FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Software Selection

Luis Cruz

PROVISIONAL VERSION

March de 2010

Contents

List of Figures

List of Tables

Abbreviations

- VPN - Virtual Private Network
- IC - Integrated Circuit
- BER - Bit Error Rate

Chapter 1

Software Selection

1.1 Scientific Tools - State of Art

The use of high level languages to increase productivity is present on a lot of Scientific domains like: Engineering, Physics, Neuroscience, etc... Productivity is a major concern on the current times, it's necessary to develop more products with less time spent.
On this chapter we will focus on two languages: Python and Matlab. There are a lot of differences between them, but the major difference is price, Python is open source and very well spread on the different Scientific Domains, instead Matlab needs a license and is used by Engineers only (off course there are exceptions). Another tool that I will take in consideration is Octave, because it's a free Matlab Clone.
On the next sub-chapters I will try to describe the differences between them in terms of speed, Macro Functions and user interface. At the end I will select one of them that will be used for the software tool implementation.

1.1.1 Free Software

All the Scientific domains require software processing tools, this ones are used to translate numbers into plots, graphs, tables, etc. Without this step the conclusions wouldn't appear. It's necessary to process the input data in order to extract the desired information.
The Software price increases with the low number of applications supported, or in other words, if there a few users the final price to the client will be higher. The other problem that the Proprietary software deals is the piracy. Let's think a school, there are a lot of lectures that requires specific software, the Proprietary version of this software is only available within the school, so this gives two options to the students: The student will work to the specific project under the school (physically or via VPN) or the student get a Pirate version and use it at home or he/she finds a free version and use it. The first option will be dependent of the available machines and will be dependent of configurations, CPU usage and Data location. The second option isn't recommended. The third option can be the easiest way to address this problem (normally the free software only offers the must used functions).

The usage of free software within the Science could be very helpful to everyone, because the free software normally provides the source code, the availability of the source code can conduct to improvements. Is known that what influences the development of science most is communication: when researchers have the possibility to share and compare ideas it is harder to make mistakes. This is why Internet is helping science so much [1].
Another issue is related with backward compatibility, usually the new software version have new output formats, this formats aren't normally supported by the previous versions. This obliges to an (usually expensive) update even if there is no actual need for it.
The use of free software under the University world could result in bigger achievements to the students, because Students have to learn, and looking into the source of a software is the best way to learn how it actually works. Studying the algorithm of some free scientific programmes , students will be able to better understand the matters and will be also able to improve or add new features to the software. The best way to understand the theory is applying them.

1.1.2 Scientific Programming in Python

The Scientific groups are trying to use open source software to address the new projects, the use of Python for Scientific porpoises has been increasing, they refer that the availability of a command line interpreter is an advantage, because this allow the user to debug the code in a faster way.
Matlab has been used in a large scale under the Scientific groups, but this software is payed and is not very fast. On the recent projects software speed is one of the Key tasks, the idea of increase the software speed is related with productivity that is one of the Keys to have success.
The neuroscience groups are moving to python according to [3], they move from Matlab to python and the results are good: *"We feel that coding in Python is more enjoyable, and that it takes significantly less time and effort to write and maintain Python code, resulting in greater productivity. Switching from MatLab has allowed us to tackle more complicated software projects with relative ease. The field of neuroscience is moving towards greater adoption of Python, which should help encourage collaboration through greater sharing of code"*.
Lets talk about of the advantages of Matlab and python:

- Matlab

  Already widely used

  Designed specifically for scientific computing

  Easy to find documentation

  Good IDE with debugging and profiling support "out of the box"

- Python

  Open source means no limits on use

  Appears to approximately superset MATLAB's functionality

Modern language with support for object orientation

Support for calling functions in other languages

Object oriented

A major advantage of python is the cross platform functionality, with just one code we can create executable/binary files to different platforms (Windows, Linux and Mac). This functionality is very important, because the software should be available in different platforms.

Inside the python website there are a lot of examples, but I will present just one: `http://www.python.org/about/success/usa/`, the United Space Alliance (USA) describes their success story, I will jus emphasize one topic:

"Software engineers have long told their bosses and clients that they can have software fast, cheap, or right, as long as they pick any two of those factors. Getting all three? Forget about it! But United Space Alliance (USA), NASA's main shuttle support contractor, had a mandate to provide software that meets all three criteria. Their experience with Python told them NASA's demands were within reach". This was really achieved.

Python is enhanced by a large set of scientific libraries that are being actively developed. Suppose you want:

- Standard science and engineering functions or plotting (MATLAB)

  Numpy, SciPy, Matplotlib

- A computer algebra system (Mathematica)

  SAGE

- Data processing

  Modular toolkit for Data Processing (MDP)

- Bioinformatics functions

  Biopython

- Machine learning functions

  PyML, mlpy, SHOGUN

- Neural nets

  Fast Artificial Neural Network (FANN) Library

- Artificial intelligence or robotics routines

  Python Robotics (Pyro)

- GUI interface

  WXPython, Tkinter

1.1.2.1 NumPy Package

NumPy is the fundamental package needed for scientific computing with Python. It contains among other things:

- A powerful N-dimensional array object

  For a list of NumPy functions, please take a look at http://www.scipy.org/Numpy_Example_List

- Sophisticated (broadcasting) functions

- Tools for integrating C/C++ and Fortran code

- Useful linear algebra, Fourier transform, and random number capabilities.

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

Why is these extension needed?
From http://numpy.sourceforge.net/numdoc/HTML/numdoc.htm#pgfId-78440: *"The core reason is a very prosaic one, and that is that manipulating a set of a million numbers in Python with the standard data structures such as lists, tuples or classes is much too slow and uses too much space. Anything which we can do in NumPy we can do in standard Python - we just may not be alive to see the program finish. A more subtle reason for these extension however is that the kinds of operations that programmers typically want to do on arrays, while sometimes very complex, can often be decomposed into a set of fairly standard operations. This decomposition has been developed similarly in many array languages. In some ways, NumPy is simply the application of this experience to the Python language - thus many of the operations described in NumPy work the way they do because experience has shown that way to be a good one, in a variety of contexts. The languages which were used to guide the development of NumPy include the infamous APL family of languages, Basis, MATLAB, FORTRAN, S and S+, and others. This heritage will be obvious to users of NumPy who already have experience with these other languages."*

1.1.2.2 SciPy Package

The SciPy library depends on NumPy, which provides convenient and fast N-dimensional array manipulation. The SciPy library is built to work with NumPy arrays, and provides many user-friendly and efficient numerical routines such as routines for numerical integration and optimization. NumPy and SciPy are easy to use, but powerful enough to be depended upon by some of the

world's leading scientists and engineers.
SciPy provides modules for:

- Statistics
- Optimization
- Numerical integration
- Linear algebra
- Fourier transforms
- Signal processing
- Image processing
- Genetic algorithms
- ODE solvers
- Special functions
- and more

SciPy is developed concurrently on both Linux and Windows. It should also compile and run successfully on Mac, Solaris, FreeBSD, and most other platforms where Python is available.

1.1.2.3 matplotlib Package

matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. matplotlib can be used in python scripts, the python and ipython shell (ala, matlab or mathematica), web application servers, and six graphical user interface toolkits.
matplotlib tries to make easy things easy and hard things possible. We can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc, with just a few lines of code.
The mplot3d toolkit has support for simple 3d graphs including surface, wireframe, scatter, and bar charts.The toolkit is included with all standard matplotlib installs. Please take a look at figure 1.1.

The plotted can also be saved in a variety of formats like: pdf, png, ps, eps and svg. You can also find some supported plotting commands under: `http://matplotlib.sourceforge.net/index.html`.

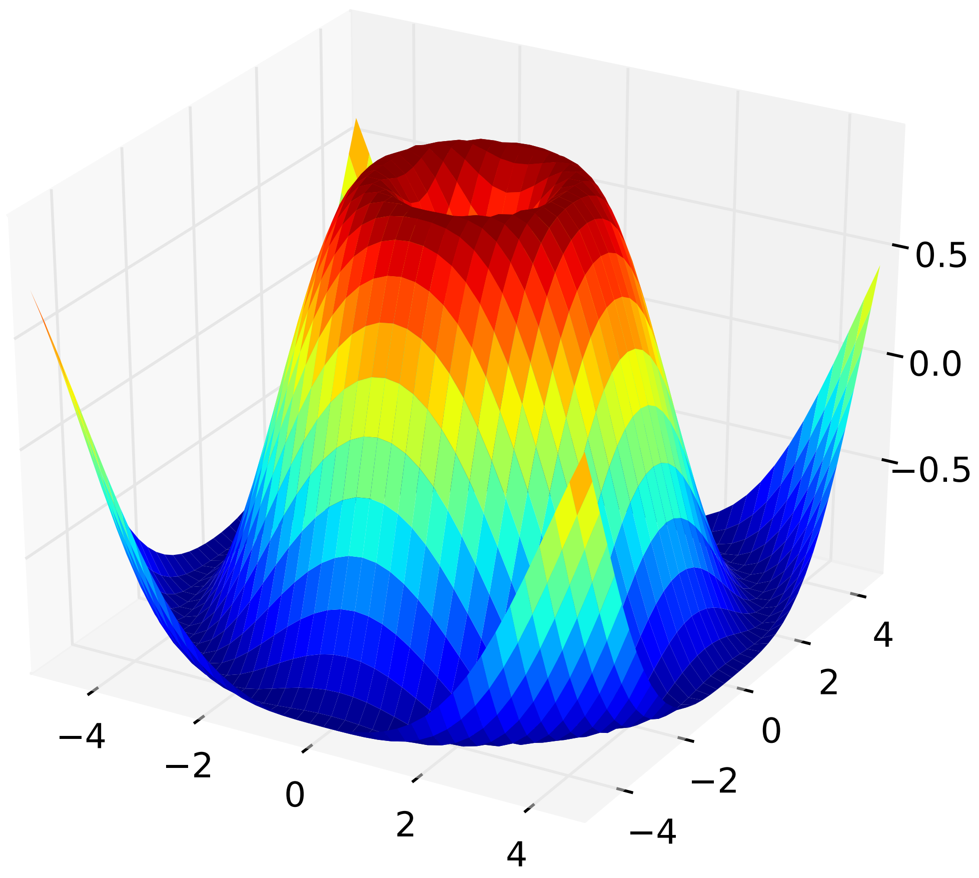

Figure 1.1: 3D figure using matplotlib package

1.1.2.4 Pyrex Package - *A language for writing Python extension modules*

Pyrex is a language specially designed for writing Python extension modules. It's designed to bridge the gap between the nice, high-level, easy-to-use world of Python and the messy, low-level world of C.
It's possible to write C modules without using Pyrex, but the task isn't easy, because it's necessary to convert the variable formats between python and C, as well as to use API calls. So the idea present on this package is to allow C performance with similar python language description. At the end the Pyrex code is converted into pure C code that it's compiled and use as a package to Python.

1.1.3 Performance Evaluation

1.1.3.1 Octave and Python: High-Level Scripting Languages Productivity and Performance Evaluation

This article [2] evaluates the difference between tree scripting languages(Matlab, Octave - Matlab Clone and Python) in terms of run speed.In order to address the major task (performance evaluation) three algorithms are used:

- 2DFFT - For this algorithm both languages have access to optimized 2D FFT algorithms that are ready-to-use and much faster than manually coded implementations.
- Pattern Matching - To further test the feasibility of Octave and Python for High Performance platforms, a more complex Signal processing algorithm was then implemented in each language. This algorithm consist on a template image matching within a field image.
- General Benchmarks - A series of benchmarks are used, they can be found under `http://www.sciviews.org/benchmark/`. This benchmarks are available for a few tools but python is not one of them. In order to run this benchmarks in python the Numpy package was used.

They concluded that productivity and performance results for each language are very depending on the specific task and the availability of high level functions in each system to address such tasks. Therefore, the choice of the best language to use in a particular instance will strongly depend upon the specifics of the Signal/Image Processing problems to be solved. Table 1.1 shows the average run time for each algorithm.

1.1.3.2 Python Performance

Inside the ScyPy.org website we can find a beginners guide to using Python for performance computing. This page explains the different packages that can be used to address a problem. The idea is to solve a 2D Laplace equation using an iterative finite difference scheme (four point averaging, Gauss-Seidel or Gauss-Jordan).

Python has a ability to run classes defined in other languages like: C++, C or Fortran. This languages are very fast, so the final result should be better if we use this interface. In order to make this process easier to packages are available: Pyrex, Inline and Blitz. The results (see figure 1.2) shows that Python can be as fast as Matlab or even better, depending of the package used. So taking into consideration the rule 90/10 for software, 90% of the time spent is in 10% of the code. We can easily port this code to faster languages like C, in order to increase the final software speed.

1.1.3.3 My Test Case

In order to correct evaluate the differences between all the packages, a simple testcase was created. The testcase is based on 5 term filter, the idea is to apply a low pass filter to an analog signal and after find the zero crossing points. The filter has 5 terms with different weights.
The module installation was very easy, because there are self-extract packages for Windows. The machine used was a intel P4-2.8GHz with 160gb(5400 rpm) of disc space and 1GB of RAM.
Figure 1.3 describes the time spent by each module to find the zero crossing point of each file. The files have data extracted from a HDMI TX macro. There was 2 packages used:

- Base
  - Used Append function - The construction of the vector with the zero crossing point was done using an append approach, so the vector starts with only one position and then it grows with the addition of elements.
  - No Append function - The dynamically construction of vectors isn't efficient, because it's necessary to allocate memory, copy the current vector and then add the element. In order to reduce the memory allocations, the zero crossing vector is a copy of one bigger vector and at the end the vector returned only has the valid elements.
- NumPy package - The use of this package implies that the input vector should be a NumPy matrix, so at the beginning it's necessary to convert the python lists to NumPy matrix. This action takes 50% of the total spent time on this testcase.
- Pyrex Package
  - Pyrex package with Append function - The pyrex code can use append() functions, but again, this isn't the best option.
  - Pyrex package without Append function - Instead of using append functions an initial memory allocation was used. This removes the need for memory reallocation and vector copy.

As expected the Pyrex code has the best results, because this is equivalent to C. The strange behavior is the NumPy module, the speed should be better, but this performance can be explained by the need to convert python lists to NumPy matrix.

Table 1.1: Tools Performance Evaluation with different algorithms (average run time)

	Octave	Matlab	Python
2DFFT	0.802s	0.943s	0.860s
Pattern Matching	26.805s	2.6625s	14.2265s
General Benchmarks	0.8s	0.295s	0.75s

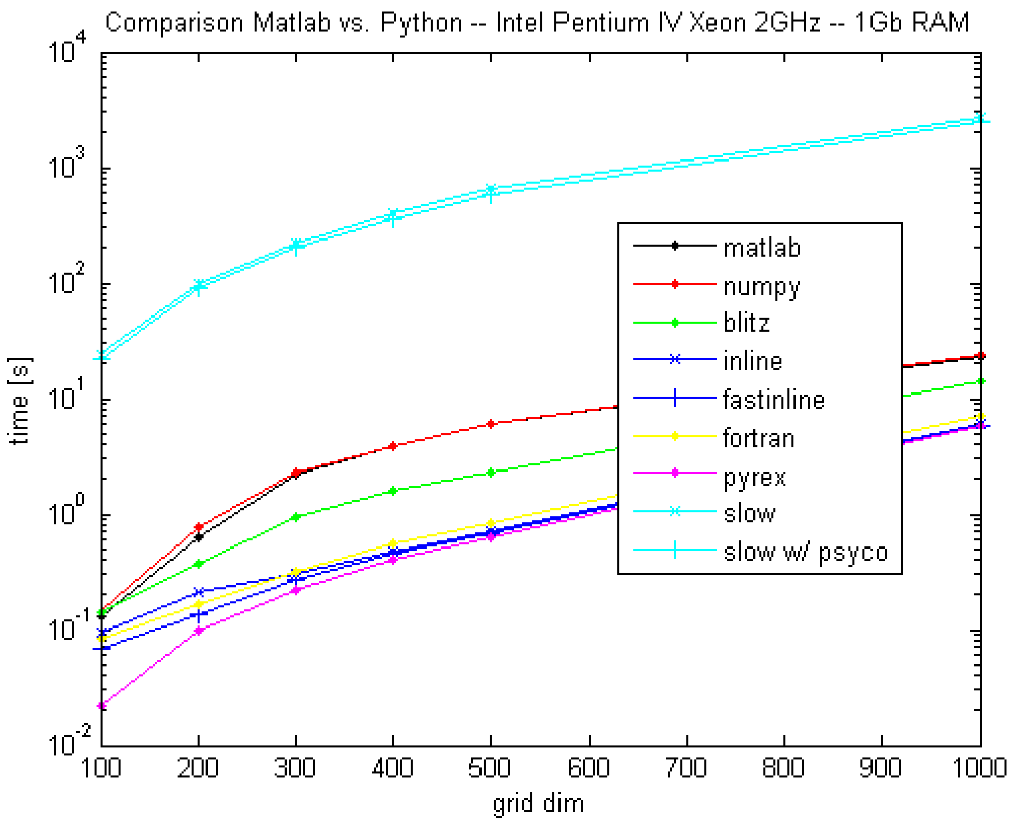

Figure 1.2: Comparison Matlab vs Python - Solving a 2DLaplace equation

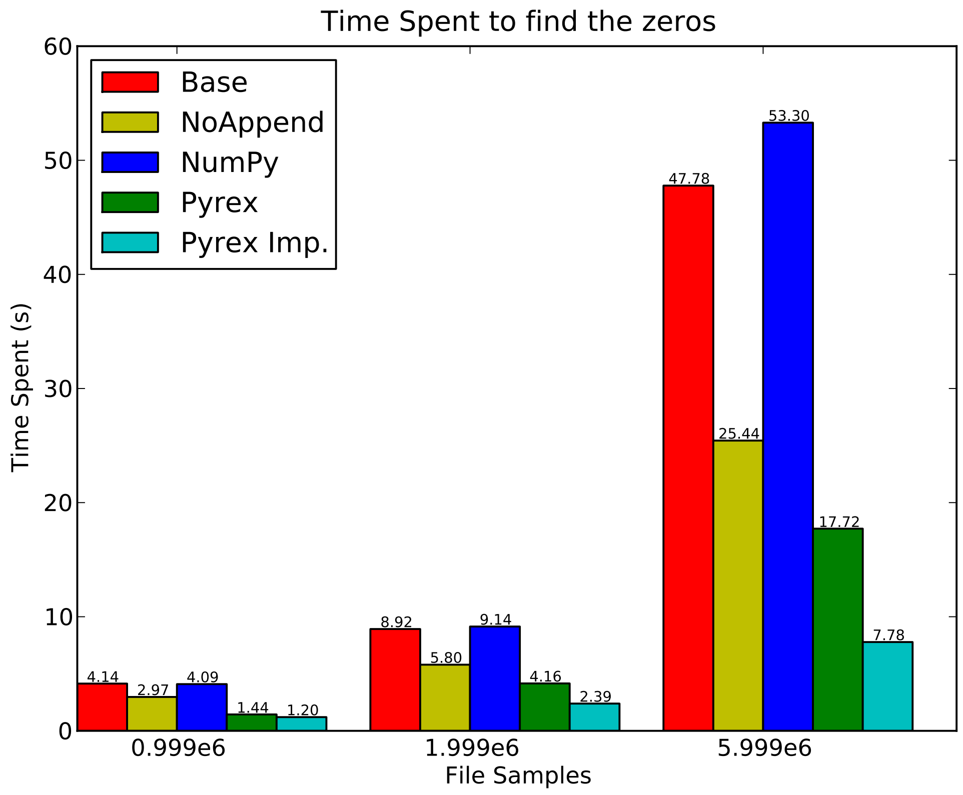

Figure 1.3: Comparison Between Python Packages

Bibliography

[1] A. Damato, “Why the future of science must be in free software,” 2005.

[2] B. G. J. G. S. A. A. K. J. U. A. C. A. W. Juan Carlos Chaves, John Nehrbass and O. Siddharth Samsi Ohio Supercomputer Center, Columbus, “A high-level scripting languages productivity and performance evaluation,” *HPCMP Users Group Conference (HPCMP-UGC’06)*, 2006.

[3] J. Spacek and N. Swindale, “Python in neuroscience,” *The Neuromorphic Engineer*, 2009. [Online]. Available: http://www.ine-web.org