Quantum Information and Computing

prof: *Simone Montangero*

Report of exercise 2

Alberto Bassi

October 19, 2020

Abstract

The aim of this exercise is to learn how to use modules and derived types in Fortran90. In particular, we will write a module which contains a double complex derived type MATRIX and some functions and subroutines. We will include this module in a simple program and we will show some results.

Code Development

'Ex2_Bassi_Code_mat2.f'

```
MODULE MATRICES
        IMPLICIT NONE
        !Define the type matrix
        TYPE MATRIX
           DOUBLE COMPLEX, DIMENSION (:,:), ALLOCATABLE :: mat
           INTEGER, DIMENSION(2) :: dimensions
           DOUBLE COMPLEX :: trace
           DOUBLE COMPLEX :: determinant
           CHARACTER*20 :: mat_name
        END TYPE MATRIX
        !Define the two interface operators for calculating the trace and the
    adjoint matrix
        INTERFACE OPERATOR (.adj.)
        MODULE PROCEDURE adjoint
        END INTERFACE
        INTERFACE OPERATOR (.trc.)
        MODULE PROCEDURE trace
        END INTERFACE
        CONTAINS
        !Define the function to compute the trace
        DOUBLE COMPLEX FUNCTION trace(tmp)
           IMPLICIT NONE
           TYPE(MATRIX), INTENT(IN) :: tmp
           INTEGER :: ss
           trace=(0d0,0d0)
           IF (tmp%dimensions(1).eq.tmp%dimensions(2)) THEN !Control if the matrix
    is square. If the matrix is rectangular this function does nothing, so the trace
    remains (0,0)
              DO ss=1,tmp%dimensions(1)
                 trace=trace +tmp%mat(ss,ss)
              ENDDO
           END IF
        END
        !Initialize the matrix
        SUBROUTINE init_mat (tmp)
           IMPLICIT NONE
           TYPE(MATRIX) :: tmp
           INTEGER :: nn,mm,ii,jj
           DOUBLE PRECISION :: xx,yy
           tmp%mat_name="A_matrix"
           tmp%mat_name=TRIM(tmp%mat_name)
           PRINT *, "Insert the dimension of the matrix nxm"
           READ *, nn,mm
           tmp%dimensions(1)=nn    !Initialize the dimensions
           tmp%dimensions(2)=mm
           ALLOCATE(tmp%mat(nn,mm)) !Allocate the memory
           !We fill the matrix with random numbers
           DO jj=1,mm
              DO ii=1,nn
                 CALL RANDOM_NUMBER(xx)
                 CALL RANDOM_NUMBER(yy)
                 tmp%mat(ii,jj)=COMPLEX(xx,yy)
              ENDDO
           ENDDO
           !We initialize the trace and the determinant
           tmp%trace=.trc.tmp
          tmp%determinant=(1d0,1d0) !We don't compute the det. actually, we set it to
     (1,1) for all the matrices, even if they are rectangular
        END
        !Calculate and initialize the adjoint matrix
        TYPE(MATRIX) FUNCTION adjoint(tmp)
           IMPLICIT NONE
```

```
            TYPE(MATRIX),INTENT(IN) :: tmp
            INTEGER :: ii,jj,aa,bb
            adjoint%mat_name="adjoint_"//tmp%mat_name
            adjoint%mat_name=TRIM(adjoint%mat_name)
            adjoint%dimensions(1)=tmp%dimensions(2)
            adjoint%dimensions(2)=tmp%dimensions(1)
            aa=adjoint%dimensions(1)
            bb=adjoint%dimensions(2)
            ALLOCATE(adjoint%mat(aa,bb)) !Allocate the memory for the adjoint matrix
            DO jj=1,adjoint%dimensions(2)
               DO ii=1,adjoint%dimensions(1)
                  adjoint%mat(ii,jj)=CONJG(tmp%mat(jj,ii))
               ENDDO
            ENDDO
            !We use the properties of the adjoint matrix for calculating the trace end the determinant
            adjoint%trace=CONJG(tmp%trace)
            adjoint%determinant=CONJG(tmp%determinant)
         END
         !This subroutine allows us to view the matrix and its properties defined in the type
         SUBROUTINE view_mat(tmp)
            IMPLICIT NONE
            TYPE(MATRIX) :: tmp
            INTEGER :: ii,jj
            PRINT *, "The dimensions are: ", tmp%dimensions
            PRINT *, "The matrix ",tmp%mat_name," is (Re,Im): "
            DO ii=1,tmp%dimensions(1)
               PRINT *, (tmp%mat(ii,jj), jj=1,tmp%dimensions(2))
            ENDDO
            IF (tmp%dimensions(1).eq.tmp%dimensions(2)) THEN
               PRINT *,"The trace is:", tmp%trace
               PRINT *,"The determinant is:", tmp%determinant
            ELSE
               PRINT *,"The trace for this matrix is not defined "
               PRINT *,"The determinant for this matrix is not defined "
            END IF
         END
         !Print the matrix in a file whose name is required as an input
         SUBROUTINE MYOPEN (tmp)
            IMPLICIT NONE
            CHARACTER*20 :: output
            TYPE(MATRIX) :: tmp
            INTEGER :: ii,jj
            PRINT *,"Insert the name of the file"
            READ *, output
            output=TRIM(output)
            OPEN(UNIT=20,FILE=output,STATUS='unknown')
            WRITE(20,*) "The dimensions are:", tmp%dimensions
            WRITE(20,*) "The matrix ",tmp%mat_name," is (Re,Im): "
            DO ii=1,tmp%dimensions(1)
               WRITE(20,*) (tmp%mat(ii,jj), jj=1,tmp%dimensions(2))
            ENDDO
            IF (tmp%dimensions(1).eq.tmp%dimensions(2)) THEN
               WRITE(20,*)"The trace is:", tmp%trace
               WRITE(20,*)"The determinant is:", tmp%determinant
            ELSE
               WRITE(20,*) "The trace is not defined"
               WRITE(20,*) "The determinant is not defined"
            END IF
            CLOSE(20)
            RETURN
         END
      END MODULE
      PROGRAM mat2
         USE MATRICES
```

```
        IMPLICIT NONE
        TYPE(MATRIX) :: A,B
        CALL init_mat(A)
        CALL view_mat(A)
        CAll MYOPEN(A)
        B=.adj.A
        CALL view_mat(B)
        CALL MYOPEN(B)
    END
```

This is the code for the exercise. At first, we defined a module *MATRICES* which contains a derived type *MATRIX*[1] that represents a double complex matrix amid some of its properties, namely its name, dimensions, trace and determinant. Moreover, the module contains two interface operators, the two functions these interface operators represent and three subroutines. Then, we wrote down a simple program which uses this module.
The subroutine *init_ mat* requires the dimensions of the matrix as input, sets them to the correspondent attribute of the type, allocates the memory and fills the allocated space with random numbers, taken from a uniform distribution in [-1,1] for the real part, and a uniform distribution in [-i,i] for the imaginary part. Then, it sets the matrix's trace, calculated with the function *trace* only when the matrix is square (for if not, the trace is set to the default value (0,0)), and its determinant. We do not compute, however, the determinant. Instead, we set it equal to the complex number (1,1) as default. The function *adjoint* gives as output a type *MATRIX*, whose matrix is the adjoint matrix of the input, along with its properties defined in the derived type. Due to the properties of the adjoint matrix, the trace and the determinant are straightforward to compute, since they are only the complex conjugates of the input matrix's trace and determinant respectively. The two interface operators are defined as *.adj.* and *.trc.* and represent the functions *trace* and *adjoint* respectively. The subroutine *view_ mat* takes as input a type *MATRIX* and prints the attributes (including the name) on the terminal. Moreover, when the matrix is rectangular it tells us that the trace and determinant are not defined (even though, they clearly have any default value). Instead, the subroutine *MYOPEN* writes the attributes of the matrix given as input in a file, whose name is expected to be inserted on the terminal.
The following program *mat2* asks the user the dimensions of the matrix, then it creates the correspondent type *MATRIX*. Afterwards, it shows the matrix and the properties on the terminal and requires the name of the file on which it has to write the output. Then, it repeats the procedure for the adjoint matrix before stopping.

Results

If we ask for a 2x3 matrix, then we get:

```
(base) ab77@alberto:~/Documents/Quantum_Information/Report_ex2/ex2$ ./mat2.out
 Insert the dimension of the matrix nxm
2
3
 The dimensions are:            2           3
 The matrix A_matrix              is (Re,Im):
              (0.65738275743690244,0.68671143755492237)
    (0.84200436139211920,0.42576445734908308)
    (0.91968028058846207,0.89405730727793031)
              (0.37633656170568142,0.97747680859466557)        (7.39793162100059742E
    -002,0.19731595986466688)            (0.29063927374248666,0.54866253873699544)
 The trace for this matrix is not defined
 The determinant for this matrix is not defined
 Insert the name of the file
mat_2x3.txt
 The dimensions are:            3           2
 The matrix adjoint_A_matrix      is (Re,Im):
             (0.65738275743690244,-0.68671143755492237)
    (0.37633656170568142,-0.97747680859466557)
```

[1] We use the capital letters for referring to the derived type, while the lowercase letters for the usual matrix.

```
            (0.84200436139211920,-0.42576445734908308)      (7.39793162100059742E-002,-0.19731595986466688)
            (0.91968028058846207,-0.89405730727793031)     (0.29063927374248666,-0.54866253873699544)
 The trace for this matrix is not defined
 The determinant for this matrix is not defined
 Insert the name of the file
adjoint_mat_2x3.txt
(base) ab77@alberto:~/Documents/Quantum_Information/Report_ex2/ex2$ cat mat_2x3.txt
 The dimensions are:           2           3
 The matrix A_matrix             is (Re,Im):
             (0.65738275743690244,0.68671143755492237)     (0.84200436139211920,0.42576445734908308)     (0.91968028058846207,0.89405730727793031)
             (0.37633656170568142,0.97747680859466557)       (7.39793162100059742E-002,0.19731595986466688)            (0.29063927374248666,0.54866253873699544)
 The trace is not defined
 The determinant is not defined
(base) ab77@alberto:~/Documents/Quantum_Information/Report_ex2/ex2$ cat adjoint_mat_2x3.txt
 The dimensions are:           3           2
 The matrix adjoint_A_matrix     is (Re,Im):
            (0.65738275743690244,-0.68671143755492237)     (0.37633656170568142,-0.97747680859466557)
            (0.84200436139211920,-0.42576445734908308)      (7.39793162100059742E-002,-0.19731595986466688)
            (0.91968028058846207,-0.89405730727793031)     (0.29063927374248666,-0.54866253873699544)
 The trace is not defined
 The determinant is not defined
```

We can notice that the matrix is correctly initialized and the adjoint matrix correctly calculated. The trace and determinant result to be not defined, as it has to be since the matrices are rectangular. If, however, the reader was interested in the output of a square matrix, he could consult the file "square.txt", for the correspondent results could not be included here due to the limited space. Even in this case, the function *trace* appears to be working properly.

Self Evaluation

By solving this exercise, we have learned how to write and use modules and derived types in Fortran90. In particular, a very nice method we have learned is defining interface operators for having a more convenient way to use functions. Moreover, we have learned how to use properly subroutines and functions along with their differences. In particular, a subroutine may be thought as the void function in C++. Last, but not least, we have learned how to generated uniformly distributed random numbers, a fundamental tool in Computer Science as well as in Physics.
However, we encountered a few problems, due to the fact we were new at writing in Fortran. For example, it appeared complicated to define correctly the complex number needed for passing to the matrix the randomly generated values. That had been the issue until we found out that the simple call zz=COMPLEX(xx,yy) was indeed able to solve the problem. This was not obvious a priori and it made us to waste a great amount of time. Although, in my opinion this wasting of time is necessary at the very beginning in order to learn a new programming language, because that will allow us to learn how to save time later.