DCSC Coding Guidelines

Ivo Grondman, Lucian Buşoniu, Gabriel Lopes

When producing code for a project or thesis – whether this is in C, MATLAB, or any other programming language – you will have to ensure that it is readable and understandable by others who may or may not have not been involved with the project or thesis. For students working on their BSc/MSc thesis: chances are that future students will build extensions on top of your code, which makes it even more important to produce understandable code. The following section describes a few strict guidelines that you will have to comply with when writing code. This might seem like an extra burden, but you will find that using these guidelines will also help you in developing, maintaining and reusing your own code.

The following rules have to be taken into account when writing code:

1. **Comment and document your code**
   Strive to make your code as self-explanatory as possible. If a line or a block of code is not self-explanatory, provide comments above that line or block. No single line in a piece of code should ever puzzle someone.

2. **Strict compliance between code and thesis / other documents**
   Variables should have the same name in the code and thesis, or the names should be as similar as possible. For example, the variable α_C in a thesis could be `alpha_C` in your code. When you have pseudo-code in your thesis, the order of operations there should be the same as the order of operations in the code.

3. **No hard-coded numbers in formulas or dependencies**
   Formulas and variable dependencies should only be expressed as operations on variables which are declared/initialized elsewhere, *not* on hard-coded numbers. This will make it much easier to adapt the code to other parameter values, etc. As a simple example, for Newton's second law of motion, you would have:

```
m = 10;          % mass
a = 5;           % acceleration
F = m*a;         % force
```

   instead of

```
m = 10;          % mass
a = 5;           % acceleration
F = 50;          % force
```

4. **Create modular code**
   Separate your code into building blocks that can be re-used. For example: if the same lines appear at several places in your code, e.g. some sort of algorithm, you should create a separate function or file out of those lines. If argument lists of such functions get too long, use structures as input variables. This approach is also applied in the example provided.

5. **Separate computation from presentation**
   This is in addition to the previous point. Make sure you have one set of functions for simulations or other procedures that generate data and a seperate set of functions for the presentation/visualization of the generated data. In MATLAB, this is easy to do by storing the generated data from a simulation in `MAT`-files and make the functions that do the visualization read from those files. This will also avoid having to run simulations over and over to reproduce plots. An example you can build upon is provided.