
!
!      SUBROUTINE ROWROWMATMUL (nn,m1,m2,mris1,timetot1)
!
!       .. Scalar Arguments ..
!       INTEGER*2             nn
!        REAL               timetot1
!       ..
!      .. Array Arguments ..
!       INTEGER*2          m1(nn,nn)
!       INTEGER*2          m2(nn,nn)
!        INTEGER*2          mris1(nn,nn)
!      ..
!
!  Purpose
```

```
!  =======
!
!\details \b Purpose:
!\verbatim
!
! ROWROWMATMUL does a matrix-matrix multiplication between
   two matrices
! given as input: m1 and m2. The resulting matrix is mris1.
   ROWROWMATMUL
! deals with INTEGER*2 squared matrices, with numbers of rows
    and columns equal ! to nn. In the entry (i,j) of mris1
   there is the scalar product between the ! i-th row of m1
   and the j-th column of m2.
! The  multiplication is row by row because while i is fixed
   j
! goes from 1 to nn. In other words the j cycle is inside the
    i cycle.
! Calling the function CPU_TIME at the beginning of that
   moltiplication, we assign the initial time to REAL
   variable start1. The same is done for the ending ! time,
   in REAL variable finish1. The total time timetot1 is the
   difference between the finish time and the ending time.

!\endverbatim
!
!  Arguments:
!  ==========
!

! \param[in] nn
! \verbatim
!          nn is INTEGER*2
!          The dimension of the squared array m1, m2 and
   mris1
! \endverbatim
! \param[in] m1
! \verbatim
!          m1 is INTEGER*2 ARRAY (nn,nn)
! \endverbatim
!
! \param[in] m2
! \verbatim
!          m2 is INTEGER*2 ARRAY (nn,nn)
! \endverbatim
!
```

```
! \param[out] m2
! \verbatim
!          mris1 is INTEGER*2 ARRAY (nn,nn)
! \endverbatim
!
!
! \param[out] timetot1
! \verbatim
!          timetot1 is REAL
! \endverbatim
!
!  Authors:
! ========
!
! \author Univ. of Padua
!
! \date 23 October 2018
!
!
!       SUBROUTINE COLCOLMATMUL (nn,m1,m2,mris2,timetot2)
!
!       .. Scalar Arguments ..
!       INTEGER*2             nn
!        REAL               timetot2
!       ..
!      .. Array Arguments ..
!       INTEGER*2          m1(nn,nn)
!       INTEGER*2          m2(nn,nn)
!        INTEGER*2          mris2(nn,nn)
!       ..
!
!  Purpose
!  =======
!
!\details \b Purpose:
!\verbatim
!
! COLCOLMATMUL does a matrix-matrix multiplication between
   two matrices
! given as input: m1 and m2. The resulting matrix is mris2.
   COLCOLMATMUL
! deals with INTEGER*2 squared matrices, with numbers of rows
    and columns equal ! to nn. In the entry (j,i) of mris1
   there is the scalar product between the j-th row of m1 and
    the i-th column of m2.
```

```
! The  multiplication is row by row because while i is fixed
   j
! goes from 1 to nn. In other words the j cycle is inside the
    i cycle.
! Calling the function CPU_TIME at the beginning of that
   moltiplication, we assign the initial time to REAL
   variable start2. The same is done for the ending time, in
   REAL variable finish2. The total time timetot2 is the
   difference between the finish time and the ending time.

!\endverbatim
!
!  Arguments:
!  ==========
!


! \param[in] nn
! \verbatim
!          nn is INTEGER*2
!          The dimension of the squared array m1, m2 and
   mris2
! \endverbatim
! \param[in] m1
! \verbatim
!          m1 is INTEGER*2 ARRAY (nn,nn)
! \endverbatim
!
! \param[in] m2
! \verbatim
!          m2 is INTEGER*2 ARRAY (nn,nn)
! \endverbatim
!
! \param[out] m2
! \verbatim
!          mris2 is INTEGER*2 ARRAY (nn,nn)
! \endverbatim
!
!
! \param[out] timetot2
! \verbatim
!          timetot2 is REAL
! \endverbatim
!
!  Authors:
!  ========
```

```
!
! \author Univ. of Padua
!
! \date 23 October 2018
!
!
!       SUBROUTINE MATMULINTRINSIC (nn,m1,m2,mris1,timetot1)
!
!       .. Scalar Arguments ..
!       INTEGER*2            nn
!        REAL              timetot3
!       ..
!       .. Array Arguments ..
!       INTEGER*2          m1(nn,nn)
!       INTEGER*2          m2(nn,nn)
!        INTEGER*2          mris3(nn,nn)
!       ..
!
!  Purpose
!  =======
!
!\details \b Purpose:
!\verbatim
!
! MATMULINTRINSIC does a matrix-matrix multiplication between
   two matrices
! given as input: m1 and m2. The resulting matrix is mris1.
  MATMULINTRINSIC
! deals with INTEGER*2 squared matrices, with numbers of rows
   and columns equal ! to nn.
! The  multiplication is done via the intrinsic function
  mris3=matmul(m1,m2).
! Calling the function CPU_TIME at the beginning of that
  moltiplication, we  assign the initial time to REAL
  variable start3. The same is done for the ending ! time,
  in REAL variable finish3. The total time timetot3 is the
  difference between the finish time and the ending time.

!\endverbatim
!
! Arguments:
!  ==========
!

! \param[in] nn
```

```
! \verbatim
!          nn*2 is INTEGER
!          The dimension of the squared array m1, m2 and
   mris3
! \endverbatim
! \param[in] m1
! \verbatim
!          m1 is INTEGER*2 ARRAY (nn,nn)
! \endverbatim
!
! \param[in] m2
! \verbatim
!          m2 is INTEGER*2 ARRAY (nn,nn)
! \endverbatim
!
! \param[out] m2
! \verbatim
!          mris3 is INTEGER*2 ARRAY (nn,nn)
! \endverbatim
!
!
! \param[out] timetot1
! \verbatim
!          timetot3 is REAL
! \endverbatim
!
! Authors:
!  ========
!
! \author Univ. of Padua
!
! \date 23 October 2018
!
!       SUBROUTINE CHECKDIM(nn,q,,debug)
!
!       .. Scalar Arguments ..
!       INTEGER*2            nn
!        INTEGER*2            qq
!        LOGICAL              debug
!       ..
!       .. Array Arguments ..
!       INTEGER*2         m(nn,nn)
!       INTEGER*2         mcheck(nn,nn)
!       ..
!  Purpose
```

```
!  =======
!
!\details \b Purpose:
!\verbatim
!
! CHECKDIM checks if the dimension of the matrix are correct.
    Firstly checks
! if the the dimension INTEGER*2 is above 10000. If it is so,
    it prints a WARNING
! message because it takes too much time. Secondly, it checks
    if nn is inferior
! to 1, printing a WARNING message. Lastly, it checks if nn
   is actually qq*100
! as it should be. If it is not true, it prints a WARNING
   message, stating which
! should be the dimension and the actual one. If everything
   goes well, it prints "okay: right dimensions matrices".

!\endverbatim
!
! Arguments:
!  ==========
!

! \param[in] nn
! \verbatim
!          nn is INTEGER*2
!          The dimension of the squared arrays
! \endverbatim
! \param[in] qq
! \verbatim
!          qq is INTEGER*2
!           the number of the cycle, it should be nn=qq*100
! \endverbatim
!
! \param[inout] debug
! \verbatim
!          debug is LOGICAL
! \endverbatim
!
! Authors:
!  ========
!
! \author Univ. of Padua
!
```

```
! \date 23 October 2018

!       SUBROUTINE CHECKMAT(nn,m,mcheck,debug)
!
!       .. Scalar Arguments ..
!       INTEGER*2            nn
!        LOGICAL              debug
!       ..
!       .. Array Arguments ..
!       INTEGER*2          m(nn,nn)
!       INTEGER*2          mcheck(nn,nn)
!       ..
!  Purpose
!
!  Purpose
!  =======
!
!\details \b Purpose:
!\verbatim
!
! CHECKMAT checks if the two imput matrices m and mcheck are
   equal. This is done
! via a do cycle which checks the equality between the two
   matrices element
! by element. For each entry which is not equal, the variable
    INTEGER*4 accum
! increseas by 1 (starting from 0). If accum in the end is
   bigger than 0,
! a WARNING message states how much entries between the two
   matrices are
! different. Otherwise, it is printed "Same matrices".
!\endverbatim
!
! Arguments:
!  ==========
!


! \param[in] nn
! \verbatim
!         nn is INTEGER*2
!         The dimension of the squared arrays
! \endverbatim
!
! \param[in]
! \verbatim
```

```
!          m is INTEGER*2 ARRAY (nn,nn)
! \endverbatim
!
! \param[in]
! \verbatim
!          mcheck is INTEGER*2 ARRAY (nn,nn)
! \endverbatim
!
! \param[inout] debug
! \verbatim
!          debug is LOGICAL
! \endverbatim
!
! Authors:
!  ========
!
! \author Univ. of Padua
!
! \date 23 October 2018

!
!       PROGRAM test_performance_mulmat

!  Purpose
!  =======
!
!\details \b Purpose:
!\verbatim
!
!This program implements matrix-matrix multiplication in
   three different method: !row by row, column by column and
   by intrinsic function. For each method an !apposite
   subroutine is called: ROWROWMATMUL, COLCOLMATMUL and
   MATMULINTRINSIC. !For each method the CPU_TIME is computed
   . This is done for squared matrix, from !100x100 to 500
   x500, with a step=100. The first matrix is such that at
   the entry (i,j) there is the value i+j, whilst the second
   matrix is such that at the at the entry (i,j) there is the
    value i*j. The multiplications are done in a do cycle. In
    each cycle, at the beginning, the matrices are allocated
   and in the ending are deallocated. The results are printed
    on a file, called "Results", at unit=40.
!
! DEBUGGING of the program. This is controlled with a
   CHARACTER*1 variable called CHOICE. The programmer can
```

```
   choose to give the choice to do or not the debugging to
   the user keeping or changing the value of CHOICE. Keeping
   its default value "X", the user can choose at the
   beginning of the program to do the debug, via a printed
   choice on the terminal.
! If he says no, the logical variable DEBUG turns .FALSE. and
    all the debugging routines do not work.
! If he says yes, the logical variable DEBUG turns .FALSE.
   and therefore all the debugging procedures below are
   executed. If one of the them reports an error, a warning
   message is printed.
! If he fails inserting the character, the program stops.
! Otherwise, if the programmer changes the value of CHOICE in
    "n", the user has not this choice and the debugging is
   not done (DEBUG is .FALSE.). Lastly, changing the value of
    CHOICE in "y", the user has not the choice and the
   debugging is done (DEBUG is .TRUE.).
! The debugging is done printing various checkpoints
   throughout the program, in specific points: at the
   beginning of each cycle, initialization of the input
   matrices, calling the subroutines of the three methods,
   checking the resulting matrices.
! Moreover the subroutines CHECKDIM and CHECKMAT are used a
   lot , interfaced via the operator CHECK. CHECKDIM is used
   every time a subroutine is called. CHECKMAT checks after
   the calling of the three method that the imput matrices
   are still the same (after each one). Moreover, it checks
   if the resulting matrices mris1, mris and mris3 are the
   same at the end of each cycle.
! Another check is printing the istant cpu_time before and
   after each calling to a function which exploits one method
   . This is done in order to check immediately if there are
   some problems with time computation.
! Lastly, error handling commands are added for the functions
    OPEN, WRITE, ALLOCATE and DEALLOCATE. They print which
   error the functions reports and its relative error message
   .
! If one of the these procedures reports an error, a warning
   message is printed.
! At the end of the program, the total time spent on
   debugging and the total time spent are printed.
   Furthermore, the percentage of time spent on debugging
   respect to the total time is printed.

!\endverbatim
```

```
!  Authors:
!  ========
!
! \author Univ. of Padua
!
! \date 23 October 2018

!
   ======================================================================

!MODULE MAT-MULTPLICATIONS
module matmultiplications

contains
subroutine rowrowmatmul(nn,m1,m2,mris1,timetot1)
!first method:row by row
implicit none
integer*2 ii, jj, kk, nn
integer*2, dimension(nn,nn) :: m1
integer*2, dimension(nn,nn) :: m2
integer*2, dimension(nn,nn):: mris1
real :: finish1, start1,timetot1
call cpu_time (start1)!initial time
do ii=1,nn
do jj=1,nn
do kk=1,nn
mris1(ii,jj)=mris1(ii,jj)+m1(ii,kk)*m2(kk,jj)
!SCALAR PRODUCT BTW
!ii-th ROW OF m1 AND j-th COLUMN OF m2
!IT GOES IN THE ENTRY (i,j) OF mris1
end do
end do
end do
call cpu_time (finish1)!finish time
timetot1=finish1-start1
end subroutine rowrowmatmul



subroutine colcolmatmul(nn,m1,m2,mris2,timetot2)
!second method:col by col
implicit none
integer*2 ii, jj, kk, nn
integer*2, dimension(nn,nn) :: m1
```

```
integer*2, dimension(nn,nn) :: m2
integer*2, dimension(nn,nn) :: mris2
real :: finish2, start2,timetot2
call cpu_time (start2)!initial time
do ii=1,nn
do jj=1,nn
do kk=1,nn
mris2(jj,ii)=mris2(jj,ii)+m1(jj,kk)*m2(kk,ii)
!SCALAR PRODUCT BTW
!jj-th ROW OF m1 AND ii-th COLUMN OF m2
!IT GOES IN THE ENTRY (j,i) OF mris2
end do
end do
end do
call cpu_time (finish2)!finish time
timetot2=finish2-start2
end subroutine colcolmatmul


subroutine matmulintrinsic(nn,m1,m2,mris3,timetot3)
!third method: intrisic matmul
implicit none
integer*2 ii, jj, kk, nn
integer*2, dimension(nn,nn) :: m1
integer*2, dimension(nn,nn) :: m2
integer*2, dimension(nn,nn) :: mris3
real :: finish3, start3,timetot3

call cpu_time (start3)!initial time
mris3= matmul(m1,m2)
!matmul DOES A MATRIX-MATRIX MULTIPLICATION m1*m2
call cpu_time (finish3)!finish time
timetot3= finish3-start3
end subroutine matmulintrinsic

end module matmultiplications

!MODULE DEBUGGING
module debugging

interface check
module procedure checkdim,checkmat
end interface

contains
```

```
subroutine checkdim(nn,qq,debug)!checking dimensions nn
implicit none
integer*2 :: nn, qq
logical :: debug

if(debug.eqv..TRUE.) then

if(nn>10000) then !is the dim too large?
print*, "WARNING:␣too␣large␣dimension:"
print*, nn
else if(nn<1) then !is the dim minor than 1?
print*, "WARNING:␣dimension␣minor␣than␣1"
else if((nn-qq*100)>0.5) then !is the dim wrong?
print*, "the␣dimension␣is␣wrong,␣it␣should␣be:"
$                               , qq*100
else
print*, "okay:␣right␣dimensions␣matrices",nn
end if

end if
end subroutine checkdim

subroutine checkmat(nn,m,mcheck,debug)
implicit none
!checking if the input matrixes are equal
integer*2, dimension(nn,nn) :: m, mcheck
integer*2 :: tt,ss,nn
integer*4 accum
logical debug
if(debug.eqv..TRUE.) then

accum=0
do tt=1,nn
do ss=1,nn
if(m(tt,ss)/=mcheck(tt,ss)) then
accum=accum+1
end if
end do
end do

if(accum>0) then
print*, "The␣two␣matrices␣have"
$                       , accum, "different␣entries"
else
print*, "Same␣matrices"
```

```
end if

end if
end subroutine checkmat

end module debugging


program test_performance_mulmat
!This program implements matrix-matrix multiplication
!in three different method: column by column, row by row
!and by intrinsic function. For each method the CPU_TIME
!is computed. This is done for squared matrix, from
!100x100 to 1000x1000, with a step=100.
!The results are printed on a file, called "Results",
!at unit=40.

!DECLARATION VARIABLES

use matmultiplications
use debugging
implicit none

integer*2, dimension(:,:), allocatable :: m1
integer*2, dimension(:,:), allocatable :: m2
integer*2, dimension(:,:), allocatable :: mcheck1, mcheck2
integer*2, dimension(:,:), allocatable :: mris1, mris2, mris3
integer*2  ii, jj, nn, qq
real :: ttime
real :: timetot1, timetot2, timetot3
real :: t1,t2, accumtime, t3,t4
logical :: debug
character*1 :: choice
integer :: my_stat
character (256) :: my_msg
call cpu_time(t1)!INITIAL TIME ALL PROGRAM

accumtime=0.0
choice="X" !Initialization of choice variable
!if it is "X", it asks the debug
!otherwise yoi can choose btw
!doing the debug or not writing "y" or "n"


!Do you want to debug?
```

```
if(choice=="X") then
print *, "Do␣you␣want␣to␣debug?"
print *, "y␣for␣yes,␣n␣for␣no"
read*, choice
end if

if(choice=="y") then
debug=.TRUE. !if sth goes wrong, it becomes false
else if(choice=="n") then
debug=.FALSE.
print*, "Remind:␣following␣no␣okay␣are␣not␣true"
else
print*, "Not␣understood"
stop
end if


!OPENING THE "Result" FILE
open(unit = 40, file = "Results",
$        status = "unknown", access="append",
$  iostat=my_stat, iomsg=my_msg)

if(my_stat /= 0) then
print*, 'Open␣-Results-␣failed␣with␣stat␣=␣'
$                         , my_stat, '␣msg␣=␣'//trim(my_msg)
end if


!HERE STARTS THE CYCLE OVER THE MATRIX DIMENSION/100
do qq= 1, 5

nn=qq*100!MATRIX DIMENSION

!ALLOCATION MATRICES

!DEBUG
call cpu_time(t3)
if(debug.eqv..TRUE.) then!DEBUG
print*, "␣"
print*, "MATRICES␣OF␣ORDER␣", nn
print*, "␣"
end if

if(debug.eqv..TRUE.) then!checkdim
call check(nn,qq,debug)
```

```
end if
call cpu_time(t4)
accumtime=t4-t3

allocate(m1(nn,nn), stat=my_stat, errmsg=my_msg)
!allocating m1
if(my_stat /= 0) then
print*, 'Failed␣to␣allocate␣m1␣with␣stat␣=␣'
$                          , my_stat, '␣and␣msg␣=␣'//trim(
   my_msg)
end if

allocate(m2(nn,nn), stat=my_stat, errmsg=my_msg)
!allocating m2
if(my_stat /= 0) then
print*, 'Failed␣to␣allocate␣m2␣with␣stat␣=␣'
$                          , my_stat, '␣and␣msg␣=␣'//trim(
   my_msg)
end if

allocate(mris1(nn,nn), stat=my_stat, errmsg=my_msg)
!allocating mris1
if(my_stat /= 0) then
print*, 'Failed␣to␣allocate␣mris1␣with␣stat␣=␣'
$                          , my_stat, '␣and␣msg␣=␣'//trim(
   my_msg)
end if

allocate(mris2(nn,nn), stat=my_stat, errmsg=my_msg)
!allocating mris2
if(my_stat /= 0) then
print*, 'Failed␣to␣allocate␣mris2␣with␣stat␣=␣'
$                          , my_stat, '␣and␣msg␣=␣'//trim(
   my_msg)
end if

allocate(mris3(nn,nn), stat=my_stat, errmsg=my_msg)
!allocating mris3
if(my_stat /= 0) then
print*, 'Failed␣to␣allocate␣mris3␣with␣stat␣=␣'
$                          , my_stat, '␣and␣msg␣=␣'//trim(
   my_msg)
end if

allocate(mcheck1(nn,nn), stat=my_stat, errmsg=my_msg)
```

```
!allocating mcheck1
if(my_stat /= 0) then
print*, 'Failed␣to␣allocate␣mcheck1␣with␣stat␣=␣'
$                         , my_stat, '␣and␣msg␣=␣'//trim(
   my_msg)
end if
allocate(mcheck2(nn,nn), stat=my_stat, errmsg=my_msg)
!allocating mcheck2
if(my_stat /= 0) then
print*, 'Failed␣to␣allocate␣mcheck2␣with␣stat␣=␣'
$                         , my_stat, '␣and␣msg␣=␣'//trim(
   my_msg)
end if

!write on file the order of matrix
write(40,*) "␣"
write(40,*) "␣"
write(40,*,iostat=my_stat, iomsg=my_msg)
$          "Matrix␣squared,␣order", nn
write(40,*) "␣"

if(my_stat /= 0) then
print*, 'Write␣-Results-␣failed␣with␣stat␣=␣'
$                         , my_stat, '␣msg␣=␣'//trim(my_msg)
end if

!INITIALIZATION m1: in the entry (i,j)
!the value i+j is assigned
do ii=1,nn
do jj=1,nn
m1(ii,jj)=ii+jj
mcheck1(ii,jj)= m1(ii,jj)
end do
end do

!DEBUG
call cpu_time(t3)
if(debug.eqv..TRUE.) then
print*, "␣"
print*, "INITIALIZATION␣m1"
end if
if(debug.eqv..TRUE.) then!checkdim
call check(nn,qq,debug)
end if
call cpu_time(t4)
```

```
accumtime=t4-t3

!INITIALIZATION m2, in the entry (i,j)
!the value i*j is assigned
do ii=1,nn
do jj=1,nn
m2(ii,jj)=ii*jj
mcheck2(ii,jj)= m2(ii,jj)
end do
end do

!DEBUG
call cpu_time(t3)
if(debug.eqv..TRUE.) then
print*, "␣"
print*, "INITIALIZATION␣m2"
end if
if(debug.eqv..TRUE.) then !checkdim
call check(nn,qq,debug)
end if

if(debug.eqv..TRUE.) then !check matrices
print*, "Imput␣matrix␣1"
call check(nn,m1,mcheck1,debug)
print*, "Imput␣matrix␣2"
call check(nn,m2,mcheck2,debug)
end if
call cpu_time(t4)
accumtime=t4-t3

!FIRST METHOD: row by row in resulting matrix

write(40,*) "␣"
write(40,*,iostat=my_stat, iomsg=my_msg)
$           "␣␣␣␣FIRST␣METHOD:␣row␣by␣row"

if(my_stat /= 0) then
print*, 'Write␣-Results-␣failed␣with␣stat␣=␣'
$                         , my_stat, '␣msg␣=␣'//trim(my_msg)
end if

call cpu_time(ttime)!printing time istant before
print*, ttime
call rowrowmatmul(nn,m1,m2,mris1,timetot1)
call cpu_time(ttime)!printing the time istant after
```

```
print*, ttime

write(40,*,iostat=my_stat, iomsg=my_msg)
$                "Time␣in␣seconds=␣", timetot1

if(my_stat /= 0) then
print*, 'Write␣-Results-␣failed␣with␣stat␣=␣'
$                         , my_stat, '␣msg␣=␣'//trim(my_msg)
end if
!printing on file the cpu_time

!DEBUG
call cpu_time(t3)
if(debug.eqv..TRUE.) then
print*, "␣"
print*, "FIRST␣METHOD:␣row␣by␣row"
end if

if(debug.eqv..TRUE.) then!checkdim
call check(nn,qq,debug)
end if

if(debug.eqv..TRUE.) then!checkmat
print*, "Imput␣matrix␣1"
call check(nn,m1,mcheck1,debug)
print*, "Imput␣matrix␣2"
call check(nn,m2,mcheck2,debug)
end if
call cpu_time(t4)
accumtime=t4-t3

!SECOND METHOD: column by column in resulting matrix

write(40,*) "␣"
write(40,*,iostat=my_stat, iomsg=my_msg)
$           "␣␣␣␣SECOND␣METHOD:␣col␣by␣col"

if(my_stat /= 0) then
print*, 'Write␣-Results-␣failed␣with␣stat␣=␣'
$                         , my_stat, '␣msg␣=␣'//trim(my_msg)
end if


call cpu_time(ttime)!printing the time istant before
print*, ttime
```

```
call colcolmatmul(nn,m1,m2,mris2,timetot2)
call cpu_time(ttime)!printing the time istant after
print*, ttime


write(40,*,iostat=my_stat, iomsg=my_msg)
$                "Time␣in␣seconds=␣", timetot2

if(my_stat /= 0) then
print*, 'Write␣-Results-␣failed␣with␣stat␣=␣'
$                         , my_stat, '␣msg␣=␣'//trim(my_msg)
end if
!printing on file the cpu_time

!DEBUG
call cpu_time(t3)
if(debug.eqv..TRUE.) then
print*, "␣"
print*, "SECOND␣METHOD:␣col␣by␣col"
end if

if(debug.eqv..TRUE.) then!checkdim
call check(nn,qq,debug)
end if
if(debug.eqv..TRUE.) then!checkmat
print*, "Imput␣matrix␣1"
call check(nn,m1,mcheck1,debug)
print*, "Imput␣matrix␣2"
call check(nn,m2,mcheck2,debug)
end if
call cpu_time(t4)
accumtime=t4-t3

!THIRD METHOD: intrinsic function matmul

write(40,*) "␣"
write(40,*,iostat=my_stat, iomsg=my_msg)
$           "␣␣␣␣THIRD␣METHOD:␣intrinsic␣matmul"

if(my_stat /= 0) then
print*, 'Write␣-Results-␣failed␣with␣stat␣=␣'
$                         , my_stat, '␣msg␣=␣'//trim(my_msg)
end if
```

```
call cpu_time(ttime)!printing the time istant before
print*, ttime
call matmulintrinsic(nn,m1,m2,mris3,timetot3)
call cpu_time(ttime)!printing the time istant after
print*, ttime


write(40,*,iostat=my_stat, iomsg=my_msg)
$                "Time␣in␣seconds=␣", timetot3

if(my_stat /= 0) then
print*, 'Write␣-Results-␣failed␣with␣iostat␣=␣'
$                         , my_stat, '␣iomsg␣=␣'//trim(my_msg)
end if
!printing on file the cpu_time


!DEBUG
call cpu_time(t3)
if(debug.eqv..TRUE.) then
print*, "␣"
print*, "THIRD␣METHOD:␣intrinsic␣matmul"
end if
if(debug.eqv..TRUE.) then!checkdim
call check(nn,qq,debug)
end if
if(debug.eqv..TRUE.) then!checkmat
print*, "Imput␣matrix␣1"
call check(nn,m1,mcheck1,debug)
print*, "Imput␣matrix␣2"
call check(nn,m2,mcheck2,debug)
end if

!the resulting matrices are different?
!DEBUG
if(debug.eqv..TRUE.) then
print*, "␣"
print*, "CHECKING␣RESULTING␣MATRICES"
end if
if(debug.eqv..TRUE.) then!checkdim
print*, "Checking␣first␣and␣second␣method"
call check(nn,mris1,mris2,debug)
print*, "Checking␣third␣and␣second␣method"
call check(nn,mris2,mris3,debug)
print*, "Checking␣first␣and␣third␣method"
```

```
call check(nn,mris1,mris3,debug)
end if
call cpu_time(t4)
accumtime=t4-t3

!DEALLOCATION matrices:

deallocate(m1, stat=my_stat, errmsg=my_msg)
!deallocating m1
if(my_stat /= 0) then
print*, 'Failed␣to␣deallocate␣m1␣with␣stat␣=␣'
$                          , my_stat, '␣and␣msg␣=␣'//trim(
   my_msg)
end if


deallocate(m2, stat=my_stat, errmsg=my_msg)
!deallocating m2
if(my_stat /= 0) then
print*, 'Failed␣to␣deallocate␣m2␣with␣stat␣=␣'
$                          , my_stat, '␣and␣msg␣=␣'//trim(
   my_msg)
end if

deallocate(mris1, stat=my_stat, errmsg=my_msg)
!deallocating mris1
if(my_stat /= 0) then
print*, 'Failed␣to␣deallocate␣mris1␣with␣stat␣=␣'
$                          , my_stat, '␣and␣msg␣=␣'//trim(
   my_msg)
end if

deallocate(mris2, stat=my_stat, errmsg=my_msg)
!deallocating mris2
if(my_stat /= 0) then
print*, 'Failed␣to␣deallocate␣mris2␣with␣stat␣=␣'
$                          , my_stat, '␣and␣msg␣=␣'//trim(
   my_msg)
end if

deallocate(mris3, stat=my_stat, errmsg=my_msg)
!deallocating mris3
if(my_stat /= 0) then
print*, 'Failed␣to␣deallocate␣mris3␣with␣stat␣=␣'
$                          , my_stat, '␣and␣msg␣=␣'//trim(
```

```
   my_msg)
end if

deallocate(mcheck1, stat=my_stat, errmsg=my_msg)
!deallocating mcheck1
if(my_stat /= 0) then
print*, 'Failed␣to␣deallocate␣mcheck1␣with␣stat␣=␣'
$                          , my_stat, '␣and␣msg␣=␣'//trim(
   my_msg)
end if

deallocate(mcheck2, stat=my_stat, errmsg=my_msg)
!deallocating mcheck2
if(my_stat /= 0) then
print*, 'Failed␣to␣deallocate␣mcheck2␣with␣stat␣=␣'
$                          , my_stat, '␣and␣msg␣=␣'//trim(
   my_msg)
end if
end do

call cpu_time(t2)!ENDING TIME ALL PROGRAM

if(debug.eqv..TRUE.) then
ttime=(accumtime*100)/(t2-t1)
print*, "␣"
print*, "Time␣spent␣on␣debugging", accumtime
print*, "Total␣time␣spent", t2-t1
print*, "Percentage", ttime
print*, "␣"
end if

stop
end program test_performance_mulmat
```

Results

Everything is debugged

When everything is debugged, what is printed on the teerminal is the following.

```
 Do you want to debug?
y for yes, n for no
y

MATRICES OF ORDER     100
```

```
okay: right dimensions matrices    100

INITIALIZATION m1
okay: right dimensions matrices    100

INITIALIZATION m2
okay: right dimensions matrices    100
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
2.28199991E-03
7.34200003E-03

FIRST METHOD: row by row
okay: right dimensions matrices    100
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
7.46400002E-03
1.22260004E-02

SECOND METHOD: col by col
okay: right dimensions matrices    100
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
1.23330001E-02
1.29180001E-02

THIRD METHOD: intrinsic matmul
okay: right dimensions matrices    100
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices

CHECKING RESULTING MATRICES
Checking first and second method
Same matrices
Checking third and second method
Same matrices
```

```
Checking first and third method
Same matrices

MATRICES OF ORDER     200

okay: right dimensions matrices    200

INITIALIZATION m1
okay: right dimensions matrices    200

INITIALIZATION m2
okay: right dimensions matrices    200
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
1.43409995E-02
5.45089990E-02

FIRST METHOD: row by row
okay: right dimensions matrices    200
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
5.50399981E-02
9.56669971E-02

SECOND METHOD: col by col
okay: right dimensions matrices    200
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
9.60559994E-02
0.100449003

THIRD METHOD: intrinsic matmul
okay: right dimensions matrices    200
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices

CHECKING RESULTING MATRICES
```

```
Checking first and second method
Same matrices
Checking third and second method
Same matrices
Checking first and third method
Same matrices

MATRICES OF ORDER     300

okay: right dimensions matrices    300

INITIALIZATION m1
okay: right dimensions matrices    300

INITIALIZATION m2
okay: right dimensions matrices    300
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
0.103786997
0.235760003

FIRST METHOD: row by row
okay: right dimensions matrices    300
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
0.236818001
0.371093005

SECOND METHOD: col by col
okay: right dimensions matrices    300
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
0.371886998
0.386101991

THIRD METHOD: intrinsic matmul
okay: right dimensions matrices    300
Imput matrix 1
Same matrices
```

```
Imput matrix 2
Same matrices

CHECKING RESULTING MATRICES
Checking first and second method
Same matrices
Checking third and second method
Same matrices
Checking first and third method
Same matrices

MATRICES OF ORDER     400

okay: right dimensions matrices    400

INITIALIZATION m1
okay: right dimensions matrices    400

INITIALIZATION m2
okay: right dimensions matrices    400
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
0.392634988
0.714726985

FIRST METHOD: row by row
okay: right dimensions matrices    400
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
0.716150999
1.07715499

SECOND METHOD: col by col
okay: right dimensions matrices    400
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
1.07869899
1.11326396
```

```
THIRD METHOD: intrinsic matmul
okay: right dimensions matrices    400
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices

CHECKING RESULTING MATRICES
Checking first and second method
Same matrices
Checking third and second method
Same matrices
Checking first and third method
Same matrices

MATRICES OF ORDER     500

okay: right dimensions matrices    500

INITIALIZATION m1
okay: right dimensions matrices    500

INITIALIZATION m2
okay: right dimensions matrices    500
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
1.12408805
1.76041400

FIRST METHOD: row by row
okay: right dimensions matrices    500
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
1.76280308
2.44460201

SECOND METHOD: col by col
okay: right dimensions matrices    500
Imput matrix 1
Same matrices
Imput matrix 2
```

```
Same matrices
2.44697905
2.51211214

THIRD METHOD: intrinsic matmul
okay: right dimensions matrices    500
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices

CHECKING RESULTING MATRICES
Checking first and second method
Same matrices
Checking third and second method
Same matrices
Checking first and third method
Same matrices

Time spent on debugging   6.33788109E-03
Total time spent   2.51670909
Percentage  0.251832098
```

The actual results (the compilation times of the three methods) are in the file called "Results".

```
Matrix squared, order    100


FIRST METHOD: row by row
Time in seconds=    5.77600021E-03

SECOND METHOD: col by col
Time in seconds=    5.55899972E-03

THIRD METHOD: intrinsic matmul
Time in seconds=    7.59999268E-04


Matrix squared, order    200


FIRST METHOD: row by row
```

```
Time in seconds=    4.66850027E-02

SECOND METHOD: col by col
Time in seconds=    4.82419990E-02

THIRD METHOD: intrinsic matmul
Time in seconds=    5.24000078E-03


Matrix squared, order    300


FIRST METHOD: row by row
Time in seconds=   0.159464002

SECOND METHOD: col by col
Time in seconds=   0.170147002

THIRD METHOD: intrinsic matmul
Time in seconds=    1.77990198E-02


Matrix squared, order    400


FIRST METHOD: row by row
Time in seconds=   0.414929032

SECOND METHOD: col by col
Time in seconds=   0.409626961

THIRD METHOD: intrinsic matmul
Time in seconds=    4.08509970E-02


Matrix squared, order    500


FIRST METHOD: row by row
Time in seconds=   0.752187967

SECOND METHOD: col by col
Time in seconds=   0.788024902

THIRD METHOD: intrinsic matmul
```

```
Time in seconds=    7.79271126E-02
```

Forced bug: changing dimensions

There is a forced bug: after the second method the dimension of the matrices is increased by one via $nn = nn + 1$. The CHECKDIM subroutine detects this fact. After then, everything is messed up and the execution is aborted.

```
 Do you want to debug?
y for yes, n for no
y

MATRICES OF ORDER     100

okay: right dimensions matrices    100

INITIALIZATION m1
okay: right dimensions matrices    100

INITIALIZATION m2
okay: right dimensions matrices    100
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
1.66299997E-03
6.55200006E-03

FIRST METHOD: row by row
okay: right dimensions matrices    100
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
6.66699978E-03
1.14230001E-02

SECOND METHOD: col by col
okay: right dimensions matrices    100
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
1.15379998E-02
1.21430000E-02
```

```
THIRD METHOD: intrinsic matmul
the dimension is wrong, it should be:         100
but is    101

Imput matrix 1
The two matrices have         193 different entries
Imput matrix 2
The two matrices have         195 different entries

CHECKING RESULTING MATRICES
Checking first and second method
The two matrices have         193 different entries
Checking third and second method
The two matrices have       10201 different entries
Checking first and third method
The two matrices have       10201 different entries
free(): invalid next size (normal)

Program received signal SIGABRT: Process abort signal.

Backtrace for this error:
#0  0x7f83e15dd31a
#1  0x7f83e15dc503
#2  0x7f83e120ff1f
#3  0x7f83e120fe97
#4  0x7f83e1211800
#5  0x7f83e125a896
#6  0x7f83e1261909
#7  0x7f83e12690ac
#8  0x55d0e2f7273d
#9  0x55d0e2f73181
#10  0x7f83e11f2b96
#11  0x55d0e2f6cbc9
#12  0xffffffffffffffff
Aborted (core dumped)
```

Forced bug: changing the imput matrices

There is a forced bug: after the first method the first matrix is changed via $m1(1, 1) = 9$. The CHECKMAT subroutine detects this fact.

```
 Do you want to debug?
y for yes, n for no
y
```

```
MATRICES OF ORDER     100

okay: right dimensions matrices    100

INITIALIZATION m1
okay: right dimensions matrices    100

INITIALIZATION m2
okay: right dimensions matrices    100
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
1.29799999E-03
6.32400019E-03

FIRST METHOD: row by row
okay: right dimensions matrices    100
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
6.43400010E-03
1.12030003E-02

SECOND METHOD: col by col
okay: right dimensions matrices    100
Imput matrix 1
The two matrices have           1 different entries
Imput matrix 2
Same matrices
1.13100000E-02
1.19009996E-02

THIRD METHOD: intrinsic matmul
okay: right dimensions matrices    100
Imput matrix 1
The two matrices have           1 different entries
Imput matrix 2
Same matrices

CHECKING RESULTING MATRICES
Checking first and second method
The two matrices have         100 different entries
Checking third and second method
```

```
Same matrices
Checking first and third method
The two matrices have         100 different entries

MATRICES OF ORDER     200

okay: right dimensions matrices    200

INITIALIZATION m1
okay: right dimensions matrices    200

INITIALIZATION m2
okay: right dimensions matrices    200
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
1.35390004E-02
5.32040000E-02

FIRST METHOD: row by row
okay: right dimensions matrices    200
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
5.35769984E-02
9.42099988E-02

SECOND METHOD: col by col
okay: right dimensions matrices    200
Imput matrix 1
The two matrices have           1 different entries
Imput matrix 2
Same matrices
9.45810005E-02
9.86849964E-02

THIRD METHOD: intrinsic matmul
okay: right dimensions matrices    200
Imput matrix 1
The two matrices have           1 different entries
Imput matrix 2
Same matrices
```

```
CHECKING RESULTING MATRICES
Checking first and second method
The two matrices have         200 different entries
Checking third and second method
Same matrices
Checking first and third method
The two matrices have         200 different entries

MATRICES OF ORDER     300

okay: right dimensions matrices    300

INITIALIZATION m1
okay: right dimensions matrices    300

INITIALIZATION m2
okay: right dimensions matrices    300
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
0.102197997
0.232566997

FIRST METHOD: row by row
okay: right dimensions matrices    300
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
0.233350992
0.368283987

SECOND METHOD: col by col
okay: right dimensions matrices    300
Imput matrix 1
The two matrices have           1 different entries
Imput matrix 2
Same matrices
0.368952990
0.384099990

THIRD METHOD: intrinsic matmul
okay: right dimensions matrices    300
Imput matrix 1
```

```
The two matrices have           1 different entries
Imput matrix 2
Same matrices

CHECKING RESULTING MATRICES
Checking first and second method
The two matrices have         300 different entries
Checking third and second method
Same matrices
Checking first and third method
The two matrices have         300 different entries

MATRICES OF ORDER     400

okay: right dimensions matrices    400

INITIALIZATION m1
okay: right dimensions matrices    400

INITIALIZATION m2
okay: right dimensions matrices    400
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
0.391400993
0.725878000

FIRST METHOD: row by row
okay: right dimensions matrices    400
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
0.727531016
1.05936301

SECOND METHOD: col by col
okay: right dimensions matrices    400
Imput matrix 1
The two matrices have           1 different entries
Imput matrix 2
Same matrices
1.06103003
1.09426296
```

```
THIRD METHOD: intrinsic matmul
okay: right dimensions matrices    400
Imput matrix 1
The two matrices have           1 different entries
Imput matrix 2
Same matrices

CHECKING RESULTING MATRICES
Checking first and second method
The two matrices have         400 different entries
Checking third and second method
Same matrices
Checking first and third method
The two matrices have         400 different entries

MATRICES OF ORDER     500

okay: right dimensions matrices    500

INITIALIZATION m1
okay: right dimensions matrices    500

INITIALIZATION m2
okay: right dimensions matrices    500
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
1.10672605
1.73068500

FIRST METHOD: row by row
okay: right dimensions matrices    500
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
1.73343897
2.39542508

SECOND METHOD: col by col
okay: right dimensions matrices    500
Imput matrix 1
The two matrices have           1 different entries
```

```
Imput matrix 2
Same matrices
2.39881110
2.46275592

THIRD METHOD: intrinsic matmul
okay: right dimensions matrices    500
Imput matrix 1
The two matrices have           1 different entries
Imput matrix 2
Same matrices

CHECKING RESULTING MATRICES
Checking first and second method
The two matrices have         500 different entries
Checking third and second method
Same matrices
Checking first and third method
The two matrices have         500 different entries

Time spent on debugging   8.14318657E-03
Total time spent   2.47011685
Percentage  0.329668075
```

Forced bug: changing the resulting matrices

There is a forced bug: after the third method the second resulting matrix is changed via $mris1(2,2) = 6$. The CHECKMAT subroutine detects this fact. It is noticeable that the time spent on debugging increases.

```
 Do you want to debug?
y for yes, n for no
y

MATRICES OF ORDER     100

okay: right dimensions matrices    100

INITIALIZATION m1
okay: right dimensions matrices    100

INITIALIZATION m2
okay: right dimensions matrices    100
Imput matrix 1
Same matrices
```

```
Imput matrix 2
Same matrices
1.44300004E-03
6.21999986E-03

FIRST METHOD: row by row
okay: right dimensions matrices    100
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
6.34300010E-03
1.10090002E-02

SECOND METHOD: col by col
okay: right dimensions matrices    100
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
1.11170001E-02
1.18779996E-02

THIRD METHOD: intrinsic matmul
okay: right dimensions matrices    100
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices

CHECKING RESULTING MATRICES
Checking first and second method
The two matrices have           1 different entries
Checking third and second method
The two matrices have           1 different entries
Checking first and third method
Same matrices

MATRICES OF ORDER     200

okay: right dimensions matrices    200

INITIALIZATION m1
okay: right dimensions matrices    200
```

```
INITIALIZATION m2
okay: right dimensions matrices    200
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
1.31909996E-02
5.43610007E-02

FIRST METHOD: row by row
okay: right dimensions matrices    200
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
5.48840016E-02
9.45490003E-02

SECOND METHOD: col by col
okay: right dimensions matrices    200
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
9.50509980E-02
9.91759971E-02

THIRD METHOD: intrinsic matmul
okay: right dimensions matrices    200
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices

CHECKING RESULTING MATRICES
Checking first and second method
The two matrices have           1 different entries
Checking third and second method
The two matrices have           1 different entries
Checking first and third method
Same matrices

MATRICES OF ORDER     300

okay: right dimensions matrices    300
```

```
INITIALIZATION m1
okay: right dimensions matrices    300

INITIALIZATION m2
okay: right dimensions matrices    300
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
0.102418996
0.232794002

FIRST METHOD: row by row
okay: right dimensions matrices    300
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
0.233592004
0.368221998

SECOND METHOD: col by col
okay: right dimensions matrices    300
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
0.369170994
0.384175003

THIRD METHOD: intrinsic matmul
okay: right dimensions matrices    300
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices

CHECKING RESULTING MATRICES
Checking first and second method
The two matrices have          1 different entries
Checking third and second method
The two matrices have          1 different entries
Checking first and third method
Same matrices
```

```
MATRICES OF ORDER      400

okay: right dimensions matrices     400

INITIALIZATION m1
okay: right dimensions matrices     400

INITIALIZATION m2
okay: right dimensions matrices     400
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
0.391465992
0.706444979

FIRST METHOD: row by row
okay: right dimensions matrices     400
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
0.707970023
1.03191900

SECOND METHOD: col by col
okay: right dimensions matrices     400
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
1.03368497
1.06721997

THIRD METHOD: intrinsic matmul
okay: right dimensions matrices     400
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices

CHECKING RESULTING MATRICES
Checking first and second method
The two matrices have           1 different entries
```

```
Checking third and second method
The two matrices have           1 different entries
Checking first and third method
Same matrices

MATRICES OF ORDER     500

okay: right dimensions matrices    500

INITIALIZATION m1
okay: right dimensions matrices    500

INITIALIZATION m2
okay: right dimensions matrices    500
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
1.07878006
1.73125196

FIRST METHOD: row by row
okay: right dimensions matrices    500
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
1.73404598
2.51496005

SECOND METHOD: col by col
okay: right dimensions matrices    500
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
2.51797199
2.58225608

THIRD METHOD: intrinsic matmul
okay: right dimensions matrices    500
Imput matrix 1
Same matrices
Imput matrix 2
Same matrices
```

```
CHECKING RESULTING MATRICES
Checking first and second method
The two matrices have           1 different entries
Checking third and second method
The two matrices have           1 different entries
Checking first and third method
Same matrices

Time spent on debugging   6.30521774E-03
Total time spent   2.58778501
Percentage  0.243653074
```

Self-Evaluation

The main goals achieved:

- Layering the code: using modules and subroutines instead of writing everything in the program.
- Documenting the code: documentation of the subroutines and of the program, checkpoints and comments. This is in order to make the code user-friendly.
- Debugging the code: checkpoints, error handling and pre-conditions and post-conditions. This in order to make the debugging easier.
- Giving to the user the choice to debug the code. Morever, giving to the programmer the choice to give that choice to the user.
- Printing how long the debugging costed in terms of time, in respect of the total time spent by the program on the execution.

Results

Risultati del secondo esercizio.

```
Matrix squared, order    100


FIRST METHOD: row by row
Time in seconds=    5.77600021E-03

SECOND METHOD: col by col
```

```
Time in seconds=    5.55899972E-03

THIRD METHOD: intrinsic matmul
Time in seconds=    7.59999268E-04


Matrix squared, order    200


FIRST METHOD: row by row
Time in seconds=    4.66850027E-02

SECOND METHOD: col by col
Time in seconds=    4.82419990E-02

THIRD METHOD: intrinsic matmul
Time in seconds=    5.24000078E-03


Matrix squared, order    300


FIRST METHOD: row by row
Time in seconds=   0.159464002

SECOND METHOD: col by col
Time in seconds=   0.170147002

THIRD METHOD: intrinsic matmul
Time in seconds=    1.77990198E-02


Matrix squared, order    400


FIRST METHOD: row by row
Time in seconds=   0.414929032

SECOND METHOD: col by col
Time in seconds=   0.409626961

THIRD METHOD: intrinsic matmul
Time in seconds=    4.08509970E-02
```

```
Matrix squared, order    500


FIRST METHOD: row by row
Time in seconds=   0.752187967

SECOND METHOD: col by col
Time in seconds=   0.788024902

THIRD METHOD: intrinsic matmul
Time in seconds=    7.79271126E-02
```