The Matlab Vectorization Paradigm

by M. Belbut Gaspar and N. Martins-Ferreira

Centre for Rapid and Sustainable Product Development,
Polytechnic Institute of Leiria, Marinha Grande, Portugal

Abstract: We give two examples of application of a vectorized programming paradigm: binary addition and cycle decomposition of directed graphs.

Keywords: Binary addition, Matlab vectorized paradigm, directed graph, cycles decomposition.

1 Introduction

This text was first presented at [5]. In this work we give two examples of application of the "Matlab way of thinking" in programming, using a vectorized paradigm [1, 3].

> A computation is vectorized by taking advantage of vector operations. A variety of programming situations can be vectorized, and often improving speed to 10 times faster or even better. Vectorization is one of the most general and effective techniques for writing fast M-code.
>
> Getreuer, in [1]

The first example is used only for educational purposes and serves as elementary example. It consists in adding one bit to the binary expansion of a number with an arbitrary length. Indeed this is a problem because the operation $(1 + 2^n) - 2^n$ returns the unexpected value of zero if n is greater than 52. The second example is concerned with directed graphs [2] and can be used to extract a simple component from a given closed path, assuming that all the self-intersections are simple.

2 Binary addition

In Matlab, the operation $(1 + 2^n) - 2^n$ returns the unexpected value of zero if n is greater than 52. This is because the system uses 64 bits to represent a number, with one bit for the signal, 52 bits for the mantissa, one bit for the signal of the exponent and 10 more bits to represent its value. This means that the number $2^{53} - 1$ is represented as a binary word with 52 ones and to add one more we would need 53 bits. This problem can be solved in several ways, for instance the fixed-point toolbox is a good option. In any case it is perhaps interesting and educational to try to solve it directly, and for that we would consider the binary representation of an arbitrary number as a vector in Matlab, with an arbitrary length, consisting only of zeros and ones. Then, in order to add one bit to it we would perform the well known procedure for binary addition of a single bit, as illustrated in the following example.

```
for i=1:length(b)
    if b(i)==0
        b(i)=1;
        break
    else
        b(i)=0;
    end
end
```

A more efficient way of doing it is to use built-in Matlab functions and vector indexing. The alternative solution is more cumbersome and difficult to decipher, but gives an example of vectorized programming in Matlab, another such example is presented in the next section.

In the first line we perform the cumulative sum of the negation of b; the result cs is the running count of the number of zeros found from the least-significant bit up to the current bit position. The expression $cs == 1$ thus gives a boolean vector with ones only between the first and second zeroes in b. The second line then sets the bits of first zero in b and up to the next zero, to one. In third line, we address the bits preceding the first zero, setting them to zero (since they originally were, by definition, ones).

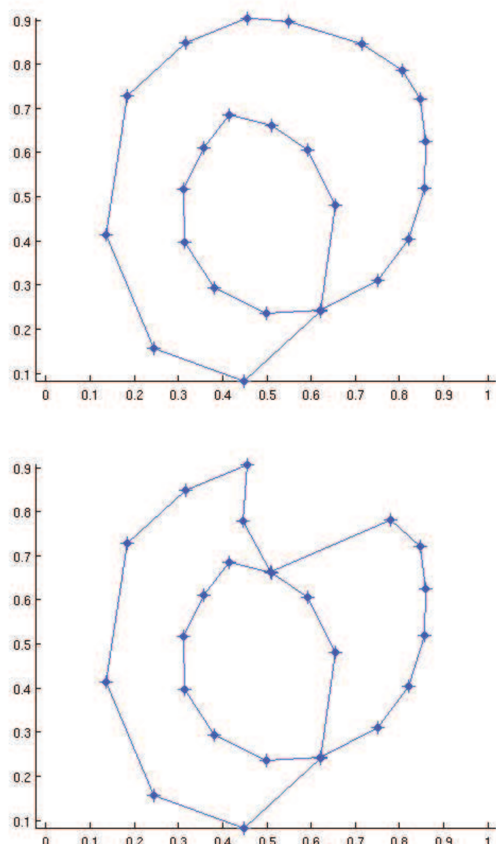
```
cs=cumsum(~b);
b=b|(cs==1);
b=b&(cs~=0);
```

The performance of this implementation is in the same order of the simple decimal addition, but much faster if we take into account the need to convert from binary to decimal and back, and is limited only by the maximum length of the vector b.

3 Directed graphs and cycle decomposition

A directed graph consist of a set of vertices, a set of arrows, and two mappings; source and target, associating respectively a source and a target vertex to each arrow. The following procedure extracts a simple component from a given closed path, assuming that all the self intersections are simple. By a path with simple self intersections we mean a path where every two cycles intersect at most in one vertex, as illustrated in Figure 1(a) on next page.

Moreover, the procedure outlined is capable to determine if the output is valid or not – in other words it tell us if the given closed path is such that all self intersections are simple. An example of a path where not all self intersections are simple is shown in Figure 1(b). This process has several applications in general, in the theory of directed graphs, where the in-degree of each vertex coincides with the respective out-degree. In particular, the application where we use it is related to the slicing and scanning of triangulated surfaces in the aim of rapid prototyping manufacturing.



(Figure 1 - A closed path with: (a) simple self intersections; (b) non simple self intersections.)

4 The code

Given an arbitrary path P in an arbitrary directed graph (i.e. P is any vector in Matlab) the procedure returns a path with the same end points with no repetition of vertices. Example, the code described below, for P=[1 2 3 4 2 5 6 7 6 8] returns [1 2 5 6 8].

```
% idxS is such that P(idxS)=S and S is sorted
[S, idxS]=sort(P)

% idxR is such that idxS(idxR)=1:end
[~,idxR]=sort(idxS)

% mark repeated vertices in P
rep=~diff(S([1:end,1]));
n1=-diff(~rep([end,1:end]));
n=cumsum(n1(idxR));

%returns the simple path
P=P(n==0);
```

A simple change in the code above allow us to obtain the simple components of the given path.

Vectorizing your code is worthwhile for several reasons:

Appearance: Vectorized mathematical code appears more like the mathematical expressions found in textbooks, making the code easier to understand.

Less Error Prone: Without loops, vectorized code is often shorter. Fewer lines of code mean fewer opportunities to introduce programming errors.

Performance: Vectorized code often runs much faster than the corresponding code containing loops.

The reader can find a more detailed exposition on this type of programming in the references below.

References

[1] P. Getreuer, Writing fast Matlab code, 2008, http://www.mathworks.com/matlabcentral/fileexchange/5685

[2] J.M.S. Simões Pereira, Matemática Discreta: Grafos, Redes, Aplicações, Editora Luz da Vida, 2009.

[3] S. Attaway, Matlab - a Practical Introduction to Programming and Problem Solving, 2nd Edition, Elsevier, Oxford 2012.

[4] D. J. Higham and N. J. Higham, Matlab Guide, SCIAM, Philadelphia 2000.

[5] M. Gaspar and N. Martins-Ferreira, The Matlab Vectorization Paradigm, CSEI2012 – Conferência Nacional sobre Computação Simbólica no Ensino e na Investigação, Lisboa, 2-3 Abril de 2012.