Week 2: Derived Types in Fortran

Alice Pagano

(Dated: October 19, 2020)

A data structure is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data. One can use the data structures already provided by the programming language; however, to have more control of the code and to develop efficient software solutions, sometimes it is necessary to define new data structures. Fortran provides five intrinsic data structures (also called types): integer, real, complex, logical and character type. However, we can derive also our own data types as well. In this Report, we define a module which contains a matrix derived type both for real and complex number in double precision. We define functions which initialize and delete this new type. Moreover, we define the functions to compute the trace of the matrix and its adjoint. Finally, the module is tested with a simple demo program.

I. THEORY

A matrix is a grid of $n \times m$ (rows and columns) numbers surrounded by brackets. We can add and subtract matrices of the same size, multiply one matrix with another as long as the sizes are compatible ($(n \times m) \times (m \times p) = n \times p$), and multiply an entire matrix by a constant.

Let us consider a $n \times m$ matrix A with complex entries. The **conjugate transpose** $A^\dagger$ (or **adjoint**) is the $m \times n$ matrix obtained by taking the transpose of A and then taking the complex conjugate of each entry (the complex conjugate of $a + ib$ being $a - ib$, for real numbers a and b):

$$\boldsymbol{A}^\dagger = \overline{\boldsymbol{A}^\mathsf{T}} \tag{1}$$

For real matrices, the conjugate transpose is just the transpose, $\boldsymbol{A}^\dagger = \boldsymbol{A}^\mathsf{T}$.

Now, let us consider a square matrix $A \in (n \times n)$ with general entries. The **trace** of a square matrix is is defined to be the sum of elements on the main diagonal (from the upper left to the lower right) of A:

$$\mathrm{Tr}(\boldsymbol{A}) = \sum_{i=1}^{n} a_{ii} = a_{11} + a_{22} + \cdots + a_{nn} \tag{2}$$

Moreover, the **determinant** is a scalar value that can be computed from the elements of a square matrix and encodes certain properties of the linear transformation described by the matrix. It is denoted as $\det(\boldsymbol{A})$.

II. CODE DEVELOPMENT

In Fortran 90, we write a **MATRIX_TYPE** module (in the file "matrix_type.f90") which contains a matrix derived type. This Matrix type definition includes:

- the matrix dimension stored in a vector of two dimension, **N**;
- matrix elements (**ELEM**) of type **ELEM_TYPE**;
- matrix trace (**Tr**) of type **ELEM_TYPE**;
- matrix determinant (**Det**) of type **ELEM_TYPE**.

```fortran
TYPE Matrix
    INTEGER, dimension(2) :: N
    ELEM_TYPE, dimension(:,:), allocatable :: Elem
    ELEM_TYPE :: Det
    ELEM_TYPE :: Tr
END TYPE Matrix
```

The type **ELEM_TYPE** can be double real (**DM_ELEM_KIND**) or double complex (**ZM_ELEM_KIND**). In particular, it is chosen at the compilation stage by the pre-processing options and it is defined in the file "elem_datatype.F". Let us note that mixed types are not allowed in the program.

In the module are defined the following functions/subroutines:

- `SUBROUTINE` **`MatInit`**`(MatServ, N, ctrl)`: it takes as input a matrix (`MatServ`) and its dimension (`N`). The matrix is initialized depending on the value of the character `ctrl`:
  - if `ctrl` is equal to `'N'` the matrix is not initialized;
  - if `ctrl` is equal to `'Z'` the matrix is initialized with zeros;
  - if `ctrl` is equal to `'O'` the matrix is initialized with ones;
  - if `ctrl` is equal to `'R'` the matrix is initialized with random numbers between $[0, 1]$;

  Then, the matrix trace and the matrix determinant are set to zero;
- `SUBROUTINE` **`MatTrace`**`(MServ)`: it first checks if `MServ` is a square matrix and, in this case, this subroutine computes the matrix trace;
- `FUNCTION` **`MatAdjoint`**`(MServ)`: it computes the adjoint of the matrix `MServ`;
- `SUBROUTINE` **`MatWrite`**`(MatServ,MatName)`: it takes as input the matrix `MatServ` and (optionally) a string with its name (`MatName`). This subroutine writes in a text formatted file: the name of the matrix, size, elements, trace and determinant;
- `SUBROUTINE` **`MatDel`**`(MatServ)`: it deletes the matrix `MatServ`.

Moreover, the corresponding interface `Trace` and interface operator `.Adj.` are defined for **`MatTrace`** subroutine and **`MatAdjoint`** function.

This module is tested with a simple demo program ("demo.f90"). The main steps of this program are:

- the row and column of the matrix are given in input;
- a matrix A with size N is random initialized and its trace is computed;
- then we compute the adjoint of A obtaining the matrix B;
- we write both matrices in a file for comparison;
- we delete A and B.

```fortran
program demo
    USE MATRIX_TYPE
    ...
    CALL MatInit(A, N, ctrl='R')
    CALL Trace(A)
    B = .Adj.A
    CALL MatWrite(A,'A')
    CALL MatWrite(B,'A_adjoint')
    CALL MatDel(A)
    CALL MatDel(B)
end program demo
```

The entire code can be easily compiled with the use of a makefile. As said, at the compilation stage the desired **`ELEM_TYPE`** can be chosen by selecting the option **`D`** for double real and **`Z`** for double complex.

III. RESULTS

We run the demo program for different sizes of the matrix. The results obtained are corrected both for double real and double complex numbers. The text files printed by **`MatWrite`** subroutine looks well formatted and with all the information needed.

IV. SELF-EVALUATION

The matrix determinant is included in the matrix type definition, however it is not computed in this version of the code. It could be interesting implementing it using the LAPACK routines. Moreover, in a further development of the code, it would be useful implementing an error handling module to handle exceptions and a test module to test performances.