The Python and Julia Programming Languages

scientific, numerical and signal processing

José Jasnau Caeiro
j.caeiro@ipbeja.pt

Lab SEPSI
Instituto Politécnico de Beja

Beja, 15^{th} May 2014

Signals and Signal Processing

An informal definition of **signal** is,

Definition

A signal is a function of independent variables such as **time**, **distance**, **position**, **temperature**, *and* **pressure** *(S. K. Mitra, 2002)*

another **signal processing** is, according to *Wikipedia* (2014),

Definition

Signal processing is an area of Systems Engineering, Electrical Engineering and Applied Mathematics that deals with operations on or analysis of analog as well as digitized signals, representing time-varying or spatially varying physical quantities.

Examples of Signals

Some examples of signals are:

sound air pressure as a function of time at a certain point in 3D space;

electromagnetic radiation

ECG electrocardiography signal, a function representing the electrical activity;

images photographs or video frames;

sensor generated signals temperature, pressure, etc.

Typical Signal Processing

Data structures and operations:

- matrix operations;
- discrete vector based processing;
- Fast Fourier Transform;
- Discrete Transforms;
- filtering;
- 1D and 2D discrete convolution (a multiply and add operation);
- N-dimensional statistical signal processing;
- image point and local based operators.

And most of the time several data formats must be supported and also scientific plotting.

Some historical landmarks

Fortran a general purpose imperative programming language (late 1950's)

- high performance compiler;
- huge set of numerical and scientific libraries;
- static memory allocation;

C another general purpose imperative programming language (1973)

- maps efficiently to typical machine instructions;
- available for the majority of computer architectures and operating systems, namely microcontroler and digital signal processors (DSPs);

MATLAB

The *golden standard* programming language used for **signal processing** research and development

- developed in the late 1970s to give easy access to students to **LINPACK** and **EISPACK** Fortran algebraic libraries;
- popular in applied mathematics, control engineering, image processing, numerical analysis and signal processing;
- simple syntax with *dynamic programming languages* properties;
- can call functions and subroutines written in **C** or **Fortran**;
- large set of functions and *toolkits* (a set of functions and libraries dedicated to a specific field).

MATLAB

- **MATLAB** is a proprietary product of **MathWorks**;
- very expensive with toolboxes sold separately using a license model;
- last release is **MATLAB** 8.3;
- offers some support for parallel processing;
- poor syntax for advanced programming and data structures, namely object oriented programming, lists, dictionaries, etc.
- offers support to symbolic processing through **MuPAD**;
- **Java**, **Perl** or **.NET** libraries can be directly called from **MATLAB**;

MATLAB and alternatives

Some open source alternatives have appeared offering a large extent of compatibility:

GNU Octave appeared around 1988 and is mostly compatible with **MATLAB** adopting a GNU General Public License;

Scilab appeared around 1990 within INRIA, it is the most compatible language with **MATLAB**, and although free presents a commercial support through *scilab enterprises*;

FreeMat another free alternative under GPL and with 95% compatibility with **MATLAB**;

ScalaLab provides an efficient **MATLAB** like working style for the *Java Virtual Machine* and is programmed in **Scala**.

General Properties

- **Python** appeared in 1991 and is one of the most successful *dynamic programming languages*;
- supports a large set of programming paradigms;
- programmers productivity is high due to the language and the large set of libraries available;
- easy to interface to **C**;
- good support for high level data structures.

numpy, scipy and matplotlib

In 2005 a stable numerical package proposal was created by Travis Oliphant, based on previous ancestor packages:

- **NumPy** provides functionality comparable to **MATLAB**;
- easily interfaces with **C**;
- **SciPy** extends this functionality;
- **Matplotlib** is an advanced plotting package;

A scientific programming framework joining several packages was created using **Python**: **sage**.
It is very large (several GB and hundreds of packages) and is currently under version 6.2.

Criticism of MATLAB

- **MATLAB** is proprietary and costly;
- **MATLAB** is used by only 0.711% of the general programming community;
- **MATLAB** data structures are cumbersome and does not support generic programming;
- **MATLAB** is slow when **cycles** and **selections** are involved.

Criticism of Python

- not compatible with **MATLAB**;
- due to design there are some multi core programming limitations;
- slow when **cycles** and **selections** are involved.

The Julia programming language

The creators of **Julia** are based on the Massasuchets Institute of Technology and are:

- *Alan Edelman*, professor of Applied Mathematics at MIT;
- *Stefan Karpinski*, a programmer;*Jeff Bezanson*, another programmer; *Viral Shah*, a Bangalore programmer that works on Julia from India.

And is being used at

Julia is used at:

- Stanford University, Spring 2014, AA222, Introduction to Multidisciplinary Design Optimization (Prof. Mykel J. Kochenderfer)
- Pennsylvania State University, Spring 2014, ASTRO 585, High-Performance Scientific Computing for Astrophysics (Prof. Eric B. Ford)
- Cornell University, Spring 2014, CS 5220, Applications of Parallel Computers (Prof. David Bindel)
- MIT, Spring 2014 18.330, Introduction to Numerical Analysis (Dr. Homer Reid), 15.S60, Software Tools for Operations Research (Iain Dunning), 15.083, Integer Programming and Combinatorial Optimization (Prof. Juan Pablo Vielma)

Why Julia?

- The authors wanted a programming language with the speed of **C** and the dynamic programming features of **Python** and **Ruby**;
- with a familiar syntax as **MATLAB**;
- usable for general programming as **Python**;
- supportive of **Hadoop**, an open source software framework for storage and large-scale processing of data-sets on clusters of commodity hardware (Wikipedia quote);
- easy to learn to people coming from the signal processing and machine learnig community.

Julia

Some characteristics of the **Julia** programming language:

- modern dynamic programming language (development started in 2009);
- designed for technical and scientific computing;
- syntax familiar for **MATLAB** programmers;
- calls **Python** and **C** functions directly;
- designed for parallel and distributed computation;
- MIT licensed: free and open source;
- developed at MIT.

A larger set of features

- Multiple dispatch: providing the ability to define function behavior across many combinations of argument types;
- Dynamic type system: types for documentation, optimization, and dispatch;
- Lisp-like macros and other metaprogramming facilities;
- multiprocessing environment based on message passing;
- parallel processing with GPUs based on **OpenCL** with bindings based on **PyOpenCL** (a **Python** based package), `https://github.com/JuliaGPU/OpenCL.jl`;
- Coroutines: lightweight "green" threading;
- LLVM-based just in time compiler.

Syntax example MATLAB

```
lena = imread('lena512.pgm');
lena = double(lena);channel = size(lena, 3);
height = size(lena, 1);width = size(lena, 2);
lenaOutput = zeros(size(lena));
Gx = [1 2 1; 0 0 0; -1 -2 -1];
Gy = [1 0 -1; 2 0 -2; 1 0 -1];
for i = 2 : height - 1
    for j = 2 : height - 1
        for k = 1 : channel
            tempLena = lena(i-1:i+1,j-1:j+1,k);
            x = sum(sum(Gx .* tempLena));
            y = sum(sum(Gy .* tempLena));
            pixValue = sqrt(x^2 + y^2);
            lenaOutput(i, j, k) = pixValue;
        end
    end
end
```

codigo/sobel.m

Syntax example Julia

```
using Images
lena = imread("lena512.pgm");lenaOutput = zeros(size(
    lena))
lena = float(lena);channel = size(lena, 3)
height = size(lena, 1);width = size(lena, 2)
Gx = [1 2 1; 0 0 0; -1 -2 -1]
Gy = [1 0 -1; 2 0 -2; 1 0 -1]
for i = 2 : height - 1
    for j = 2 : height - 1
        for k = 1 : channel
            tempLena = lena[i-1:i+1,j-1:j+1,k]
            x = sum(sum(Gx .* tempLena))
            y = sum(sum(Gy .* tempLena))
            pixValue = sqrt(x^2 + y^2)
            lenaOutput[i, j, k] = pixValue
        end
    end
end
```

Availability and Development Tools

An important issue...

- availability
  - It is available for **Windows** 32 and 64 bit versions;
  - **OSX** 10.6 and above;
  - **Linux**;
  - source code.
- editors
  - **emacs** mode is available;
  - an **eclipse** plugin is being developed (`https://hacknight.in/thejulialang/julia-hacknight/projects/7-ide-for-julia-eclipse-integration`);
  - an IDE called **Julia Studio** is available (`https://github.com/forio/julia-studio`);
  - syntax highlighting is available for **Notepad++** (`https://gist.github.com/catawbasam/3858496`).

Julia and Python Notebooks

A **Mathematica** *notebook* inspired development environment appeared within the **Python** scientific community:

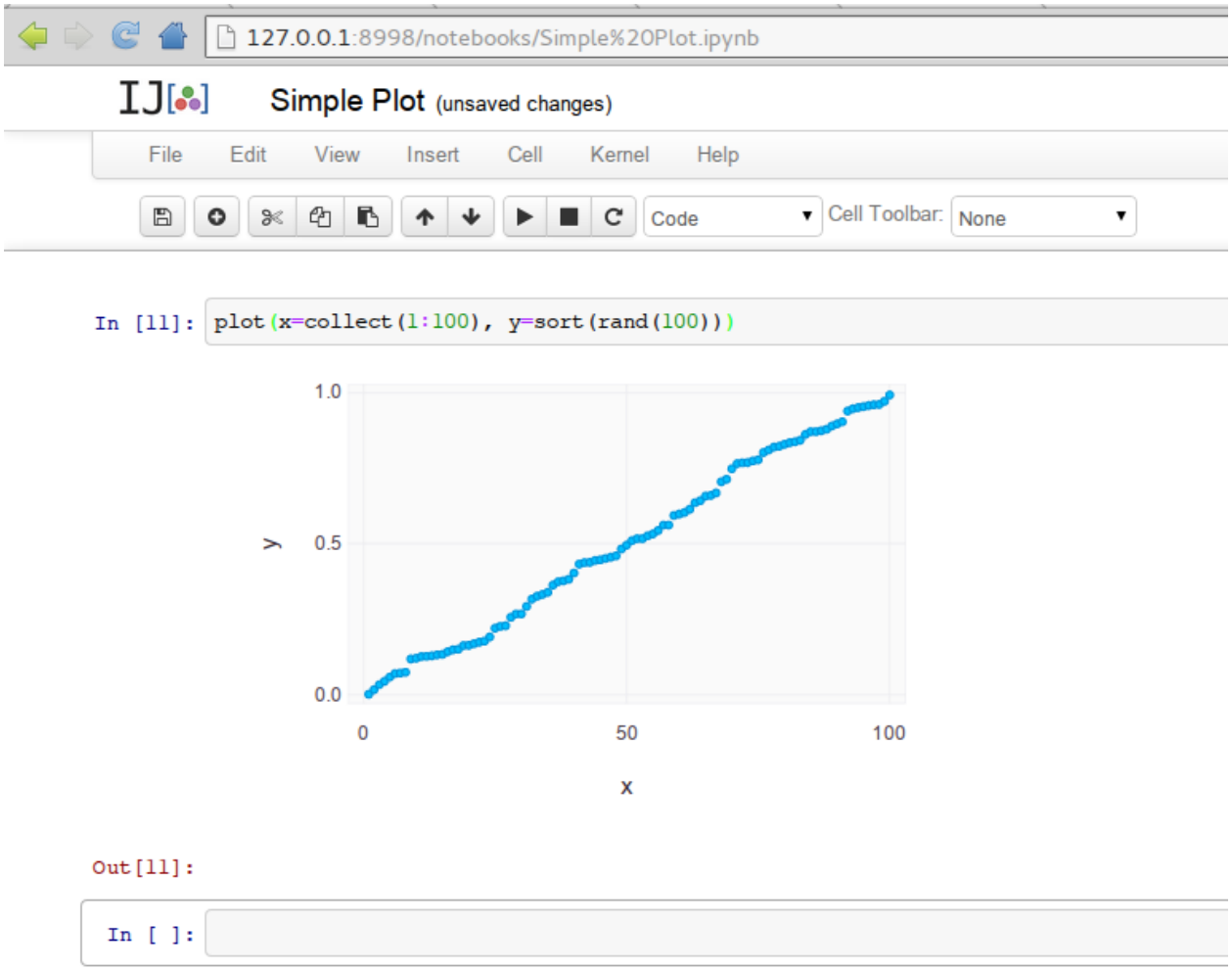
- it is based on the **IPython** interactive kernel, `http://ipython.org/`;
- in 2014 appeared a book by *José Unpingco* based on IPython notebooks, *Python for Signal Processing: featuring IPython Notebooks*, Springer-Verlag;
- it features a browser based editing environment;
- integrates **NumPy** and **SciPy**;
- fully integrates the powerfull graphics package **matplotlib**, `http://matplotlib.org/`;
- supports high quality mathematical typesetting using `MathJax`, which is a Javascript version of most of LaTeX;
- currently works using *websockets* thus only supports `firefox` and `google chrome`.

Julia and Python Notebooks

The **IPython** *notebook* development environment is also available for **Julia**:

- brings an easy to use development environment to **Julia**;
- since **Julia** calls directly **Python** scripts, a very large set of programming resources is available;
- provides a familiar interactive development environment.

IJulia Notebook



Speed Comparison

According to the Julia site (`http://julialang.org/`), some micro-benchmarks results were obtained on a single core (serial execution) on an Intel Xeon CPU E7-8850 2.00GHz CPU with 1TB of 1067MHz DDR3 RAM, running Linux:

	Fortran gcc 4.8.1	Julia 0.2	Python 2.7.3	R 3.0.2	Matlab R2012a	Octave 3.6.4	Mathematica 8.0	JavaScript V8 3.7.12.22	Go go1
fib	0.26	0.91	30.37	411.36	1992.00	3211.81	64.46	2.18	1.03
parse_int	5.03	1.60	13.95	59.40	1463.16	7109.85	29.54	2.43	4.79
quicksort	1.11	1.14	31.98	524.29	101.84	1132.04	35.74	3.51	1.25
mandel	0.86	0.85	14.19	106.97	64.58	316.95	6.07	3.49	2.36
pi_sum	0.80	1.00	16.33	15.42	1.29	237.41	1.32	0.84	1.41
rand_mat_stat	0.64	1.66	13.52	10.84	6.61	14.98	4.52	3.28	8.12
rand_mat_mul	0.96	1.01	3.41	3.98	1.10	3.41	1.16	14.60	8.51

Figure: benchmark times relative to C (smaller is better, C performance = 1.0).

To digest with some care...

Conclusion

- **Python** is still inspiring new programming languages proposals
- **Julia** is a new scientific programming language proposal (still at version 0.30) but soon expected to jump to 1.0, already being used to teach numerical programming at the MIT
- **Julia** is modern dynamic programming language
- **Julia** designed to be faster than **Matlab** and **Python** and with better support por multiprocessing and paralelism
- is still at its infancy...

THANK YOU