Quantum information and Computing
Ex. 02

Andrea Nicolai - 1233407

October 19, 2020

Abstract

In this work we present a new module in $FORTRAN90$, which introduces a new derived type which we called $CMATRIX$. It allows us to deal with double complex matrices and compute some interesting quantities by the mean of some operators we implemented, sticking to their definition, such as the adjoint matrix and the Trace. Finally, we introduce a subroutine to print to a text file a matrix and some quantities of it, namely the trace and the determinant, if defined. Some checks are implemented to understand whether we are dealing with square matrices and set output accordingly.

Theory

We define in *Fortran90* a new module, which contains a new derived type (i.e. CMATRIX) that allows us to deal with double complex matrices and some of their most useful operators, that is the Adjoint and the Trace. In this module there are also two variables for the storage of the values of Trace and Determinant.
In particular the Adjoint operator is defined for square matrices, that is $(N \times N)$. Given the matrix A, its adjoint will be the matrix $A^\dagger$ such that its generic element is:

$$A^\dagger_{ij} = A^*_{ji} \tag{1}$$

Where the symbol $*$ stands for the conjugate operator.
The second operator we introduced is the $Trace$, denoted as $Tr(A)$, that is defined as the sum of the element along the main diagonal of a $(N \times N)$ matrix. We assume therefore A to be square. Formally:

$$Tr(A) = \sum_{i=1}^{N} a_{ii} \tag{2}$$

An identity one can check is the following one:

$$Tr(A) = Tr^*(A^\dagger) \tag{3}$$

Code Development

First of all, we defined the derived type which was based on double complex 2-dim array, namely the matrix itself and its elements. Therefore some operators were defined in order to deal with these entities, and some double complex variables were introduced to store, as an example, the determinant and the trace. We pass to the initializer functions two integers that set the dimensions of our matrix, and that will be saved into a 2-dim integer array. One should note as *DO* cycle operates along a single column first, in order to take into account the way *FORTRAN* stores arrays, hence optimizing the code.
During the development of this module we implemented some checks to handle some errors that may occur. In the case we want to properly initialize a matrix, whose elements maybe either equal to a certain value

```
FUNCTION INIT_MATRIX_VALUE(rows, columns, myvalue)
```

or drawn randomly

```
FUNCTION INIT_MATRIX_RANDOM(rows, columns)
```

in the interval $[-1, +1]$, it is needed to correctly define its dimensions: in the functions we implemented, we return a message and exit the program with flag 1 if either one of the integers we pass to the initializer is null or negative.
We mentioned above that the adjoint and the trace operators are defined only for square matrices. Therefore we introduced a check on the dimensions of the matrix: if the two dimensions are different, then we set arbitrarily the trace equal to zero and print this statement via output. Moreover, its adjoint is not defined as well and returns to the main program printing on screen that matrix is not square.
Finally, we have implemented a subroutine that prints into a text file the aforementioned matrix, whose elements are written in the exponential notation, and its trace and determinant. For doing this, the string for the format parameter had to be set accordingly.

Results

The module was implemented in a simple test program that initializes a square matrix, given its dimensions, and either draw randomly or set to a predefined complex value all its elements. Next, it computes its trace and adjoint matrix if possible, that is if the matrix is square. Finally, it prints to a text file both the matrix and its determinant and trace. Another call to the subroutine which prints to an other output file is made, in order to print the adjoint matrix as well, given that the input one is square. Checking them, the operator does correctly its job, being also respected the identity 3.
No other problem has occurred so far, given that the matrix was initialized as square and, if not, we tested that all the checks implemented have been working properly.

Self Evaluation

For this work we wrote a module, containing a derived type in order to deal with complex matrices and its main quantities. After having introduced some basic and useful operators, by the mean of intrinsic functions that are for sure better optimized, further works can be done to implement some less ”immediate” operators, as well as some more checks based on different linear algebra identities.
In addition, there might be better ways in order to deal with non-square matrices: as an example we could have generalized the Adjoint operator also to rectangular $(M \times N)$ matrices, simply neglecting the check in the code, or alternatively exit the program with a different flag from the one in initialization (we recall is 1). This actually differs from what may happen with the Trace operator if the dimensions are not equal: we might have added either a number $\Delta = |M - N|$ rows or columns, filled with 0s, in order to make the Matrix square and finally compute the Trace. An other option would be again exit the program with a certain flag, as mentioned before for the Adjoint matrix or, as we did, set the Trace arbitrarily to a null value and print a message explaining what actually occurred.
In addition, the code might have been implemented in such way that, when running the executable, some arguments were to be passed: as examples the dimensions of the matrix and/or a flag whether to initialize it by setting all its elements to a predefined value or random ones.