Embedded Systems Programming Lab Report

Rhydian Windsor

October 2018

Abstract

In this report, the speed of C++ in solving a set of exactly determined linear equations is compared with three other programming paradigms; using standard Python, Python using the numpy module and MATLAB.

1 Introduction

Solving systems of linear equations is important as it provides the basis for most linear algebra. In particular it can be used to determine eigenvectors. Furthermore, we can gain insight into non-linear systems using linear algorithms by careful approximation. It has been roughly estimated that around 75% of all scientific computing problems require solving linear equations.

This report focuses on *exactly determined* linear systems, that is a set of N equations, each with N unknowns. These are used because they have unique solutions, provided they are both non-degenerate and consistent.

2 Method

This section focuses on the method used to implement a linear system equation solver in C++. The methods used for the other paradigms can be read about in their documentation or seen in the source code for this practical.

We are interested in solving linear equations of the form

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \tag{1}$$

The method used to achieve this is Gaussian elimination, where by we put our matrix into *augmented form*

$$\left(\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{array}\right) \tag{2}$$

which can the be converted into *row echelon* form by subtracting multiples of different rows from each other;

$$\left(\begin{array}{cccc|c} a'_{11} & a'_{12} & \cdots & a'_{1n} & b'_1 \\ 0 & a'_{22} & \cdots & a'_{2n} & b'_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & a'_{nn} & b'_n \end{array}\right). \tag{3}$$

This augmented matrix can then be converted back into standard linear system form and can by easily solved by first determining x_n from the bottom row and continuing up the rows.

In C++, the algorithm was operated on a `Matrix` class, defined and implemented in `matrix_class.h` and `matrix_class.cpp` respectively. It contains three variables; `num_rows` indicating the number of rows, `num_columns` indicating the number of columns, and `data`, an element of type `vector<vector<double>>` containing the elements of each row. This was done using dynamically assigned vectors rather than statically to allow extensions of the class whereby matrices can be reshaped and appended to.

Linear systems of equations are solved using the function named `solve_linear_equation`. This function takes an augmented `Matrix` of the form given in Equation 2. It should be noted that this function only accepts matrices of form $\mathbb{R}^N \bigotimes^{N+1}$ where N is any positive integer as we are only interested in solving exactly determined systems.

By looping over elements of the array this is converted into row echelon form and the corresponding linear system is then solved by considering the final row first and then moving up one row at a time.

3 Results

The linear equation solver was compared with several other programming paradigms, specifically python, python with numpy and MATLAB. A matrix, $M \in \mathbb{Z}^N \bigotimes^{N+1}$ was generated for $\{N \in \mathbb{Z} | N > 0, N < 100\}$. Each element of this matrix was initialised random integer from 0 to 99. This was then treated as an augmented matrix for a linear system of equations and solved. The time taken for this system to be solved was recorded and is shown in Figure 1.

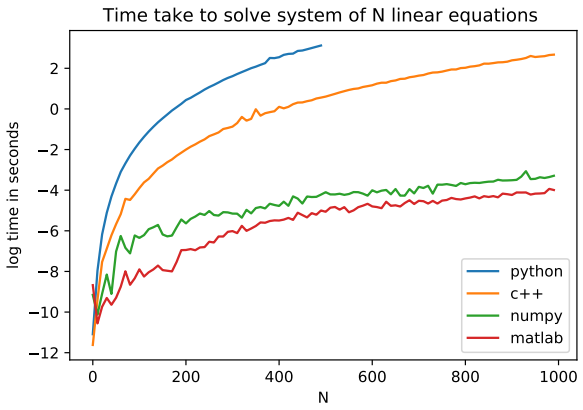

Figure 1: A comparison of speeds of the different paradigms at solving a system of N linear equations. The python calculation was terminated early due to time constraints.

4 Conclusion

From Figure 1 we can see that Gaussian elimination in C++ is considerably faster than in pure python. This is unsurprising as C++ is an example of a compiled language rather than interpretive. As the entire program is compiled into an executable in C++ before running it can be heavily optimised by the CPU depending on the task being achieved. Python, on the other hand, is an interpretive language so each routine calls on a variety of other already-compiled subroutines. Furthermore, as python does not require type declarations, a great deal of time is spent on performing type checks and then conversions which can be expensive.

It is also unsurprising that numpy and MATLAB outperform C++ in this method. Both these languages rely on specially optimised libraries written mostly C and FORTRAN for matrix manipulations which are heavily optimised.

References

[1] Dahlquist, G. and Bjorck, A. Numerical Methods. Prentice-Hall, Englewood Cliffs NJ, 1974