Getting Started

It seems like computers are here to stay. They certainly represent a powerful tool for analysing the performance of structures. Just wait until you create your own computer program and see it run: you will find it empowering. Instead of developing pre-packaged computer programs we should ideally develop libraries of algorithms that can be shared, understood, improved, and used in ad hoc programs. It is perhaps fun to observe how algorithms play an increasingly large role in our modern society. Elon Musk suggests that artificial intelligence algorithms could become a danger to the public. Yuval Noah Hari discusses the role of algorithms, including the ones in your brain, in the book Homo Deus. The joke that the word "algorithm" stems from Al Gore is funny but of course not true, although he promoted the public use of ARPANET, which is now the Internet. Rather, the word algorithm stems from the Persian mathematician Muhammad bin Musa al-Khwarizmi who was born in 780 and worked at the House of Wisdom in Baghdad. Euclid's Algorithm, probably the earliest known algorithm, is from before that time and attributed to Euclid, born in 335 BC. He devised an algorithm, i.e., systematic approach, for determining the greatest common divisor of two integers. This document is NOT about running algorithms in pre-existing computer programs, such as S-FRAME, ETABS, SAP2000, ABAQUS, or ANSYS. Rather, the objective here is to help you create and run your own code. Several programming languages are available for this purpose. The following are examples of "compiled languages," in which you first write the code, then compile it into an executable file that you subsequently run:

- C++ (used in this document together with Python)
- Fortran
- Java

Alternatively, you may wish to use "scripting languages," in which you never see the compilation step. The code runs when you press "Run." These are easier to work with, but the code runs slower. Common scripting languages are:

- Python (used in this document together with C++)
- Matlab
- Javascript

Another aspect that separates programming languages is static and dynamic type checking. With dynamic type checking, which is common in scripting languages, you need not declare the type of a number before you use it. In contrast, with compiled languages you usually cannot say a=2 without first declaring what a is. For example, in C++ you must write:

```
double a=2;
double b=3.1;
double c=a+b;
```

or alternatively start with the statement `double a,b,c;` to declare the type of a, b, and c. Type conversion is possible, such as casting an integer into a double to facilitate a computation:

```
double a = 2.0;
int    b = 2;
double c = a+(double)b;
```

Conversely, in Python you can directly say:

```
a=2
b=3.1
c=a+b
```

Before starting programming, take a moment to commit to tidiness and clarity. Looking at computer code is like looking at someone's signature. Everyone has a different style, and some variation in tidiness is unavoidable. However, do not write code without blank lines to provide air and comment-lines to explain what is going on. In C++ comment lines can be inserted anywhere with two preceding forward-slashes, and these lines are simply ignored by the compiler:

```
// This is a comment line in C++
```

Sometimes it is useful to "comment out" many lines. To accomplish this, the text is encapsulated by /* and */ in the following manner:

```
/* These symbols encapsulate several
comment lines in C++ */
```

In Python a comment line starts with the pound sign:

```
# This is a comment line in Python
```

To comment out a block of code in Python the text is encapsulated by three single quotation marks:

```
''' These symbols encapsulate several
comment lines in Python '''
```

Python

There are several ways to get things done with Python. You can run it as a calculator by entering statements directly in a command prompt window, you can run input files that you prepare in a stand-alone text editor, or you can work in an integrated developer environment (IDE). The last approach is preferred and many IDEs are available to help you edit and execute Python code. Examples include PyCharm, which is similar to a Matlab editor, and Jupyter Notebooks in which you can sequentially write blocks of code and text. Python is particularly useful in engineering applications when certain external packages are added to it. These packages contain functions that let us work with matrices, create plots, etc. The following packages are particularly popular:

- NumPy for vectors, matrices, linear algebra, etc.
- SciPy for probability distributions, etc.
- Matplotlib for plotting

To get started with Python, using an IDE called PyCharm, together with those packages, follow these steps:

1. Install the latest version of Python3 from https://www.python.org
2. Install the IDE called PyCharm from https://www.jetbrains.com/pycharm
3. In PyCharm, select File > New Project and give it a file path and a name
4. In PyCharm, select Preferences to open the settings dialog box for the project
5. On the left-hand side of that dialog box, select the Project: (name) tab and specifically the Project Interpreter
6. Now on the right-hand side, make sure that some version of Python3 is selected as Project Interpreter
7. In the same dialog box, press + near the bottom to install NumPy, SciPy, and Matplotlib, and then close the dialog box
8. In PyCharm, right-click on the project at the top of the left-most pane and select New > Python file and give it a name
9. Enter some Python code into that .py file, for example `print("Hello World!")` and right-click anywhere in the file to find the option to run it.

Please note that many of the Python functions posted on this website require libraries to be imported at the top of the file in which you store the functions:

```
import numpy as np
from scipy.stats import norm, lognorm, uniform
import matplotlib.pyplot as plt
```

C++

The first step towards creating programs with C++ is to install a compiler. Xcode is the default option on Mac, while Visual Studio and MinGW are popular options on the Windows operating system. C++ code is essentially a collection of statements terminated by a semicolon, such as:

```
a = b + c;
```

Most C++ code is organized into "header files" (.h) and "code files" (.cpp). A header file essentially shows a table of contents of the code that is written in the corresponding code file. Typically at the top of these files, but sometimes elsewhere, are compiler directives that start with the pound symbol, such as the following include-statements:

```
#include <stdio.h>
#include "mycode.h"
```

The content of the files that are included in that way are read by the compiler. Angle brackets cause the compiler to search for the file in folders specified by include-type environment variables. Conversely, double quotation marks mean that the compiler first searches the directory where the file with the include statement is located. The double quotation marks can also be used to specify the exact path to the included file.