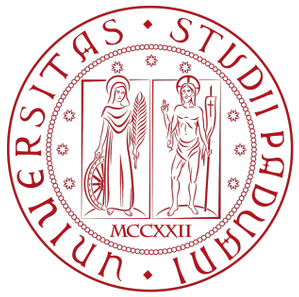


Assignment 2

Quantum Information and Computing

Giacomo Gasparotto

18 November 2025, A.Y. 2025/2026

Overview

A. Checkpoints:

- Manage errors
- Control the program flow
- Early stopping

B. Documentation and comments: everything is properly documented and commented to clearly explain the program's behavior

C. Derived types:

- Derived type to handle complex matrices
- *Subroutines*: allocate, deallocate and fill matrix
- *Functions*: Trace and Adjoint

Checkpoints

```fortran
subroutine checkpoint(debug, verbosity, message, A, B, nrows, ncols)
        use iso_fortran_env, only: real32
        implicit none

        ! Allocate variables
        logical, intent(in) :: debug
        integer, intent(in) :: verbosity
        character(len=*), intent(in), optional :: message
        real(real32), intent(in), optional :: A(:,:), B(:,:)
        integer, intent(in), optional :: nrows, ncols

        ! Early exit if debugging is disabled
        if (.not. debug) return

        select case(verbosity)
        case (0) ! Print a custom debug message
            if (present(message)) then
                print *, ""
                print *, trim(message)
                print *, ""
            end if

        case (1) ! Check matrix compatibility for multiplication
            if (present(A) .and. present(B)) then
                if (size(A,2) /= size(B,1)) then
                    print *, "ERROR: incompatible matrix dimensions!"
                    stop
                end if
            else
                print *, "WARNING: Missing A or B in verbosity level 1"
            end if

        case (2) ! Check if matrix is square
            if (present(nrows) .and. present(ncols)) then
                if (nrows /= ncols) then
                    if (present(message)) then
                        print *, "ERROR: nrows /= ncols — the ", trim(message), " is only defined
for square matrices."

                    else
                        print *, "ERROR: nrows /= ncols — matrix is not square."
                    end if
                    stop
                end if
            else
                print *, "WARNING: Missing nrows/ncols in verbosity level 2"
            end if

        case default
            print *, "ERROR: Invalid verbosity level. Choose 0, 1, or 2."
            stop
        end select
    end subroutine checkpoint
```

Checkpoint activation only if the ‘debug’ flag is .true.

Three verbosity levels:

0) Print an input message.

1) Check the dimension compatibility for matrix-matrix multiplication.

2) Check whether the matrix is square or not.

Documentation

Documentation example

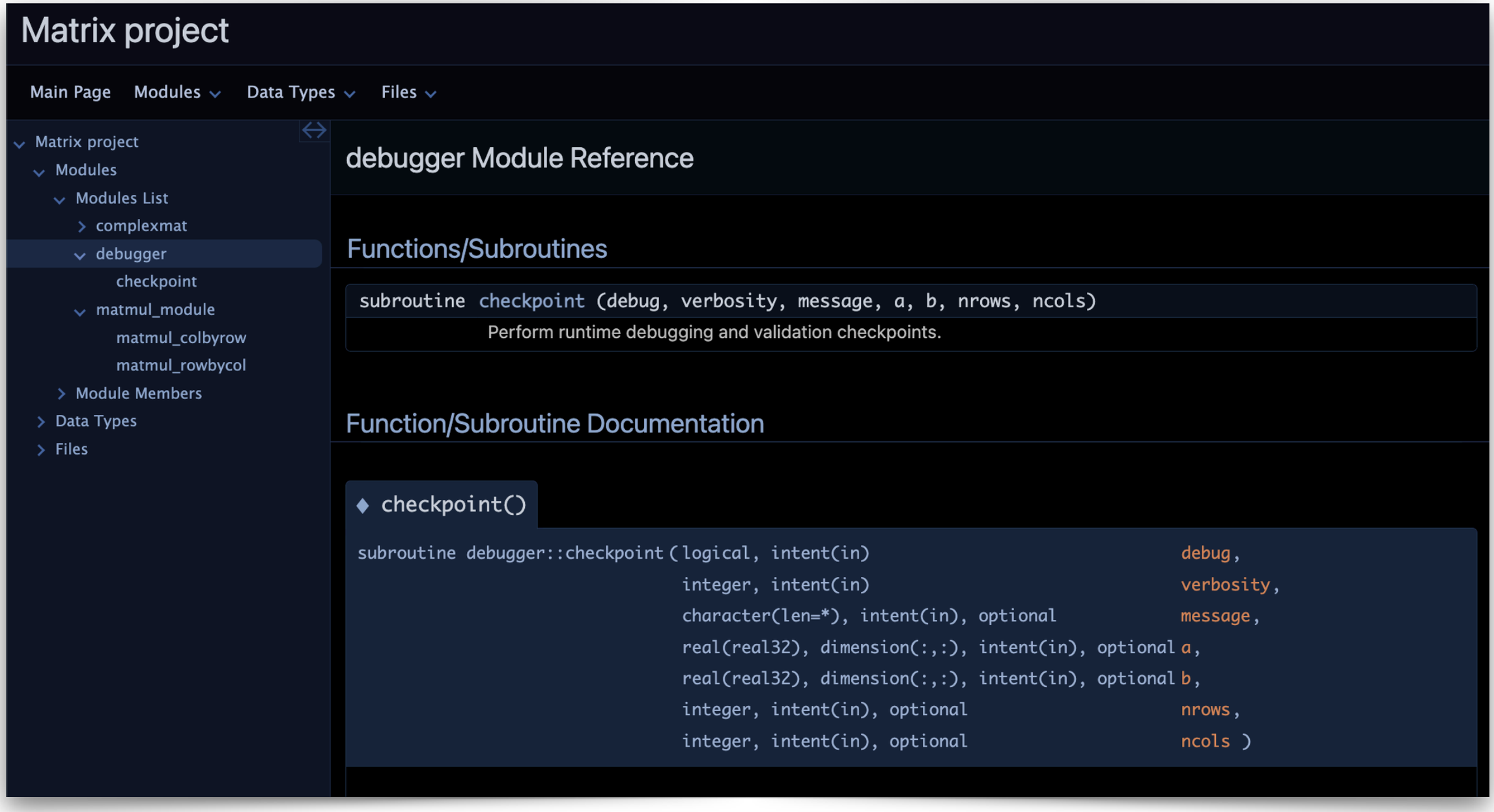
```
!> @file debugger.f90
!> @brief Debugging and error-checking utilities for matrix operations.
!>
!> This module provides the subroutine 'checkpoint', which is useful for
!> runtime debugging, error handling, and code flow control.
!> It supports different verbosity levels to perform optional checks
!> (e.g., printing messages, dimension compatibility, square-matrix
verification).

module debugger

    !*********************************************************************
    !> @brief Perform runtime debugging and validation checkpoints.
    !>
    !> The 'checkpoint' subroutine provides an interface for handling
    !> debugging messages and validity checks for matrices during execution.
    !> Depending on the selected verbosity level, it can:
    !>  - 0: print a debug message
    !>  - 1: check matrix dimension compatibility
    !>  - 2: verify that a matrix is square
    !>
    !> @param[in] debug      Logical flag to enable or disable debugging.
    !> @param[in] verbosity  Integer verbosity level:
    !>                       - **0**: print a debug message
    !>                       - **1**: check matrix dimension compatibility
    !>                       - **2**: verify if a matrix is square or not
    !> @param[in,optional] message  Custom message to print (verbosity 0 or 2).
    !> @param[in,optional] A(:,:)   First matrix to check (verbosity 1).
    !> @param[in,optional] B(:,:)   Second matrix to check (verbosity 1).
    !> @param[in,optional] nrows    Number of rows (verbosity 2).
    !> @param[in,optional] ncols    Number of columns (verbosity 2).
    !>
    !> @throws Stops execution if:
    !>         - matrix dimensions are incompatible (verbosity=1)
    !>         - matrix is not square (verbosity=2)
    !>         - an invalid verbosity level is provided
    !>
    !> @note The subroutine safely exits if 'debug' is '.false.'
    !*********************************************************************
```

Comments and checkpoints

```
! ====================================
!       COMPLEX MATRIX EXERCISE
! ====================================

! Initialise and fill matrix A
nrows = 5
ncols = 5
call initMatrix(A, nrows, ncols)
call fillRandomComplexMatrix(A)
print *, "Matrix A:"
call writeMatrix(A, "A.txt", .true.)

! Compute the adjoint of A
call checkpoint(.true., 2, “adjoint",
nrows=nrows, ncols=ncols)
call initMatrix(adj_A, nrows, ncols)
adj_A = .Adj.A
print *, "Adjoint of A:"
call writeMatrix(Adj_A, "adj_A.txt", .true.)

! Compute the trace of A
call checkpoint(.true., 2, "trace",
nrows=nrows, ncols=ncols)
tr_A = .Tr.A
print *, "Trace of A:", tr_A

! Deallocate matrices
call deallocateMatrix(A)
call deallocateMatrix(adj_A)
```

Documentation

The documentation style is compatible with Doxygen in order to create web page documentation.



Derived types

```
type complex8_matrix
        integer, dimension(2) :: size
        complex(real32), dimension(:,:), allocatable :: elem
    end type

    interface operator(.Tr.)
        module procedure Trace
    end interface

    interface operator(.Adj.)
        module procedure Adjoint
    end interface
```

Data Types

type	complex8_matrix
interface	operator(.tr.)
interface	operator(.adj.)

Functions/Subroutines

subroutine initmatrix (m, nrows, ncols)	Initialize a complex matrix of given dimensions.
subroutine fillrandomcomplexmatrix (m)	Fill a complex matrix with random complex numbers.
complex(real32) function trace (m)	Compute the trace of a complex matrix.
type(complex8_matrix) function adjoint (m)	Compute the adjoint (conjugate transpose) of a complex matrix.
subroutine writematrix (m, filename, do_print)	Write a complex matrix to a text file and optionally print it.
subroutine deallocatematrix (m)	Deallocate the memory of a complex matrix.

Constructor and fill subroutines

Compute trace and adjoint

Write matrix

Destructor subroutine