Quantum Information and Computing 2020/21
Week 2 report

Rocco Ardino
(Dated: Wednesday 21st October, 2020)

We discuss the creation of a new module in Fortran90, containing a double precision complex matrix derived type, and including useful properties such as the matrix size, trace and determinant. We implement the code needed for this purpose along with debug checks, printing functions and "constructor" functions to initialize variables of the new type. Lastly, we check the correct behaviour in a test program to validate the correctness of the module.

1 Theory

Given a complex square matrix $A = [a_{ij}]$, where $i = 1, \dots, n$ and $j = 1, \dots, n$, its trace is computed through:

$$\mathrm{tr}[A] = \sum_{i=1}^{n} a_{ii} \quad . \tag{1}$$

In general, given a complex matrix $A = [a_{ij}]$, where $i = 1, \dots, n$ and $j = 1, \dots, m$, its adjoint is computed through:

$$\mathrm{adj}[A] := A^\dagger = [a^*_{ji}] \quad . \tag{2}$$

Note that when computing the adjoint of a square matrix, the trace of the new matrix is the complex conjugate of the original matrix.

Lastly, given A a $n \times n$ square matrix and S_n the set of all the permutations σ of the set $\{1, 2, \dots, n\}$, the determinant of A can be obtained from Leibniz formula:

$$\det(A) = \sum_{\sigma \in S_n} \left(\mathrm{sgn}(\sigma) \prod_{i=1}^{n} a_{i,\sigma_i} \right) \quad . \tag{3}$$

However, this algorithm is highly inefficient for large n since its complexity is $\mathcal{O}(n!)$. A good alternative is to exploit the "LU" decomposition of a square matrix $A = LU$ in order to make feasible this computation for large matrices. Therefore, the new determinant computation algorithm reads:

$$\det(A) = \det(L)\det(U) = \left(\prod_{i=1}^{n} l_{ii} \right) \left(\prod_{i=1}^{n} u_{ii} \right) \quad . \tag{4}$$

2 Code Development

The first operation is the definition of the new module, called **`zmatmod`**. Inside it, there are:

- the definition of the new type, called **`zmat`**. It contains:
  - ▷ an array of two 2-byte integers, namely n and m, representing the matrix dimensions;
  - ▷ an array of dimensions (n, m) of double precision complex numbers, storing the matrix elements;
  - ▷ a double precision complex, storing the trace;

- ▷ a double precision complex, storing the determinant;

- the definition of several functions, in particular:
  - ▷ **`zmat_init_zero(dim)`**: it initializes a `zmat` variable with matrix dimensions given by the input array `dim`. Every entry of the matrix is set to zero;
  - ▷ **`zmat_init_rand(dim)`**: it behaves like `zmat_init_zero`, but the matrix entries are set to uniform random numbers inside $(0, 1)$;
  - ▷ **`zmat_tr(a)`**: it computes and returns the trace of the matrix contained in the `zmat` input argument, if the dimensions are the same. Otherwise, it returns an error message;
  - ▷ **`zmat_adj(a)`**: it computes and returns the adjoint of the `zmat` input argument. Therefore, it returns a `zmat` object containing the inverted dimensions of the input, the adjoint of the matrix, the conjugate of the trace and of the determinant (if the dimensions are the same);
  - ▷ **`zmat_det(a)`**: it computes and returns the determinant of the matrix contained in the `zmat` input argument, if the dimensions are the same. Otherwise, it returns an error message. Note that the calculation is not yet implemented, so the result is set to 1 for convention;

- the definition of several subroutines, in particular:
  - ▷ **`zmat_print_std(a,formatted)`**: it prints on the standard output, namely on the terminal, the complex matrix of the `zmat` input argument `a` in a human readable form. If the argument flag `formatted` is enabled, the numbers are truncated and formatted in order to have a cleaner output;
  - ▷ **`zmat_print_file(a,fname,unit,formatted)`**: it does the work of `zmat_print_std`, but on a file `fname` associated to the unit argument `unit`;

- some operators that are an interface to a certain function, in particular:
  - ▷ **`.tr.`**: it is an interface to `zmat_tr`, so it computes and returns the trace;
  - ▷ **`.adj.`**: it is an interface to `zmat_adj`, so it computes and returns the adjoint;
  - ▷ **`.det.`**: it is an interface to `zmat_det`, so it computes and returns the determinant.

In order to test the new module, the latter is inserted inside a test program where simple operations are done using the new type. Its compilation does not require particular compilation flags. This test program is showed in Listing 1.

```fortran
program test
    use zmatmod
    implicit none

    integer*2, dimension(2) :: dim
    type(zmat) :: a                                 ! declare zmat var of name 'a'
    type(zmat) :: a_adj                             ! declare zmat var for adj(a)

    dim(1) = 5                                      ! fix a%elem dimensions to (5,5)
    dim(2) = 5

    a = zmat_init_rand(dim)                         ! initialize randomly 'a'
    a_adj = .adj.a                                  ! adjoint of 'a'

    print *, "zmat variable 'a':"
    call zmat_print_std(a, .TRUE.)                  ! print zmat type on terminal
    print *, NEW_LINE('a'), "Adjoint of 'a':"
    call zmat_print_std(a_adj, .TRUE.)              ! print zmat adjoint on terminal

    print *, NEW_LINE('a'), "Writing 'a' variable on file..."
    call zmat_print_file(a, "zmat.txt", 20, .TRUE.) ! write zmat type on file

    print *, NEW_LINE('a'), "Check if new operators work fine..."
    print *, "Trace of a:            ", .tr.a
    print *, "Trace of adj(a):       ", .tr.a_adj
    print *, "Determinant of a:      ", .det.a       ! Not yet implemented
    print *, "Determinant of adj(a):", .det.a_adj   ! Not yet implemented
end program test
```

Listing 1. Test program. The module definition is omitted for simplicity.

3 Results

The execution of the test does not show particular problems. The double precision complex matrix is printed in a human readable form on a text file in a limited precision format. This depends on the numerical format given in the write command inside the `zmat_print_file` subroutine when the `formatted` flag is enabled. Concerning the trace and adjoint operations, their correctness is tested by printing on terminal their results for a 5×5 matrix. A manual calculation confirms that they work fine.

4 Self-evaluation

The implementation of the code has led to successful results. The calculation of the determinant has not been implemented, so it can be the first future development of the code. For this purpose, an implementation of "LU" decomposition, mentioned in the theory Section, would be useful. Last but not least, I have done new progresses in learning Fortran programming language and writing new code in a cleaner and more efficient way is becoming easier.