# Fortran Modernisation Workshop
# OOP Exercise

Alberto F. Martín*† Javier Principe ‡ †

July 26, 2017

The objective of this exercise is to illustrate the advantages of Object-Oriented Programming (OOP). This will be achieved by comparing the code development effort involved in the **extension of two versions** of a small software project that solves a symmetric positive definite (s.p.d.) linear system of equations $Ax = b$ using either direct or iterative methods (even though the potential of OOP for code reusability will be in particular made clear with the latter family of methods).

The first version of the software project is written using a procedural F90 style, while the second, a F200X OOP programming style. Both of them are grounded on a refactorization of a legacy code[1], i.e., we already took the effort to modernise it following the guidelines provided during the first two days of the workshop.

The **extension to be performed** on both versions of the code is as follows. When the structure of the linear system coefficient matrix $A$ is **banded and symmetric** (i.e., only a number of diagonals of the matrix contain nonzero entries and $a_{ij} = a_{ji}$, respectively), a significant amount (roughly half) of the required memory can be saved by only storing either the lower or upper triangle of the matrix. This requires a specific data structure and a specific implementation of the algorithms operating on it, and thus, an extension of the software project that you have to develop.

This document itself, the codes required to perform the exercise, a set of accompanying CMake scripts for autocompilation, and bash shell scripts for execution, are available at a public Git repository. This repository has to be cloned on your `$HOME` directory, and accessed as:

```
$ cd $HOME
$ git clone https://gitlab.com/femparadmin/fmw_oop_bcn_material.git
$ cd fmw_oop_bcn_material
```

The structure of this repository is illustrated in Figure 1. The source code is available at the `exercises` folder. In particular, the `exercise1` subfolder contains the source

*Department of Civil and Environmental Engineering. Universitat Politècnica de Catalunya, Jordi Girona 1-3, Edifici C1, 08034, Barcelona, Spain

†CIMNE Centre Internacional de Mètodes Numèrics en Enginyeria, Parc Mediterrani de la Tecnologia, UPC, Esteve Terradas 5, 08860 Castelldefels, Spain.

‡Departament de Mecnica de Fluids, Escola d'Enginyeria de Barcelona Est (EEBE), Universitat Politècnica de Catalunya, C. Eduard Maristany, 10-14, 08019, Barcelona, Spain

[1]Available at `https://people.sc.fsu.edu/~jburkardt/f_src/wathen/wathen.html`.

code of the procedural F90 version of the code, while `exercise2` contains the one of the OOP Fortran200X version. The main programs that you ultimately have to execute are available at `exercise1.f90` and `exercise2.f90` source codes, respectively. These are the right starting point to get familiarized with both versions of the software project.

The code of either version is compiled using the following commands, with `X = 1,2`:

```
$ cd $HOME/fmw_oop_bcn_material/exercises
$ mkdir build_exerciseX
$ cd build_exerciseX
$ cmake ../exerciseX
$ make
```

Right after compiling the source code, the compiled executable program is available at `build_exerciseX/bin/exerciseX`. You do not have to execute it directly though. You have to do it through the shell bash scripts named `run_exerciseX_tests.sh` as follows:

```
$ cd $HOME/fmw_oop_bcn_material/exercises
$ ./run_exerciseX_tests.sh
```

These shell scripts run the executable program passing different combinations of values for its command-line parameters. In particular, the same linear system of equations is solved using three storage formats for the coefficient matrix $A$ (full, banded, and sparse), and two different solvers (direct Gaussian elimination and iterative Conjugate Gradients-CG solvers). The solution error for each combination is shown on the screen. The solution should be on the order of $10^{-14}$ accurate if everything succeeded.

The exercises to be performed are as follows:

1. Implement the symmetric (upper triangular) version of the band matrix storage case in the procedural F90 version of the program (source code available at `exercise1` subfolder). The `direct.f90` source code contains subroutines to perform direct Gaussian elimination of full and banded storage matrices, while `iterative.f90`, three different implementations of the CG solver (and accompanying matrix-vector product subroutines) for full, banded, and sparse matrix storage; see Figure 1.

2. Implement the symmetric (upper triangular) version of the band matrix storage case in the OOP F2003 version of the program (source code available at `exercise2` subfolder). In order to to so, add a new module `symmetric_band_matrix_mod` to the software project that contains a data type `symmetric_band_matrix_t` as a new type extension of the `matrix_t` abstract data type (available at the `matrix.f90` source code; see Figure 1). You MUST thus implement the deferred TBPs of the latter in the former.

In order to develop both exercises, we supply the subroutines operating on symmetric upper triangular banded matrices at the `symmetric_band_matrix_subroutines.f90` source file; see Figure 1.

```
.
|-- examples
|   ... Not relevant for the exercise,
|   ... just for the course slides
|-- exercises
|   |-- exercise1
|   |   |-- CMakeLists.txt
|   |   |-- direct.f90
|   |   |-- exercise1.f90
|   |   |-- iterative.f90
|   |   |-- mcheck.i90
|   |   |-- random.f90
|   |   |-- types.f90
|   |   `-- wathen.f90
|   |-- exercise2
|   |   |-- band_matrix.f90
|   |   |-- blas.f90
|   |   |-- CMakeLists.txt
|   |   |-- dense_matrix.f90
|   |   |-- direct.f90
|   |   |-- exercise2.f90
|   |   |-- iterative.f90
|   |   |-- matrix.f90
|   |   |-- matrix_factory.f90
|   |   |-- mcheck.i90
|   |   |-- random.f90
|   |   |-- solver.f90
|   |   |-- solver_factory.f90
|   |   |-- sparse_matrix.f90
|   |   |-- types.f90
|   |   `-- wathen.f90
|   |-- run_exercise1_tests.sh
|   |-- run_exercise2_tests.sh
|   `-- symmetric_band_matrix_subroutines
|       `-- symmetric_band_matrix_subroutines.f90
```

Figure 1: Structure of Git repository.

How and how much did you have to change the code in either case (you can, e.g., execute `git diff` on the root of the Git repository to observe this)? What did you have to do in either case to have a Conjugate Gradient (CG) iterative solver able to deal with the new matrix data structure ?