Quantum Information and Computing Assignment 2

Marco Chiloiro

14/11/2023

1. Checkpoints

Subroutine within the module ‘debugger’ to be used as a checkpoint for debugging.

```fortran
subroutine checkpoint(debug, msg, var_int, var_real)
!-----------------------------------------------------------------------------
! Description:
!     when debug is true, print the string msg and the variables var_int, var_real,
!     if present. When debug is false, nothing is printed.
!-----------------------------------------------------------------------------
```

```fortran
program checkpoint_test
    use debugger
    implicit none

    ! debug = true
    ! test message
    call checkpoint(debug=.true., msg='Test')
    ! test int variable
    call checkpoint(debug=.true., var_int=4)
    ! test real variable
    call checkpoint(debug=.true., var_real=1.5)
    ! test all the three
    call checkpoint(debug=.true., msg='Test all three', var_int=4, var_real=1.5)

    ! debug = false
    call checkpoint(debug=.false., msg='NO', var_int=0, var_real=0.)

end program checkpoint_test
```

```
(base) marco@marco-
 Test
           4
   1.50000000
 Test all three
           4
   1.50000000
(base) marco@marco-
```

2. Documentation

- Documentation
- Post-conditions
- Pre-conditions
- Error handling
- Checkpoints
- Comments

```fortran
subroutine MatrixMultiplication(n_min, n_max, step, verb)
    use debugger
    implicit none

!----------------------------------------------------------------------------------
! Description:
!   Matrix-matrix multiplication row-by-row, column-by-column and using MATMUL by
!   increasing the matrix size from n_min to n_max by step. Save the execution times
!   in three different txt files respectively.
!
! Pre-conditions:
!   n_min, n_max, step must be greater than 0.
!   n_min, step must be less than n_max
!
! Post-conditions:
!   the output txt files can be read as csv files with comma ',' as delimiter.
!----------------------------------------------------------------------------------

! Subroutine arguments
    ! watch 'Description'
    integer, INTENT(IN) :: n_min, n_max, step
    ! verbosity; if true, print the state of the execution
    logical, INTENT(IN) :: verb

! Local constants
    ! file names
    character(len=10) :: rbr = "rbr.txt", cbc = 'cbc.txt', MM = 'MM.txt'

! Local variables
    ! matrix dimension
    integer :: N
    ! matrices
    real, allocatable :: A(:,:), B(:,:), C(:,:)
    ! for loops
    integer :: ii, jj, kk
    ! for time measurements
    real :: start_time, end_time, elapsed_time

!----------------------------------------------------------------------------------

    ! check pre-conditions
    if (n_min <= 0 .or. n_max <= 0 .or. step <= 0) then
        call checkpoint(debug=.true., msg='Input variables must be positive integer.')
        stop
    end if
    if (n_min >= n_max) then
        call checkpoint(debug=.true., msg='n_max must be larger than n_min.')
        stop
    end if
    if (step >= n_max) then
        call checkpoint(debug=.true., msg='n_max must be larger than step.')
        stop
    end if

    ! executable
    call random_seed()
```

3. Derived types

Module containing a double complex matrix derived type, i.e. a matrix whose elements are complex*16 (both real and imaginary parts are real*8).

```
module mod_complex16_matrix

use debugger
implicit none

!-----------------------------------------------------------------------------
! Description:
!   complex matrix derived type for operating with matrices in double
!   precision.
!-----------------------------------------------------------------------------

type complex16_matrix
  ! Elements
  complex*16, dimension(:,:), allocatable :: elem
  ! Dimensions
  integer, dimension(2) :: size
end type

interface randInit
  module procedure :: cdmRandInit
end interface

interface zerosInit
module procedure :: cdmZerosInit
end interface

interface operator(.trace.)
  module procedure :: cdmTrace
end interface

interface operator(.Adj.)
  module procedure :: cdmAdj
end interface

interface to_txt
module procedure :: cdmToTxt
end interface
```

- **randInit**: initialize with random complex numbers, with both real and imaginary part values between 0 and 1.
- **zerosInit**: initialize with complex numbers with both real and imaginary part values equal 0.
- **.trace.:** given a complex double matrix, compute its trace.
- **.Adj.:** given a complex double matrix, returns its adjoint matrix.
- **to_txt:** write the given matrix on a .txt file.

3. Derived types - test

Test program which includes everything.

```fortran
program test
    use mod_complex16_matrix
    implicit none

    type(complex16_matrix) :: A, B, A_adj, B_adj
    complex*16 :: t_A, t_B

    ! initialize A randomly
    call randInit((/2,2/), A)
    ! initialize B as a matrix of 0s
    call zerosInit((/3,4/), B)
    ! set the diagonal elements of A to 1.+i*2.
    A%elem(1,1) = cmplx(1.d0, 2.d0, kind=8)
    A%elem(2,2) = cmplx(1.d0, 2.d0, kind=8)
    ! set the first element of B to 1.+i*1.
    B%elem(1,1) = cmplx(1.d0, 1.d0, kind=8)

    ! trace of A
    t_A = .trace.(A)
    print*, t_A

    ! Adoint of A
    A_adj = .Adj.(A)
    ! Adjoint of B
    B_adj = .Adj.(B)

    ! save A, B, A_adj and B_adj on different txt files
    call to_txt(A, 'A.txt')
    call to_txt(B, 'B.txt')
    call to_txt(A_adj, 'A_adj.txt')
    call to_txt(B_adj, 'B_adj.txt')

    ! trace of B (not square matrix, then we should expect an error)
    t_B = .trace.(B)
end program test
```

$$A = \begin{pmatrix} 1+2i & 1+2i \\ 1+2i & 1+2i \end{pmatrix} \qquad B = \begin{pmatrix} 1+i & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

```
Trace of A:          (2.0000000000000000,4.0000000000000000)
Matrix must be square.
```

3. Derived types - test

A.txt

```
0.10000E+01 + i*0.20000E+01,  0.69368E+00 + i*0.17812E+00
0.18039E+00 + i*0.80981E+00,  0.10000E+01 + i*0.20000E+01
```

A_adj.txt

```
0.10000E+01 - i*0.20000E+01,  0.18039E+00 - i*0.80981E+00
0.69368E+00 - i*0.17812E+00,  0.10000E+01 - i*0.20000E+01
```

B.txt

```
0.10000E+01 + i*0.10000E+01,  0.00000E+00 + i*0.00000E+00,  0.00000E+00 + i*0.00000E+00,  0.00000E+00 + i*0.00000E+00
0.00000E+00 + i*0.00000E+00,  0.00000E+00 + i*0.00000E+00,  0.00000E+00 + i*0.00000E+00,  0.00000E+00 + i*0.00000E+00
0.00000E+00 + i*0.00000E+00,  0.00000E+00 + i*0.00000E+00,  0.00000E+00 + i*0.00000E+00,  0.00000E+00 + i*0.00000E+00
```

B_adj.txt

```
0.10000E+01 - i*0.10000E+01,  0.00000E+00 + i*-.00000E+00,  0.00000E+00 + i*-.00000E+00
0.00000E+00 + i*-.00000E+00,  0.00000E+00 + i*-.00000E+00,  0.00000E+00 + i*-.00000E+00
0.00000E+00 + i*-.00000E+00,  0.00000E+00 + i*-.00000E+00,  0.00000E+00 + i*-.00000E+00
0.00000E+00 + i*-.00000E+00,  0.00000E+00 + i*-.00000E+00,  0.00000E+00 + i*-.00000E+00
```

NB! For simplicity, I used the format ‘e11.5’ to represent each real*8. It is possible to change the format as needed.