

2018-06

ET.

EFFICIENT TOOLS
for RESEARCH



*JULIA: MY NEW FRIEND
FOR COMPUTING & OPTIMIZATION?*

« Julia, my new friend for computing and optimization? »

- Intro to the Julia programming language, for MATLAB users
- *Date:* 14th of June 2018
- *Who:* Lilian Besson & Pierre Haessig
(SCEE & AUT team @ IETR / CentraleSupélec campus Rennes)



Agenda for today [30 min]

1. What is Julia? [5 min]
2. Comparison with MATLAB [5 min]
3. Two examples of problems solved Julia [5 min]
4. Longer ex. on optimization with JuMP [13min]
5. Links for more information ? [2 min]

1. What is Julia ?

- **Open-source and free programming language** (MIT license)
 - Developed since [2012](#) (creators: MIT researchers)
 - Growing popularity worldwide, in research, data science, finance etc...
 - Multi-platform: Windows, Mac OS X, GNU/Linux...
- Designed for *performance*:
 - Interpreted *and* compiled, very efficient
 - Easy to run your code in parallel (multi-core & cluster)
- Designed to be *simple to learn and use*:
 - Easy syntax, dynamic typing (MATLAB & Python-like)

Ressources

- Website:
 - JuliaLang.org for the language
 - & Pkg.JuliaLang.org for packages
- Documentation : docs.JuliaLang.org



Comparison with MATLAB (1/3)

	Julia 😊	MATLAB 😓
Cost	Free 🙌	Hundreds of euros / year
License	Open-source	1 year user license (no longer after your PhD!)
Comes from	A non-profit foundation, and the community	MathWorks company
Scope	Mainly numeric	Numeric only
Performances	Very good performance	Faster than Python, slower than Julia

Comparison with MATLAB (2/3)


	Julia	MATLAB
Packaging	Pkg manager included. Based on git + GitHub, very easy to use	Toolboxes already included but 💰 have to pay if you want more!
Editor/IDE	<i>Jupyter</i> is recommended (<i>Juno</i> is also good)	Good IDE already included
Parallel computations	Very easy, low overhead cost	Possible, high overhead


Comparison with MATLAB (3/3)

	Julia	MATLAB
Usage	Generic, worldwide 🌍	Research in academia and industry
Fame	Young but starts to be known	Old and known... In decline 📉 ?
Support?	Community ¹ : StackOverflow, Forum	By MathWorks
Documentation	OK and growing, inline/online	OK, inline/online

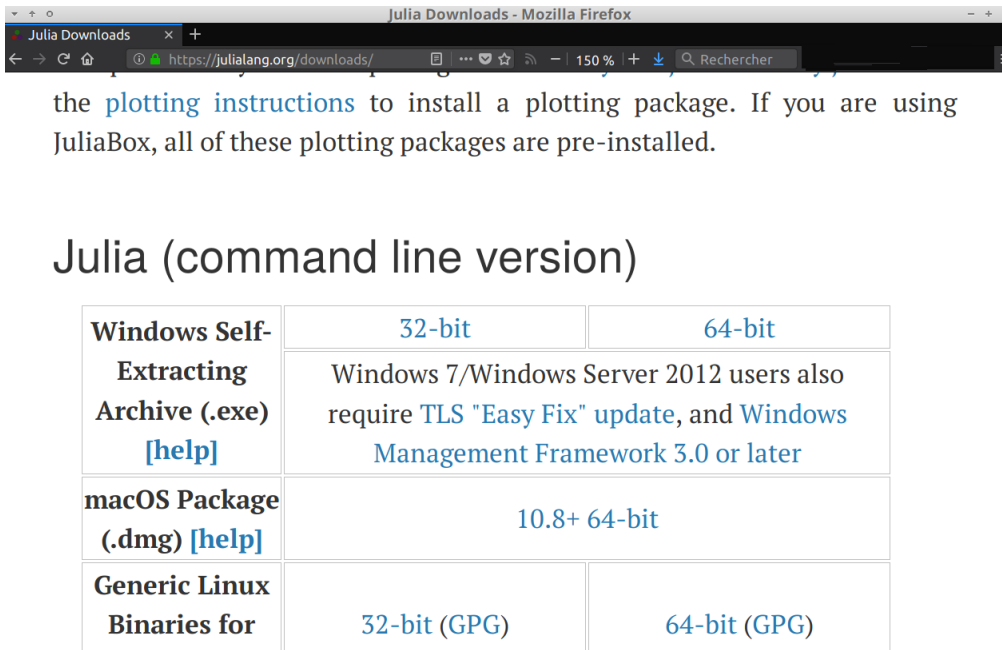
Note¹: [Julia Computing, Inc.](#) (founded 2015 by Julia creators) offer paid licenses ([JuliaPro](#) Enterprise) with professional support.

How to install Julia (1/2)

- You can try online *for free* on JuliaBox.com
- On Linux, Mac OS or Windows:
 - You can use the default installer 
from the website JuliaLang.org/downloads
- Takes about 4 minutes... and it's free !

You also need Python 3 to use Jupyter ✨, I suggest to use  Anaconda.com/download if you don't have Python yet.





How to install Julia (2/2)



the [plotting instructions](#) to install a plotting package. If you are using JuliaBox, all of these plotting packages are pre-installed.

Julia (command line version)

Windows Self-Extracting Archive (.exe) [help]	32-bit	64-bit
	Windows 7/Windows Server 2012 users also require TLS "Easy Fix" update, and Windows Management Framework 3.0 or later	
macOS Package (.dmg) [help]	10.8+ 64-bit	
Generic Linux Binaries for	32-bit (GPG)	64-bit (GPG)

1. Select the binary of your platform 
2. Run the binary  !
3. Wait  ...
4. Done  ! Test with `julia` in a terminal

Different tools to use Julia

- Use `julia` for the command line for short experiments

```
(lun. juin 11 -- 03:06:27)<lilian@jarvis:[~]> {bashv4.4} — Konsole
```

```
$ julia
```

```
(_) | A fresh approach to technical computing  
    | Documentation: https://docs.julialang.org  
    | Type "?help" for help.  
    |  
    | Version 0.6.0 (2017-06-19 13:05 UTC)  
    | Official http://julialang.org/ release  
    | x86_64-pc-linux-gnu  
  
julia> println("Hello world from Julia!")  
Hello world from Julia!  
  
julia>
```

- Use the *Juno* IDE to edit large projects

Demo time 🕒 !

Different tools to use Julia

- Use **Jupyter** notebooks to write or share your experiments (examples: github.com/Naereen/notebooks)

demo — Jupyter Notebook - Chromium

demo (autosaved) Logout

File Edit View Insert Cell Kernel Navigate Widgets Help Snippets Trusted Julia 0.6.0

Enter/Exit RISE Slideshow Table of Contents Toggle spell checking on markdown cells

This is a short demo of Julia

To learn more, check the [Julia documentation!](#)

Hello world

In [14]: `println("Hello from Julia + Jupyter!")`

Hello from Julia + Jupyter!

A simple math example

For $x \in \mathbb{R}^{n,n}$, $iFFT(FFT(x)) = x$?

In [2]: `x = rand(10, 10)`

Out[2]: 10×10 Array{Float64,2}:

0.414411	0.304796	0.239104	0.36513	...	0.0458627	0.801216	0.245571
0.0540293	0.79935	0.455725	0.523839		0.158364	0.971185	0.595249
0.26901	0.276373	0.804781	0.308718		0.562864	0.240393	0.477163
0.550455	0.393096	0.415986	0.934639		0.0959417	0.166348	0.95397
0.050985	0.929405	0.535396	0.85512		0.363232	0.979027	0.13972

Demo time 🕒 !

How to install modules in Julia ?

- Installing is **easy** !

```
julia> Pkg.add("IJulia") # installs IJulia
```

- Updating also!

```
julia> Pkg.update()
```

How to find the module you need ?

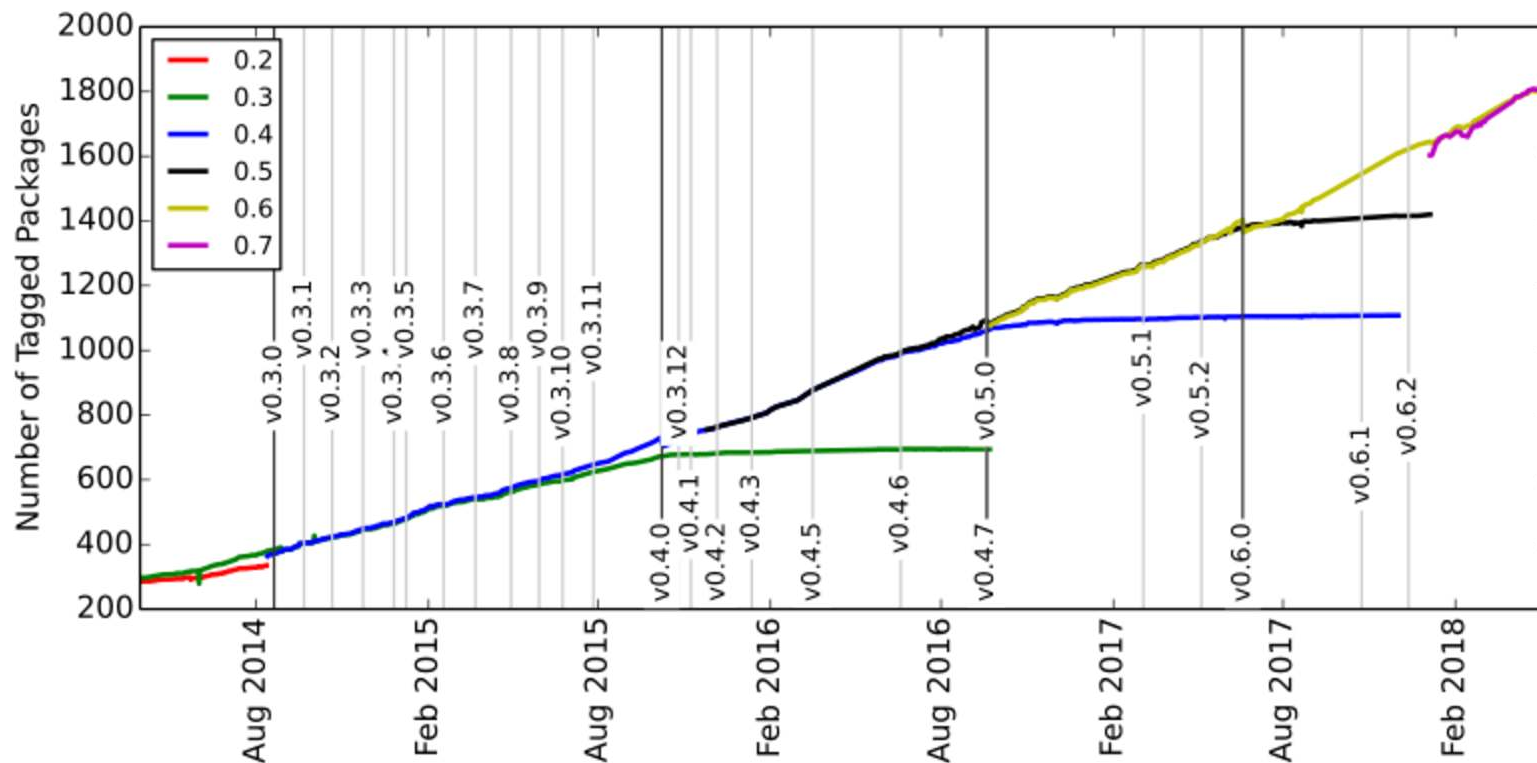
- First... ask your colleagues 😊 !
- Complete list on [Pkg.JuliaLang.org](https://pkg.julialang.org)

Overview of famous Julia modules

- Plotting:
 - `Winston.jl` for easy plotting like MATLAB
 - `PyPlot.jl` interface to Matplotlib (Python)
- The `JuliaDiffEq` collection for **differential equations**
- The `JuliaOpt` collection for **optimization**
- The `JuliaStats` collection for **statistics**
- And many more!

Find more specific packages on [GitHub.com/svaksha/Julia.jl](https://github.com/svaksha/Julia.jl)

Many packages, and a quickly growing community



Julia is still in development, in version v0.6 but version 1.0 is planned soon!

2. Main differences in syntax between Julia and MATLAB

Ref: [CheatSheets.QuanteCon.org](https://cheatsheets.quantecon.org)

2. Main differences in syntax between Julia and MATLAB

Ref: [CheatSheets.QuanteCon.org](https://cheatsheets.quantecon.org)

	Julia	MATLAB
File ext.	<code>.jl</code>	<code>.m</code>
Comment	<code># blabla...</code>	<code>% blabla...</code>
Indexing	<code>a[1]</code> to <code>a[end]</code>	<code>a(1)</code> to <code>a(end)</code>
Slicing	<code>a[1:100]</code> (view)	<code>a(1:100)</code> (⚠ copy)
Operations	Linear algebra by default	Linear algebra by default
Block	Use <code>end</code> to close all blocks	Use <code>endif</code> <code>endfor</code> etc

	Julia	MATLAB
Help	?func	help func
And	a & b	a && b
Or	a b	a b
Datatype	Array of <i>any</i> type	multi-dim doubles array
Array	[1 2; 3 4]	[1 2; 3 4]
Size	size(a)	size(a)
Nb Dim	ndims(a)	ndims(a)
Last	a[end]	a(end)

	Julia	MATLAB
Tranpose	<code>a.'</code>	<code>a.'</code>
Conj. transpose	<code>a'</code>	<code>a'</code>
Matrix x	<code>a * b</code>	<code>a * b</code>
Element-wise x	<code>a .* b</code>	<code>a .* b</code>
Element-wise /	<code>a ./ b</code>	<code>a ./ b</code>
Element-wise ^	<code>a ^ 3</code>	<code>a .^ 3</code>
Zeros	<code>zeros(2, 3, 5)</code>	<code>zeros(2, 3, 5)</code>
Ones	<code>ones(2, 3, 5)</code>	<code>ones(2, 3, 5)</code>
Identity	<code>eye(10)</code>	<code>eye(10)</code>
Range	<code>range(0, 100, 2)</code> or <code>1:2:100</code>	<code>1:2:100</code>

	Julia	MATLAB
Maximum	<code>max(a)</code>	<code>max(max(a))</code> ?
Random matrix	<code>rand(3, 4)</code>	<code>rand(3, 4)</code>
L^2 Norm	<code>norm(v)</code>	<code>norm(v)</code>
Inverse	<code>inv(a)</code>	<code>inv(a)</code>
Solve syst.	<code>a \ b</code>	<code>a \ b</code>
Eigen vals	<code>V, D = eig(a)</code>	<code>[V,D]=eig(a)</code>
FFT/IFFT	<code>fft(a)</code> , <code>ifft(a)</code>	<code>fft(a)</code> , <code>ifft(a)</code>

Very close to MATLAB for linear algebra!

3. Scientific problems solved with Julia

Just to give examples of syntax and modules

1. 1D numerical integration and plot
2. Solving a 2nd order Ordinary Differential Equation

3.1. 1D numerical integration and plot

Exercise: evaluate and plot this function on $[-1, 1]$:

$$\text{Ei}(x) := \int_{-x}^{\infty} \frac{e^u}{u} du$$

How to?

Use packages and everything is easy!

- `QuadGK.jl` for integration
- `Winston.jl` for 2D plotting

```

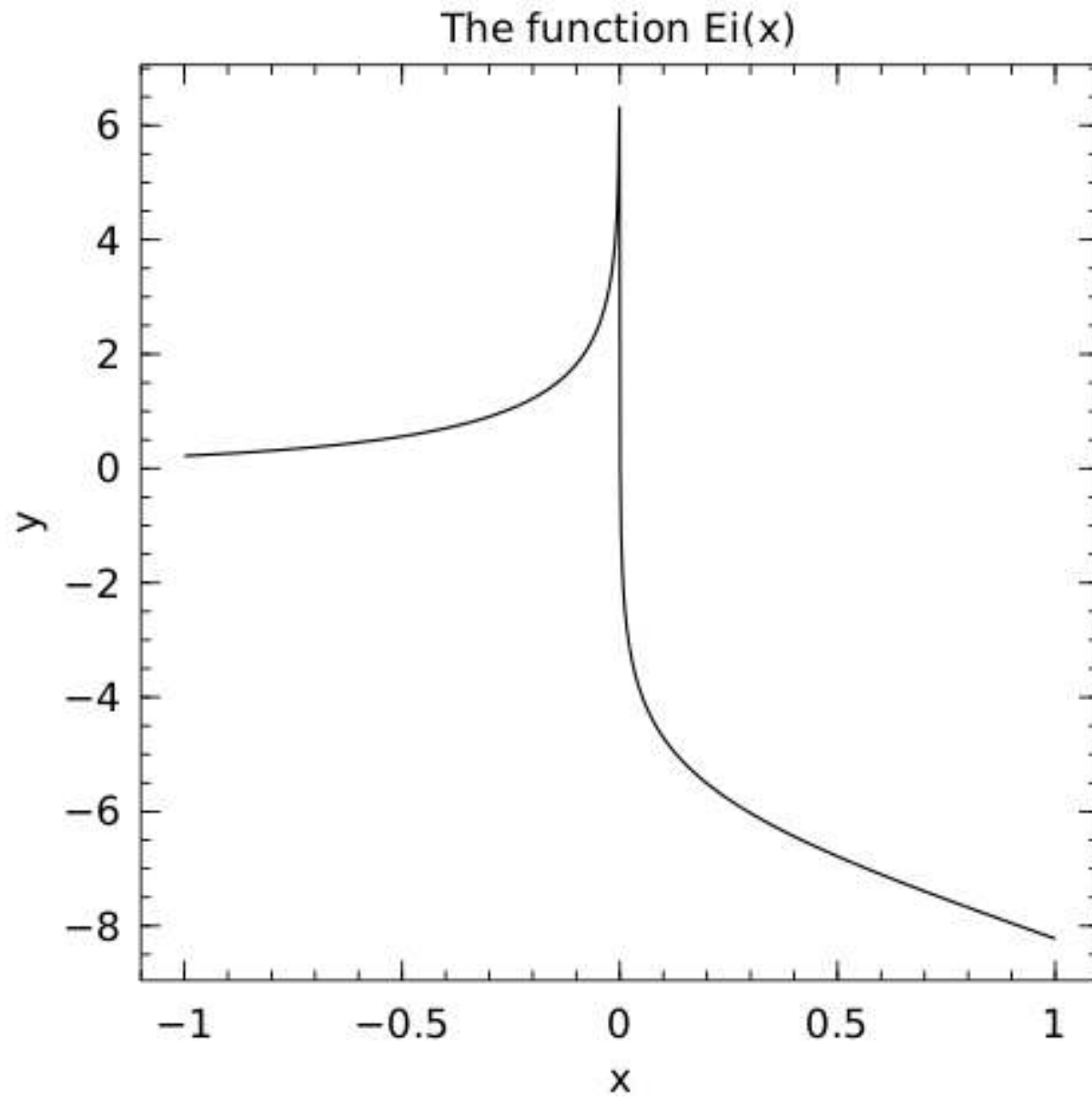
using QuadGK

function Ei(x, minfloat=1e-3, maxfloat=100)
    f = t -> exp(-t) / t  # inline function
    if x > 0
        return quadgk(f, -x, -minfloat)[1]
        + quadgk(f, minfloat, maxfloat)[1]
    else
        return quadgk(f, -x, maxfloat)[1]
    end
end

X = linspace(-1, 1, 1000)  # 1000 points
Y = [ Ei(x) for x in X ]  # Python-like syntax!

using Winston
plot(X, Y)
title("The function Ei(x)")
xlabel("x"); ylabel("y")
savefig("figures/Ei_integral.png")

```



3.2. Solving a 2nd order ODE

Goal: solve and plot the differential equation of a pendulum:

$$\theta''(t) + b \theta'(t) + c \sin(\theta(t)) = 0$$

For $b = 1/4, c = 5, \theta(0) = \pi - 0.1, \theta'(0) = 0, t \in [0, 10]$

How to?

Use packages!

- `DifferentialEquations.jl` function for ODE integration
- `Winston.jl` for 2D plotting

```
using DifferentialEquations
```

```
b, c = 0.25, 5.0  
y0 = [pi - 0.1, 0]
```

```
# macro magic!
```

```
pend2 = @code_def Pendulum begin
```

```
    dθ = ω # ← yes, this is UTF8, θ and ω in text
```

```
    dω = (-b * ω) - (c * sin(θ))
```

```
end
```

```
prob = ODEProblem(pend, y0, (0.0, 10.0))
```

```
sol = solve(prob) # ↑ solve on interval [0,10]
```

```
t, y = sol.t, hcat(sol.u...)'
```

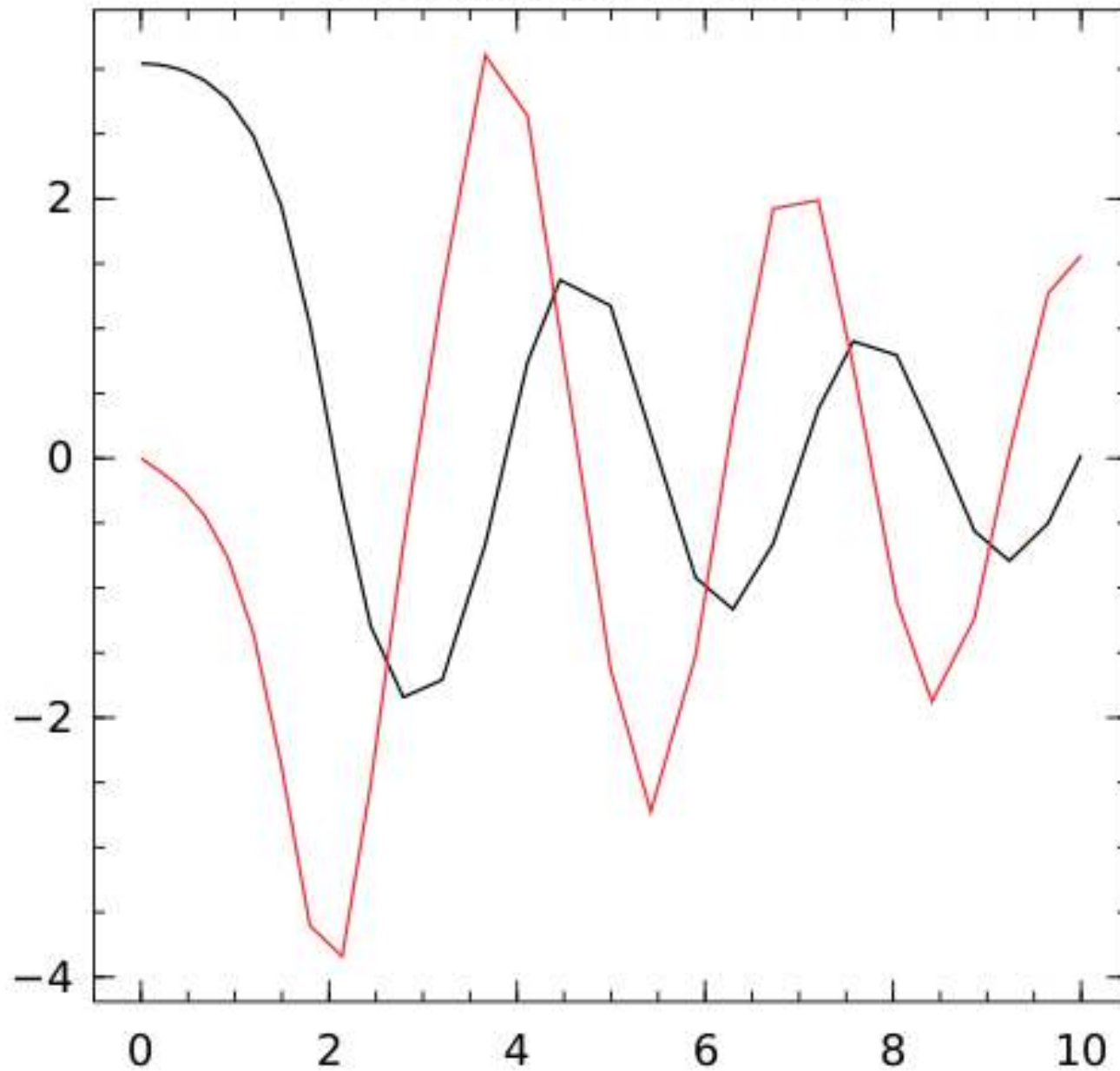
```
using Winston
```

```
plot(t, y[:, 1], t, y[:, 2])
```

```
title("2D Differential Equation")
```

```
savefig("figures/Pendulum_solution.png")
```

2D Differential Equation



Examples

1. **Iterative computation:** signal filtering
2. **Optimization:** robust regression on RADAR data

Ex. 1: Iterative computation

Objective:

- show the efficiency of Julia's Just-in-Time (JIT) compilation
- but also its fragility...

Note: you can find companion notebooks on [GitHub](#)

Iterative computation: signal filtering

The classical saying:

« *Vectorized code often runs much faster than the corresponding code containing loops.* » (cf. [MATLAB doc](#))

does not hold for Julia, because of its **Just-in-Time compiler**.

Example of a computation that cannot be vectorized

Smoothing of a signal $\{u_k\}_{k \in \mathbb{N}}$:

$$y_k = ay_{k-1} + (1 - a)u_k, \quad k \in \mathbb{N}^+$$

Parameter a tunes the smoothing (none: $a = 0$, strong $a \rightarrow 1^-$).

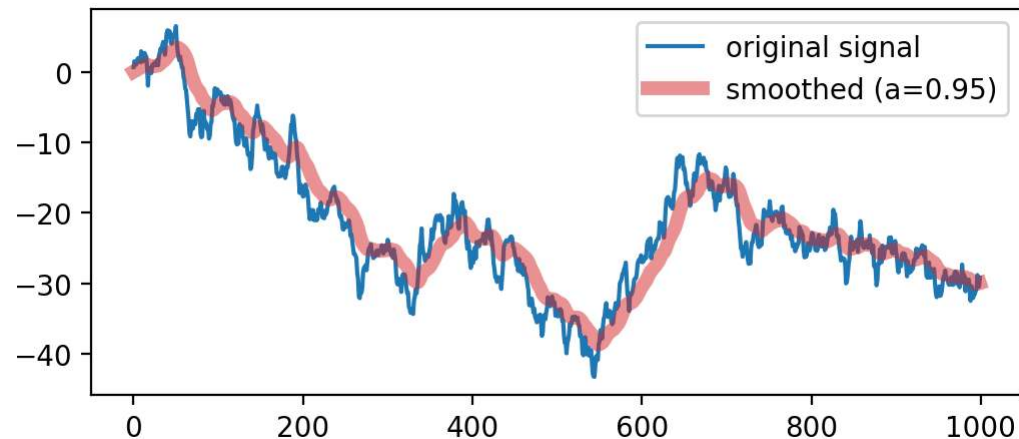
 Iteration (`for` loop) **cannot** be avoided.

Signal filtering in Julia 🙌

```
function smooth(u, a)
    y = zeros(u)

    y[1] = (1-a)*u[1]
    for k=2:length(u) # this loop is NOT slow!
        y[k] = a*y[k-1] + (1-a)*u[k]
    end

    return y
end
```



Performance of the signal filter

Implementation	Time for 10 Mpts	notes
Julia	50 – 70 ms	Fast! Easy! 🙌
Octave native	88000 ms	slow!! 🐌
Python native	4400 ms	slow! 🐌
SciPy's <code>lfilter</code>	70 ms	many lines of C
Python + <code>@numba.jit</code>	50 ms	since 2012

```
@numba.jit # <- factor ×100 speed-up!
def smooth_jit(u, a):
    y = np.zeros_like(u)
    y[0] = (1-a)*u[0]
    for k in range(1, len(u)):
        y[k] = a*y[k-1] + (1-a)*u[k]
    return y
```


Conclusion on the performance

For this simple iterative computation:

- Julia performs very well, much better than native Python
- but it's possible to get the same with fresh Python tools ([Numba](#))
- more realistic examples are needed

Fragility of Julia's JIT Compilation

The efficiency of the compiled code relies on **type inference**.

```
function smooth1(u, a)
    y = 0
    for k=1:length(u)
        y = a*y + (1-a)*u[k]
    end
    return y
end
```

```
function smooth2(u, a)
    y = 0.0    # <- difference is here!
    for k=1:length(u)
        y = a*y + (1-a)*u[k]
    end
    return y
end
```

An order of magnitude difference 🐌 vs 🏃

```
julia> @time smooth1(u, 0.9);  
0.212018 seconds (30.00 M allocations: 457.764 MiB ...)
```

```
julia> @time smooth2(u, 0.9);  
0.024883 seconds (5 allocations: 176 bytes)
```

Fortunately, Julia gives a good diagnosis tool 🛠️

```
julia> @code_warntype smooth1(u, 0.9);  
... # ↓ we spot a detail  
y::Union{Float64, Int64}  
...
```

`y` is **either** `Float64` or `Int64` when it should be just `Float64`.

Cause: initialization `y=0` vs. `y=0.0` !

Ex. 2: Optimization in Julia

Objective: demonstrate JuMP, a Modeling Language for Optimization in Julia.

- Some researchers migrate to Julia just for this!
- I use JuMP for **my research** (energy management)

Note: you can find companion notebooks on [GitHub](#)

Optimization problem example

Example problem: identifying the sea clutter in Weather Radar data.

- is a **robust regression** problem
 - \hookrightarrow is an optimization problem!

An « IETR-colored » example, inspired by:

- Radar data+photo: P.-J. Trombe *et al.*, « Weather radars – the new eyes for offshore wind farms?, » *Wind Energy*, 2014.
- Regression methods: S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004. (Example 6.2).

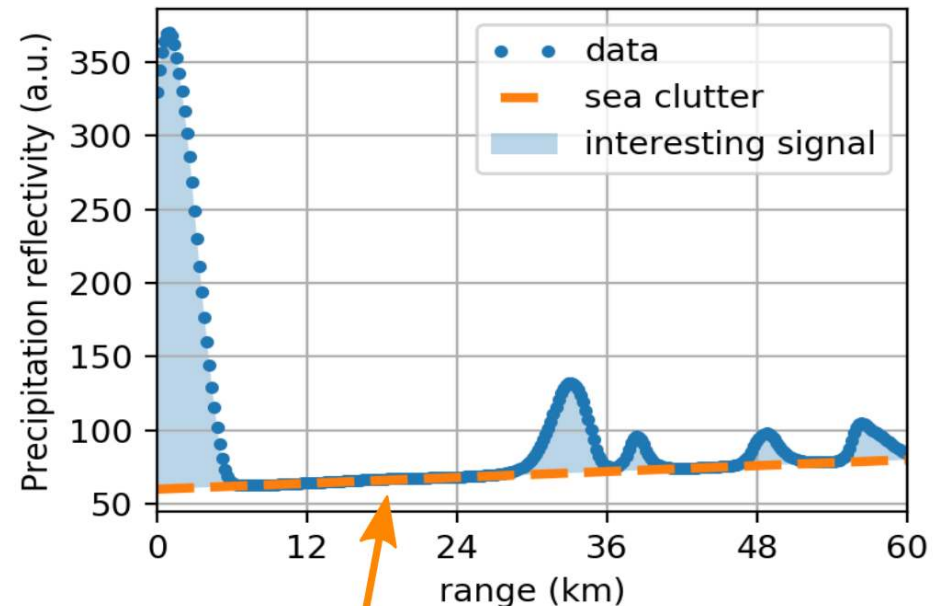
Weather radar: the problem of sea clutter

Offshore wind farm (Horns Rev, Denmark)



Weather radar: X-band (9.41 GHz), 60 km range

Reflection data, for one azimuth



Objective: fit this trend (to substract it)

Given n data points (x_i, y_i) , fit a linear trend:

$$\hat{y} = a.x + b$$

An **optimization problem** with two parameters: a (slope), b (intercept)

Regression as an optimization problem

The parameters for the trend (a, b) should minimize a criterion J which penalizes the residuals $r_i = y_i - \hat{y} = y_i - a.x + b$:

$$J(a, b) = \sum_i \phi(r_i)$$

where ϕ is the *penalty function*, to be chosen:

- $\phi(r) = r^2$: quadratic deviation \rightarrow least squares regression
- $\phi(r) = |r|$: absolute value deviation
- $\phi(r) = h(r)$: **Huber loss**
- ...

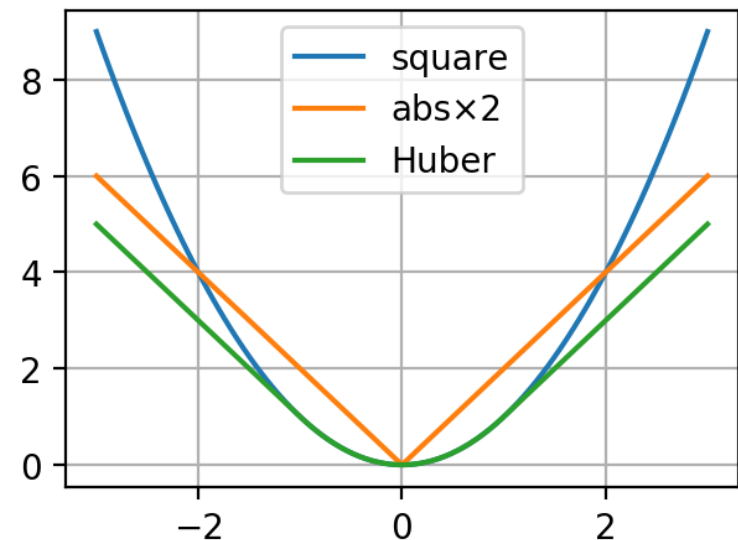
Choice of penalty function

The choice of the loss function influences:

- the optimization result (fit quality)
 - *e.g.*, in the presence of outliers
- the properties of optimization problem: convexity, smoothness

Properties of each function

- quadratic: convex, smooth, heavy weight for strong deviations
- absolute value: convex, not smooth
- Huber: a mix of the two



How to solve the regression problem?

Option 1: a big bag of tools

A specific package for each type of regression:

- « least square toolbox » (\rightarrow [MultivariateStats.jl](#))
- « least absolute value toolbox » (\rightarrow [quantile regression](#))
- « Huber toolbox » (*i.e.*, robust regression \rightarrow ??)
- ...

Option 2: the « One Tool »



\implies a Modeling Language for Optimization

- more freedom to explore variants of the problem

Modeling Languages for Optimization

Purpose: make it easy to specify and solve optimization problems without expert knowledge.

JuMP: optimization modeling in Julia

- The [JuMP](#) package offers a domain-specific modeling language for mathematical optimization.

JuMP interfaces with many optimization solvers: open-source (Ipopt, GLPK, Clp, ECOS...) and commercial (CPLEX, Gurobi, MOSEK...).

- Other Modeling Languages for Optimization:
 - Standalone software: AMPL, GAMS
 - Matlab: YALMIP ([previous seminar](#)), CVX
 - Python: Pyomo, PuLP, CVXPY

Claim: JuMP is fast, thanks to Julia's [metaprogramming](#) capabilities (generation of Julia code within Julia code).



Regression with JuMP — common part

- Given `x` and `y` the 300 data points:

```
m = Model(solver = ECOSolver())  
  
@variable(m, a)  
@variable(m, b)  
  
res = a*x .- y + b
```

`res` (« residuals ») is an Array of 300 elements of type `JuMP.GenericAffExpr{Float64, JuMP.Variable}`, *i.e.*, a semi-symbolic affine expression.

- Now, we need to specify the penalty on those residuals.

Regression choice: least squares regression

$$\min \sum_i r_i^2$$

Reformulated as a [Second-Order Cone Program](#) (SOCP):

$$\min j, \quad \text{such that } \|r\|_2 \leq j$$

```
@variable(m, j)
@constraint(m, norm(res) <= j)
@objective(m, Min, j)
```

(SOCP problem \implies [ECOS](#) solver)

Regression choice: least absolute deviation

$$\min \sum_i |r_i|$$

Reformulated as a **Linear Program** (LP)

$$\min \sum_i t_i, \quad \text{such that} \quad -t_i \leq r_i \leq t_i$$

```
@variable(m, t[1:n])  
@constraint(m, res .<= t)  
@constraint(m, res .>= -t)  
@objective(m, Min, sum(t))
```

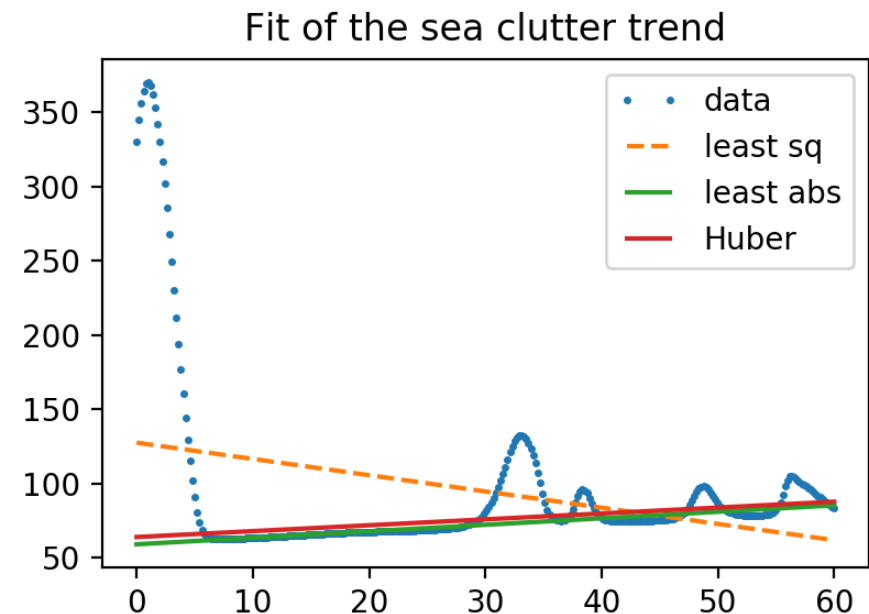
Solve! ⚙️

```
julia> solve(m)
[solver blabla... ⌚ ]
:Optimal # hopefully
```

```
julia> getvalue(a), getvalue(b)
(-1.094, 127.52) # for least squares
```

Observations:

- least abs. val., Huber ✓
- least squares ✗



JuMP: summary

A modeling language for optimization, *within Julia*:

- gives access to all classical optimization solvers
- very fast (claim)
- gives freedom to explore many variations of an optimization problem (fast prototyping)

 More on optimization with Julia:

- [JuliaOpt](#): host organization of JuMP
- [Optim.jl](#): implementation of classics in Julia (*e.g.*, Nelder-Mead)
- [JuliaDiff](#): Automatic Differentiation to compute gradients, thanks to Julia's strong capability for code introspection

Conclusion (1/2)

Sum-up

- I hope you got a good introduction to Julia 🙌
- It's not hard to migrate from MATLAB to Julia
- Good start:

docs.julialang.org/en/stable/manual/getting-started

- Julia is fast!
- Free and open source!
- Can be very efficient for some applications!

Conclusion (2/2)

Thanks for joining 🙌 !

Your mission, if you accept it... ✨

1. 🧑🎓 *Padawan level*: Train yourself a little bit on Julia
↳ [JuliaBox.com](https://julia-box.com) ? Or install it on your laptop!
And read [introduction in the Julia manual](#)!
2. 🎓 *Jedi level*: Try to solve a numerical system, from your research or teaching, **in Julia instead of MATLAB**
3. 🗡️ *Master level*: From now on, try to use open-source & free tools for your research (Julia, Python and others)... 💰

Thank you !!