A Comparison of C, MATLAB and Python as Teaching Languages in Engineering

Hans Fangohr

University of Southampton, Southampton SO17 1BJ, UK, fangohr@soton.ac.uk

Abstract. We describe and compare the programming languages C, MATLAB and Python as teaching languages for engineering students. We distinguish between two distinct phases in the process of converting a given problem into a computer program that can provide a solution: (i) finding an algorithmic solution and (ii) implementing this in a particular programming language. It is argued that it is most important for the understanding of the students to perform the first step whereas the actual implementation in a programming language is of secondary importance for the learning of problem-solving techniques. We therefore suggest to chose a well-structured teaching language that provides a clear and intuitive syntax and allows students to quickly express their algorithms. In our experience in engineering computing we find that MATLAB is much better suited than C for this task but the best choice in terms of clarity and functionality of the language is provided by Python.

1 Introduction

Computers are increasingly used for a variety of purposes in engineering and science including control, data analysis, simulations and design optimisation. It is therefore becoming more important for engineering students to have a robust understanding of computing and to learn how to program.

In this paper, we outline the difficulties in learning and teaching programming in an academic context including the choice of the programming language. In section 2, we suggest a distinction between the algorithmic problem-solving part of computer programming and the efforts to implement the algorithm using a particular programming language. In section 3, we describe and compare MATLAB, C and Python as potential teaching languages and report our experience of them in an Engineering Department in section 4 before we conclude.

2 Teaching objectives

We understand the subject of "computing" to broadly represent the usage of computers and numerical methods to solve scientific and engineering problems. In the curriculum, we aim to go beyond the usage of dedicated software packages and to enable students to write their own computer programs to provide insight into the functionality of any software (at least in principle) that they may encounter. In this section, we derive our requirements for programming languages to be used in education.

Universal programming building blocks: To analyse the programming process, we list the main ingredients for any computer program (this can be done more formally but is sufficient for our purposes here): (i) statements that do something (for example adding two numbers, perform a Fourier transform, read a sensor), (ii) blocks of statements, (iii) loops that repeat blocks (for-loops, for-each-loops, while-loops, repeat until loops), (iv) conditional execution of blocks (if-then statements, case, switch) and (v) grouping of blocks into modules (functions, procedures, methods).

This lists a set of commands that is sufficient to describe sequential algorithms, and by grouping statements into modules, moderately large and well-structured programs can be written within this framework.[1] Virtually all programming languages provide constructs that correspond to the listed items.

Problem-solving process: Computer programs are generally used to solve a given problem. We divide the process of writing a computer program into two parts:

1. finding the algorithmic solution (we will call this the "algorithmic problem-solving part") and
2. implementing the algorithm in a particular language (the "implementation part").

For example, to compute an approximation A of the integral $I = \int_a^b f(x)\mathrm{d}x$ using the composite trapezoidal rule with n subdivisions of the interval $[a, b]$: $A = \frac{h}{2}\left(f(a) + f(b) + 2\sum_{i=1}^{n-1} f(x_i)\right)$ with $h = \frac{b-a}{n}$ and $x_i = a + ih$, the two parts of the solution are:

1. the algorithmic solution which can be written in some form of pseudo-code:

```
user provides f, a, b, n
compute interval width h=(b-a)/n
set initial area=0.5*h*(f(a)+f(b))
for each point x=x_i with i increasing from 1 to n-1
    compute area under f(x)
        x = a + i*h          (this is the current x)
        dA = f(x)*h          (this is the new contribution)
    area = area + dA         (update the total area)
return area to user
```

2. the implementation which expresses the algorithmic solution in some programming language. For example figure 1 shows MATLAB code[2] that performs the calculation (for $f(x) = \exp(-x^2)$ and $a = 0$ and $b = 1$).

While in this example the algorithmic problem-solving is relatively straight forward (because the problem was posed in form of an equation), in general this

[1] We recognise the importance of object orientation but have excluded such features from the list above for clarity of argument.

[2] There are more efficient, general and elegant ways to code this but these are unlikely to be used by beginners and irrelevant to the situation described here.

```
% input from user
a = 0.0;    b = 1.0;    n = 100;

h = (b-a)/n;
area = 0.5*h*( exp(-a^2) + exp(-b^2) );

for i=1:n-1
   x = a+i*h;
   area = area + h*exp(-x^2);
end

fprintf('The value of the approximation is \%f\n', area)
```

Fig. 1. A MATLAB program to approximate $\int_0^1 \exp(-x^2)\mathrm{d}x$.

is the main challenge the students face: the conversion of a problem described vaguely and informally in natural language into a sequence of instructions that break the problem in many small parts that a computer can solve subsequently. The implementation part can be complicated and time consuming but does not (or at least should not) contain major intellectual challenges.

The boundary between the algorithmic problem-solving and the implementation is, of course, not clearly defined. However, it is clear that different implementations of a problem-solving algorithm in different languages share the same underlying algorithm (which we use here as the definition for the algorithmic problem solving part). In the teaching practice, the algorithmic problem-solving and implementation tasks are often entangled simply because the students need to test an algorithm they invented by implementing it.

Teaching objectives in computing: We argue that the primary target in teaching computing is to enable the students to convert engineering problems into pseudo-code. This is a challenging task that requires analytical thinking and creativity. The conversion of this pseudo-code into a program written in one programming language is of secondary importance because it is, in principle, an algorithmic procedure and requires less intellectual effort.

Consequently, the choice of the teaching language should be governed by which language provides the best support to the student in performing the implementation part of the problem-solving task. (The remainder of this paper addresses this question.) Once students are confident in the algorithmic problem-solving part, they can learn new programming languages as required to port the algorithmic solutions to their current working environment.

3 Overview of programming languages used

The C programming language: The C programming language [1] is a low-level compiled language (sometimes classified as a 3^{rd} generation language) that is widely used in academia, industry and commerce. Fortran falls into the same category but while Fortran is still commonly used in academia it appears to be overtaken by C (and C++) in many industrial applications. C++ provides a different programming paradigm than C but for the purpose of this work, C++

is more similar to C than it is to MATLAB or Python. The main advantage of compiled low-level languages is their execution speed and efficiency (for example in embedded systems).

MATLAB: The MATLAB programming language is part of the commercial MATLAB software [2] that is often employed in research and industry and is an example of a high-level "scripting" or "4$^{\text{th}}$ generation" language.

The most striking difference to C and other compiled languages is that the code is interpreted when the program is executed (an interpreter program reads the source code line by line and translates it into machine instructions on the fly), *i.e.* no compilation is required. While this decreases the execution speed, it frees the programmer from memory management, allows dynamic typing and interactive sessions. It is worth mentioning that programs written in scripting languages are usually significantly shorter [3] than equivalent programs written in compiled languages and also take significantly less time to code and debug. In short, there is a trade-off between the execution time (small for compiled languages) and the development time (small for interpreted languages).

An important feature for teaching purposes is the ability of MATLAB (and other interpreted languages) to have interactive sessions. The user can type one or several commands at the command prompt and after pressing return, these commands are executed immediately. This allows interactive testing of small parts of the code (without any delay stemming from compilation) and encourages experimentation. Using the interactive prompt, interpreted languages also tend to be easier to debug than compiled executables.

The MATLAB package comes with sophisticated libraries for matrix operations, general numeric methods and plotting of data. Universities may have to acquire licences and this may cost tens of thousands of pounds.

Python: Python [4] is another high-level language and at first sight very similar to MATLAB: it is interpreted, has an interactive prompt, allows dynamic typing and provides automatic memory management (and comes with in-built complex numbers).

We have included Python in this work because it provides several advantages over MATLAB in the context of teaching: (i) Python has a very clear, unambiguous and intuitive syntax and uses indentation to group blocks of statements. (ii) Python has a small core of commands which provide nearly all the functionality beginners will require. (iii) Python can be used as a fully object-orientated language and supports different styles of coding. (iv) The Python interpreter is free software (*i.e.* readily available), and Python interpreters for virtually all platforms exist (including Windows, Linux/Unix, Mac OS).

It is worth noting that although Python has been around for only approximately 10 years, it is a relatively stable language and used increasingly in industry and academia (currently including organisations such as Philips, Google, NASA, US Navy and Disney). It also provides the framework for creating and managing large modularised codes. Commonly used extension modules provide access to compiled libraries including high performance computation [5] and visualisation tools.

```
#include<stdio.h>
#include<math.h>

int main( void ) {
  int n = 100;   double a = 0.0;   double b = 1.0;

  int i;
  double h = (b - a) / (double) n;
  double area = h*0.5*( exp( -a*a ) + exp( -b*b ) );
  double x;

  for (i=1; i<n; i++) {
    x = a + i*h;
    area = area + h*exp( -x*x );
  }

  printf("The value of the approximation is %f\n", area);

  return 0;
}
```

Fig. 2. C program to approximate $\int_0^1 \exp(-x^2)\mathrm{d}x$.

4 Teaching Experience

In this section, we present for each of the languages in question a short program that performs the numerical integration as given in the equation in section 2. The source codes shown are neither commented, optimised for speed and elegance nor do they explore advanced features of the respective language. This is to save space and to represent the coding style that students would initially use. We then report and discuss experiences in teaching the three languages to first and second year undergraduate engineering students and to postgraduate students from different backgrounds.

Using C: Figure 2 shows one C program that performs the same computation as the MATLAB program shown in figure 1. In comparison to the MATLAB program, this code is longer and carries a substantial overhead (such as the include statements, the wrapping of the main code into the `main` function and the return of the exit status). It is necessary to declare variables and their types before any statements are executed. Eventually, the student needs to compile the code (and link to the mathematics library) before it can be executed.

Typical problems students experience while programming in C are: (i) Indentation of for-loop (and other blocks) and scope (as defined by curly braces) do not agree, thus the for-loop executes incorrect commands. (ii) Missing semicolons, curly braces, parentheses around if-statement tests in combination with moderately useful error messages from the compiler stop the compilation process. (iii) Passing of values of wrong type in function calls or printing numbers using the wrong format identifier token; both problems result in wrong numerical results that students find difficult to interpret and rectify.

While these are not particularly challenging to the experienced programmer it can be observed that beginners struggle when they come across this framework,

```
import math
n = 100;    a = 0.0;     b = 1.0

h = (b-a)/n
area = 0.5*h*( math.exp(-a**2) + math.exp(-b**2) )

for i in range(1,n):
    x = a+i*h
    area = area + h*math.exp(-x**2)

print "The value of the approximation is", area
```

Fig. 3. Python program to approximate $\int_0^1 \exp(-x^2)\mathrm{d}x$.

whereas there are significantly less difficulties when they start using MATLAB or Python.

Using MATLAB: Figure 1 shows the example program. We generally find that it is much easier for students to start programming using MATLAB than it is using C because MATLAB addresses many of the issues raised above.

It is good practice to split code into small functional units where-ever possible to modularise programs and to be able to re-use the functions individually for different projects. One of the recurring problems that students experience in learning programming using MATLAB is the convention of storing only one (globally visible) function in a file. This can result in a large number of files and initially the usage of functions is experienced by students as being counterproductive: several files have to be displayed to see all the source code at the same time (thus making it hard to follow the programme flow on the screen). A related issue is that in MATLAB the name of the function as specified within the source file should be identical to the file name of the source file containing that function. (The filename determines the globally visible name of the function.) This is often overlooked (by the students) and a source for many errors.

Using Python: Figure 3 shows a Python implementation of the integration problem. By importing the `math`ematics module in the first line, the use of name spaces is nicely demonstrated.[3]

The `range` command in line 7 returns a list of integers (ranging from 1 up to but not including `n`) which `i` refers to in subsequent iterations. The for-loop used here is actually a for-*each* loop because for each element in the list of integers, the body of the loop is executed. For-each-loops are in general more powerful than the for-loop, although this is not exploited here. (Note that the for-loop in MATLAB is also a for-each loop although the syntax makes this less obvious.)

It is of significant benefit in teaching programming that the block of statements in the body of the for-each loop is limited solely by indentation because much time is spent encouraging students to ensure that the actual block delimiters (curly braces {} in C, the `for` and `end` keywords in MATLAB) are in

[3] This also addresses issues with unintented use of global variables in MATLAB which can confuse beginners but which are outside the scope of this report.

```
import math

def integrate( f, a, b, n):
    h = (b-a)/n
    sum = (f(a) + f(b))*0.5

    for i in range(1,n-1):
        x = a+i*h
        sum = sum + f( x )
    return sum*h

def f1(x):
    return math.exp(-x**2)

def f2(x):
    return math.sin(-x**4)

# main program starts here:
n = 100; a = 0.0; b = 1.0
print "The approximation of f1 is ", integrate( f1, a, b, n)
print "The approximation of f2 is ", integrate( f2, a, b, n)
```

Fig. 4. A Python program demonstrating how to pass functions as arguments.

agreement with the chosen indentation (because our perception of the program structure is led by indentation). Any disagreement between the two is likely to represent an error.

Python offers an environment which addresses most of the problems we observe in teaching C and MATLAB. Students experience comparatively few problems in starting to program in Python and develop to enjoy and experiment within the intuitive environment.

Advanced example: It is outside the scope of this work to explain many of the reasons why Python is generally appealing as a first programming language (see for example [6]). Instead, we present the implementation of one logical extension of the integration programmes in Python and compare this with C and MATLAB.

Students should experience that it is advantageous to build a library of small generic functions that can be used and re-used to solve bigger problems rather than to write specific code to solve one, and only one, problem. As part of this, it is sensible to code the integration routine in a function which takes the following parameters: the function to integrate (`f`), the lower and upper integration limits (`a` and `b`) and the number of subdivisions (`n`). A Python program that first defines this function with the name `integrate` and then integrates two functions `f1` and `f2` is shown in figure 4.

Note that the function to be integrated is passed to the `integrate` function just like any other object, and that `f` is evaluated inside the `integrate` function as if a function `f` would be defined (whereas in fact `f` refers to either `f1` or `f2`). This is in stark contrast to C where the passing of functions as arguments needs the introduction of pointers (and where the type of theses pointers take the argument list of the function they point to into account), and where the pointer has to be de-referenced to call the function. In MATLAB, the situation is not as complicated as in C but the function `integrate` would have to be rewritten to use the `feval` command to evaluate the function rather than to evaluate it intuitively.

The technical reason for the ease with which functions can intuitively be passed and used in Python, is that everything in Python (including numbers, functions and classes) are objects. Note that it would be easy to replace the function calls to the `integrate` function in the last lines of figure 4 by this for-each loop over the list of functions `f1` and `f2`:

```
for f in [f1,f2]:
    print "The approximation is", integrate( f, a, b, n)
```

Each object stores its name in an attribute `__name__`, so the above code can be modified to print not only the value of the integral of `f1` and `f2` respectively, but also the name `f1` and `f2`:

```
for f in [f1,f2]:
    print "The approximation of",f.__name__,"is",integrate( f, a, b, n)
```

A corresponding solution in MATLAB is far less elegant, and it requires significant effort (which we believe is outside the scope of first year students) to achieve this in C.

5 Conclusions

In comparing the three different languages (C, MATLAB and Python) as outlined in section 4, we find that the Python language appears most intuitive to the students followed by MATLAB, and C. We believe that Python is an excellent language to begin computer programming with and which allows the student to focus on the problem-solving rather than on the syntax and structure of the programming language.

The author thanks A. J. Chipperfield, R. P. Boardman, M. Molinari and J. M. Generowicz for useful discussions.

References

1. Kernighan, B.W., Ritchie, D.M.: The C Programming Language. Prentice Hall Software Series (1988)
2. The Mathworks: Matlab (2003) `www.mathworks.com`.
3. Prechelt, L.: An empirical comparison of seven programming languages. IEEE Computer **33** (2000) 23–29
4. van Rossum, G.: Python tutorial. Centrum voor Wiskunde en Informatica (CWI), Amsterdam. (1995) `www.python.org`.
5. NumPy Team: Numerical Python (2003) `www.pfdubois.com/numpy`.
6. Donaldson, T.: Python as a first programming language for everyone. In: Western Canadian Conference on Computing Education. (2003) `www.cs.ubc.ca/wccce/Program03/papers/Toby.html`.