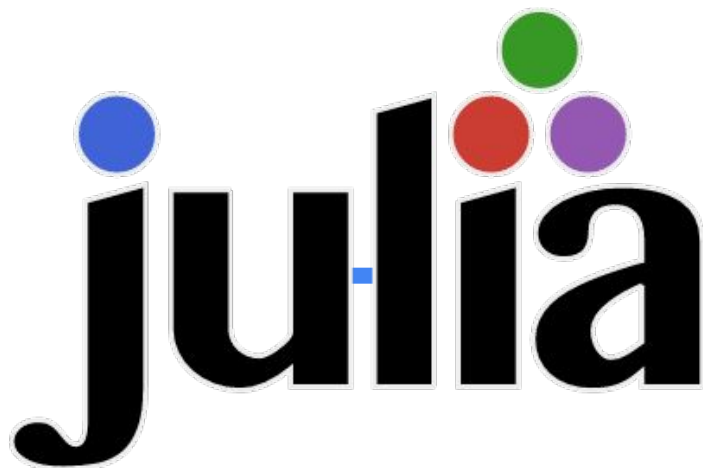


Introduction to



Shashank Shetty
Kalavara

Tomasz Niewiadomski

and its usage in scientific computing

**Advantages of
over the status
quo.**





Language interface

Elegant for science.

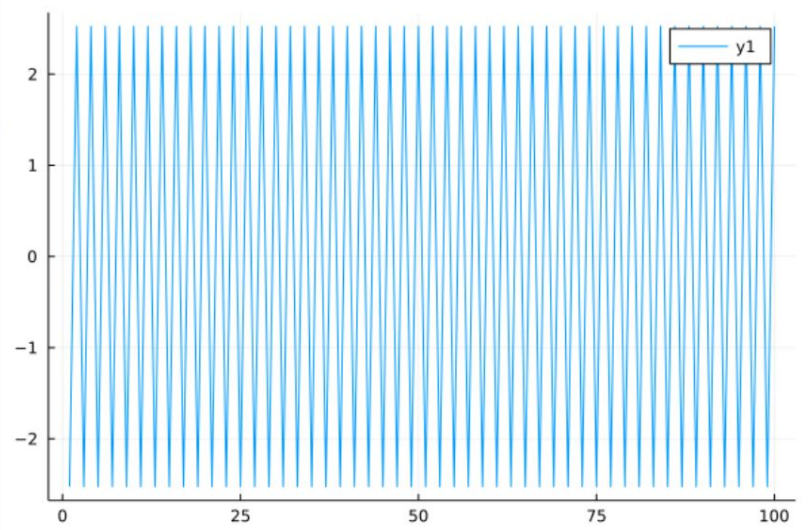
Intuitive mathematical layouts which makes reading the code much easier for people in sciences.

File Edit Selection View Go Run Terminal Help

function radius(x,y,z) Untitled-1 • using Plots Untitled-2 •

1 using Plots ✓
2 $\zeta(\theta) = A \sin(\pi \theta + \varphi)$ ζ (generic function with 1 method)
3 A= 3 3
4 $\varphi=1$ 1
5
6 $\theta= 1:100$ 1:100
7
8 plot($\zeta.(\theta)$) | Plot{Plots.GRBackend() n=1}

Julia Plots (7/7) ×



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

GKS: Possible loss of precision in routine SET_WINDOW

1:1000

 ζ (generic function with 1 method)

1:100

julia>

Julia REPL + -

Ln 8, Col 12 Spaces: 4 UTF-8 CRLF Julia Main

From Live demonstration of - simple language interactions

Typical trade-off that developers make when choosing a language — it can either be relatively easy for humans to write, or relatively easy for computers to run, but not both.



The 2 language problem.

Language Abstractions



Matlab



Python

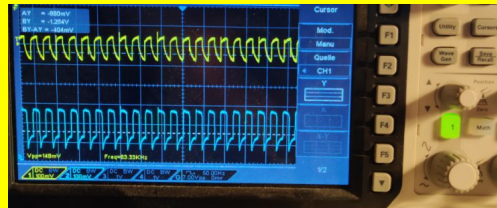
Java

C

Fortran

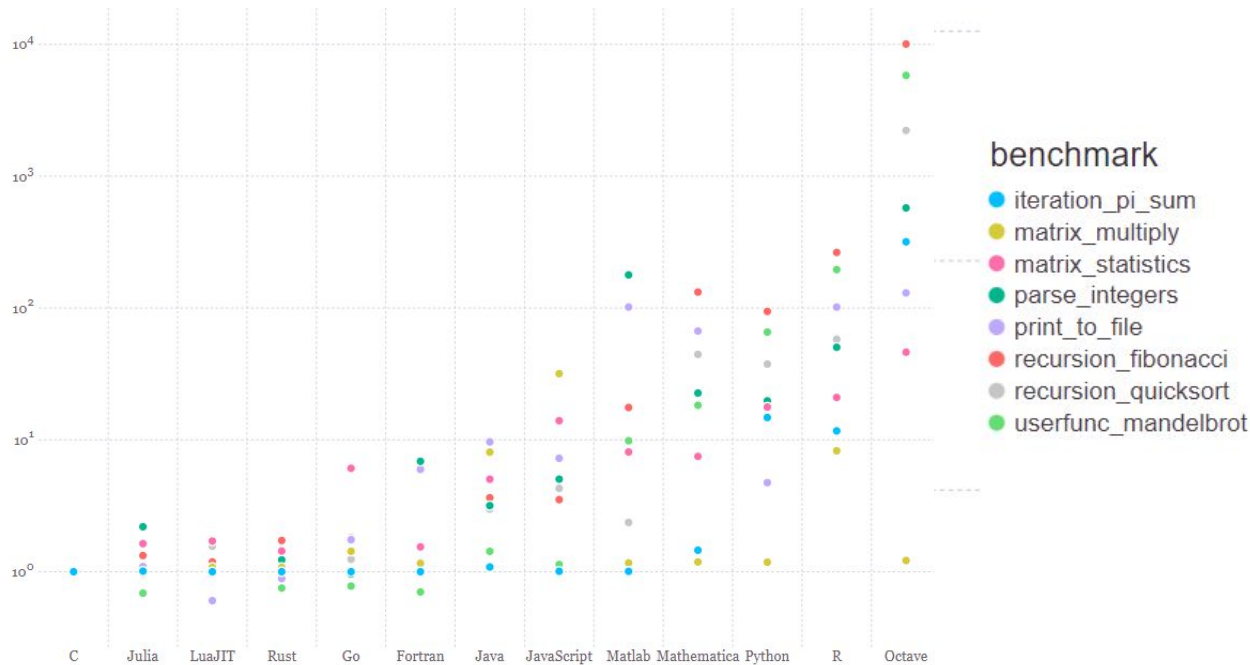
Assembly

Punch card
programming



Level of
abstraction

1010111010110101



Fast as long as you know what
you are doing !!!

Some performance benchmark comparisons.

BLAS [Basic Linear Algebra Subprograms] REMOVE!

Julia implements BLAS interface in difference to CBLAS interface for C.

BLAS implementations are often optimized for speed on a particular machine, so using them can bring substantial performance benefits.

It originated as a Fortran library in 1979, and its interface was standardized by the BLAS Technical (BLAST) Forum.

Fortran is legendary for being on of the fastest matrix computing languages for a very very long time.

Super computer deployments are faster.

- Code is scaled to run on massive computing clusters or supercomputers
- Often the case for scientific computing
- Even small advantages in performance make a huge difference in compute time
- Which is very expensive

Simple Ex. :

Sorting black rocks from a huge database

Huge matrix multiplications



By being a compiled language.

Julia, unlike Python which is interpreted, is a **compiled language** that is primarily written in its own base i.e Julia is written in Julia. However, unlike other compiled languages like C, Julia is compiled at run-time, whereas traditional languages are compiled prior to execution. Julia, especially when written well, can be as fast and sometimes even faster than C. Julia uses the **Just In Time (JIT)** compiler and compiles incredibly fast, though it compiles more like an interpreted language than a traditional low-level compiled language like C, or Fortran.

Compiled language and interpreted language

Recipe for Hummus in Arabic

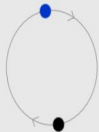
You speak English

Compiled language -> You get an English translated file, with which you start making Hummus

Interpreted language -> You have an Arabic interpreter next to you who tells you what's written and you start making Hummus.

We engineer products and solutions that empower those on the frontlines of healthcare, from *discovery* to *delivery*.

ABOUT US



pumas^{AI}

Accelerated drug development

The one-stop integrated modeling and simulation platform designed to multiply your productivity across the drug development lifecycle.

READ MORE

REQUEST DEMO

Lyv^{AI}

Personalized patient care

A robust clinical decision support system that leverages patient history and targeted medical data for personalized healthcare delivery.

READ MORE

REQUEST DEMO

The world's leading businesses, research labs and universities are choosing us as their healthcare intelligence partners.



Examples



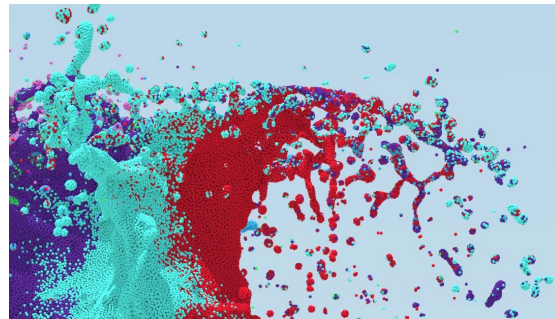
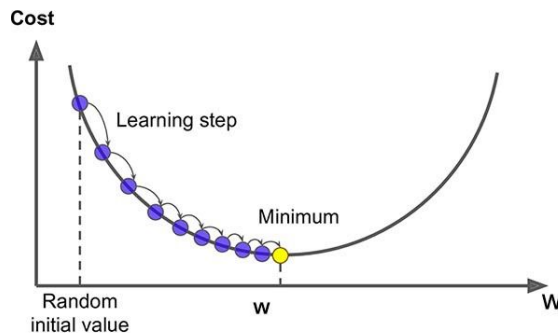
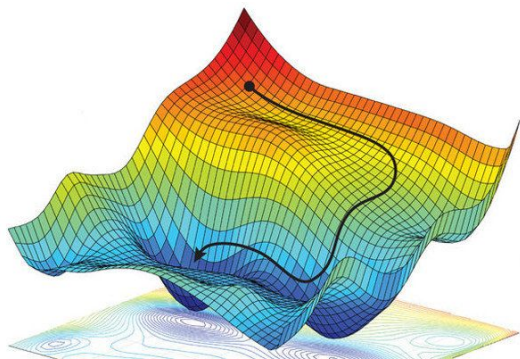
Implementation of

Automatic differentiation

The technical features of Julia, namely, **multiple dispatch**, source code via reflection, **JIT compilation**, and first-class access to expression parsing make implementing and using techniques from automatic differentiation easier than ever before

Advantage of automatic differentiation

Is **Machine Learning**. Especially implementing **gradient descent** or even the activation functions. And solving differential equation for physical **Simulations**.



Automatic differentiation

It is NOT: Symbolic or Manual differentiation

Extremely tedious & Expression swell a lot

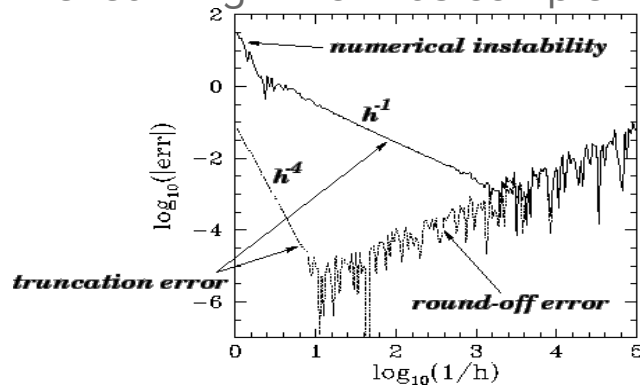
It is NOT: Numerical method

Problems with error [Truncation and rounding] Error accumulation

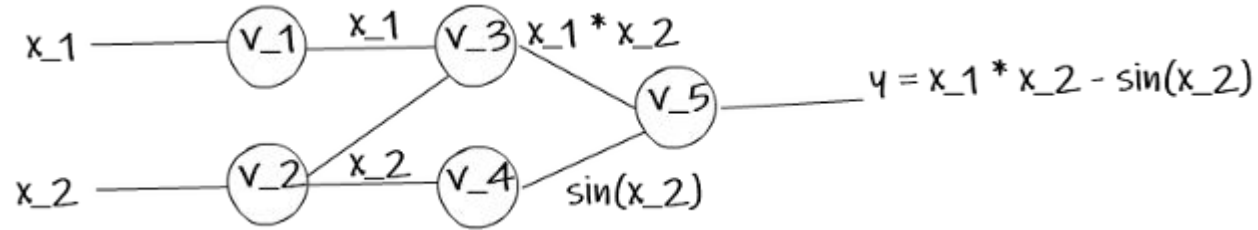
O(n) evaluations each time- especially in machine learning which has complexity with millions of parameters.

$$f'(x) = \lim_{\epsilon \rightarrow 0} \frac{f(x + \epsilon) - f(x)}{\epsilon}$$

Soft ReLU (Softplus)	$\log(1 + e^{wx+b})$
Derivative wrt w_1 for two layers	$\frac{e^{b_1+b_2+w_1x+w_2} \log[1 + e^{b_1+w_1x}] w_2 x}{(1 + e^{b_1+w_1x}) \left(1 + e^{b_2+w_2} \log[1 + e^{b_1+w_1x}]\right)}$



Automatic Diff.



- Automatic Diff. calculate at the same precision as symbolic
- but operates only on the function of interest and gives a numerical value.
- The technical features of Julia, namely, multiple dispatch;
- And others such as :
 - source code via reflection
 - first-class access to expression parsing
 - Just In Time Compilation

multiple dispatch

<https://docs.julialang.org/en/v1/manual/methods/>



function radius(x,y,z) Untitled-1 •



```
1 function radius(x,y,z)
2     sqrt(x^2+y^2+z^2)
3 end | radius (generic function with 1 method)
4 function radius(x,y)
5     sqrt(x^2+y^2)
6 end | radius (generic function with 2 methods)
7 function radius(x)
8     abs(x)
9 end | radius (generic function with 3 methods)
10
11 radius(3,6,7) | 9.695359714832659
12 radius(5,8) | 9.433981132056603
13
14 function glue(x::Float64, y::String)
15     [x ; y]
16 end
17 function glue(x::String, y::String)
18     x*y
19 end |
20 print(glue("Billed", "Tomasz"))
21 print(glue(76.89, "Tomasz"))
```

Excerpt from live VC code demo of multiple dispatch.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

Julia REPL + ▾ □ □ ×

radius (generic function with 1 method)

radius (generic function with 2 methods)

radius (generic function with 3 methods)

9.695359714832659

9.433981132056603

julia>

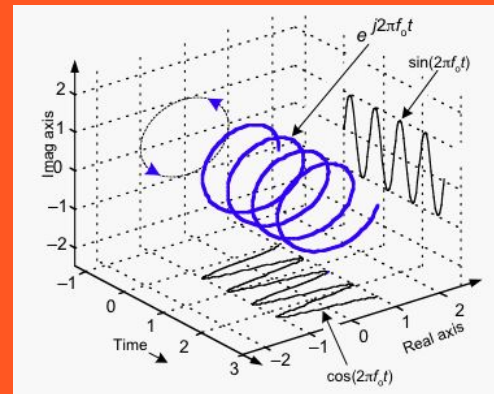


0 0 Julia env: v1.7

Ln 19, Col 5 Spaces: 4 UTF-8 CRLF Julia Main

Dual Numbers #simply explained

- One regular number and one special number
- Akin to complex number or quaternion with 3 special numbers.
- But here you can program the what the special number should behave like. Ex. $(x, \epsilon) \epsilon^2=0$



Resources to learn julia: <https://computationalthinking.mit.edu/Spring21/>
<https://www.youtube.com/c/TheJuliaLanguage>
<https://docs.julialang.org/en/v1/>

Fast Numeric Computation and an amazing interface.

Elegant interface while allowing you to write codes that run blazingly fast on large data sets.

Datasets so large Python sometimes doesn't even have methods to load them.

