

# « Julia, my new optimization friend »

- **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 [25 min]

1. What is Julia [3 min]
2. Comparison with MATLAB [3 min]
3. Examples of problems solved Julia [5 min]
4. Longer example on optimization with JuMP [10min]
5. Links for more information ? [2 min]

# 1. What is Julia ?

- Developed and popular from the **last 7 years**
- Open-source and free programming language (MIT license)
- Interpreted *and* compiled, very efficient
- But easy syntax, dynamic typing, inline documentation etc
- Multi-platform, imperative
- MATLAB-like syntax for linear algebra etc
- Designed and acknowledged as *simple to learn and use*
- Easy to run your code in parallel (multi-core & cluster)
- Used worldwide: research, data science, finance etc...

# Ressources

- Website: [JuliaLang.org](https://JuliaLang.org) for the language & [Pkg.JuliaLang.org](https://Pkg.JuliaLang.org) for packages
- Documentation : [docs.JuliaLang.org](https://docs.JuliaLang.org)



# Comparison with MATLAB

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

# Comparison with MATLAB


	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

	Julia	MATLAB
Usage	Generic, worldwide 🌍	Research in academia and industry
Fame	Young but starts to be known	Old and known, in decline
Support?	Community <sup>1</sup> (StackOverflow, mailing lists etc).	By MathWorks
Documentation	OK and growing, inline/online	OK, inline/online

**Note<sup>1</sup>:** JuliaPro offer paid licenses, if professional support is needed.

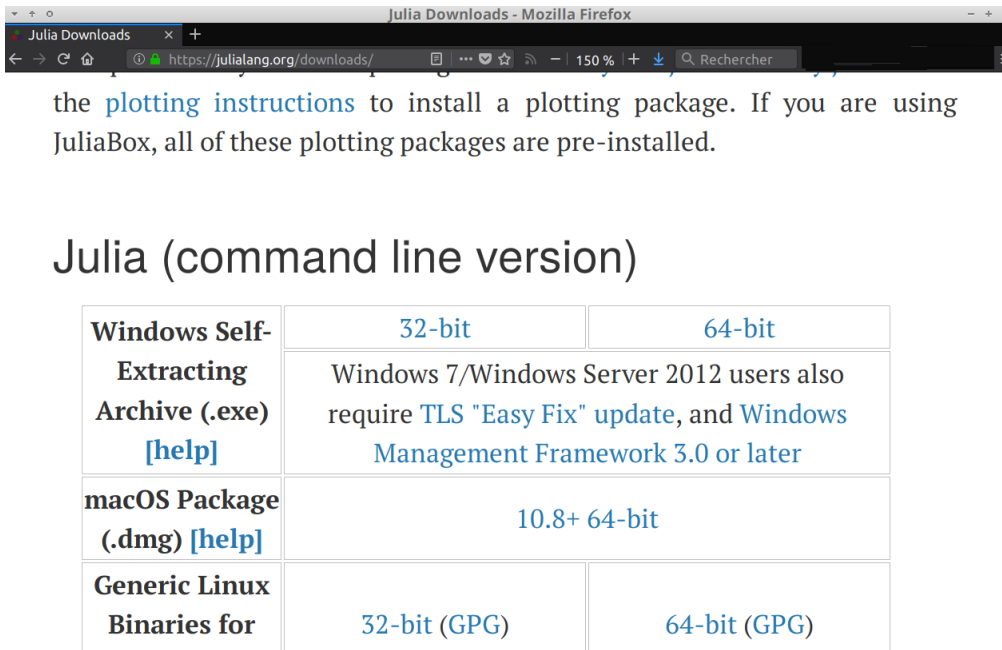
# How to install Julia (1/2)

- You can try online *for free* on [JuliaBox.com](https://JuliaBox.com)
- On Linux, Mac OS or Windows:
  - You can use the default installer   
from the website [julialang.org/downloads](https://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](https://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) <a href="#">[help]</a>	32-bit	64-bit
	Windows 7/Windows Server 2012 users also require <a href="#">TLS "Easy Fix" update</a> , and <a href="#">Windows Management Framework 3.0</a> or later	
macOS Package (.dmg) <a href="#">[help]</a>	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

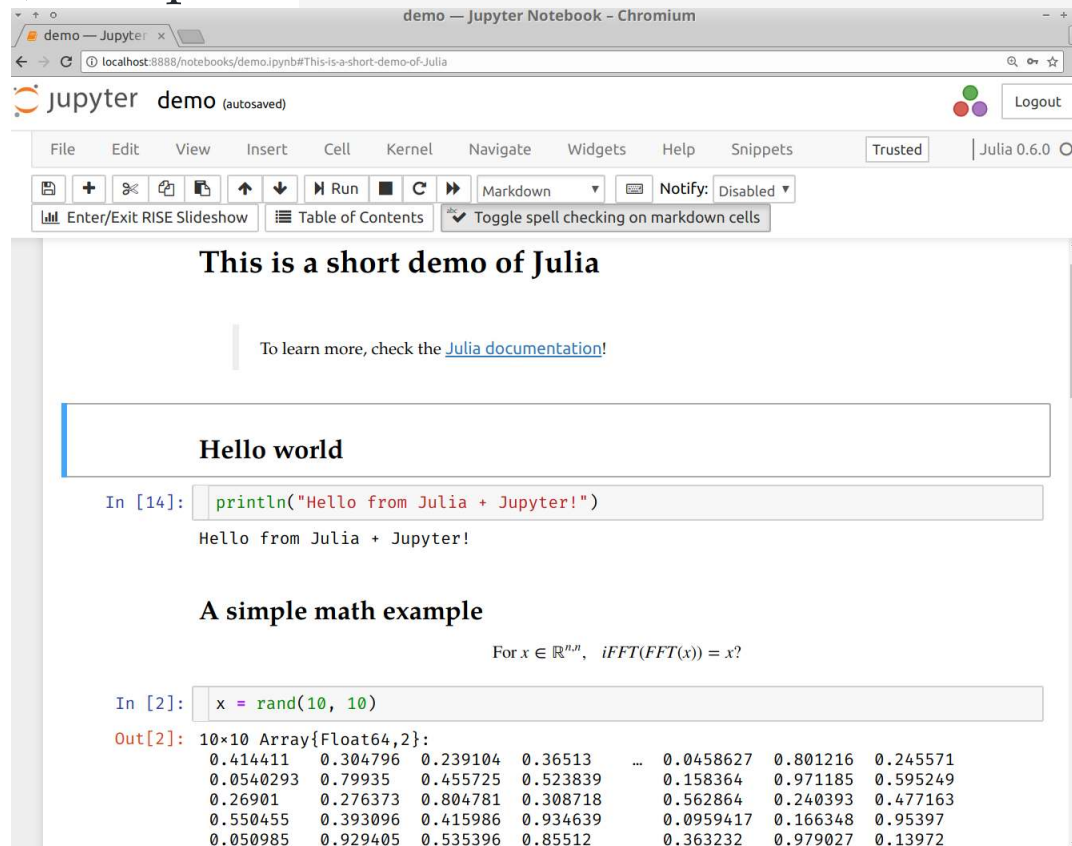
[illegible]

- 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](https://github.com/Naereen/notebooks) )



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](http://pkg.julialang.org)

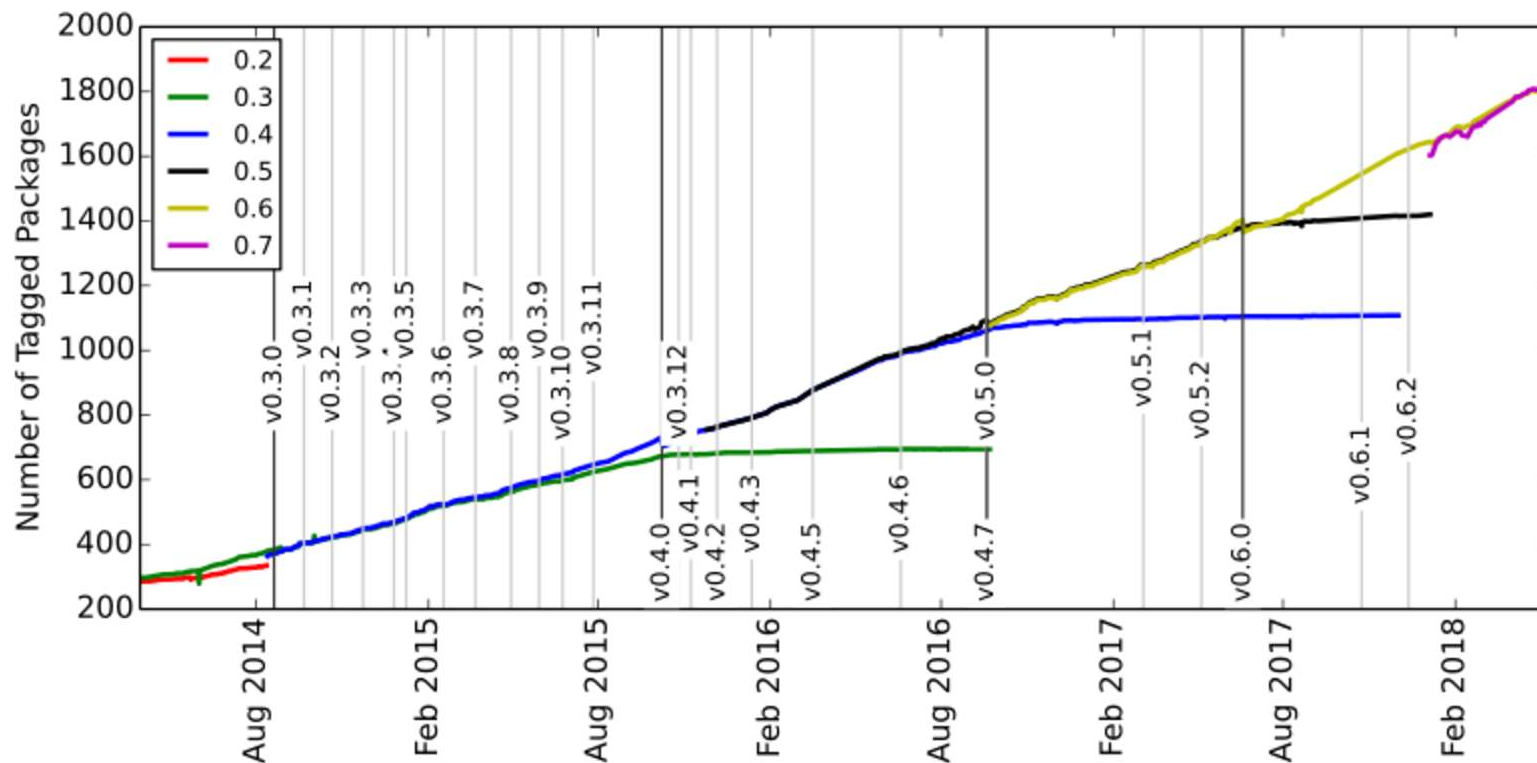


# Overview of famous Julia modules

- `Winston.jl` for easy plotting like MATLAB
- 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)

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



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

	<b>Julia</b>	<b>MATLAB</b>
<b>Maximum</b>	<code>max(a)</code>	<code>max(max(a))</code> ?
<b>Random matrix</b>	<code>rand(3, 4)</code>	<code>rand(3, 4)</code>
<b>L2 Norm</b>	<code>norm(v)</code>	<code>norm(v)</code>
<b>Inverse</b>	<code>inv(a)</code>	<code>inv(a)</code>
<b>Solve syst.</b>	<code>a \ b</code>	<code>a \ b</code>
<b>Eigen vals</b>	<code>V, D = eig(a)</code>	<code>[V,D]=eig(a)</code>
<b>FFT/IFFT</b>	<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 2<sup>nd</sup> 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

```

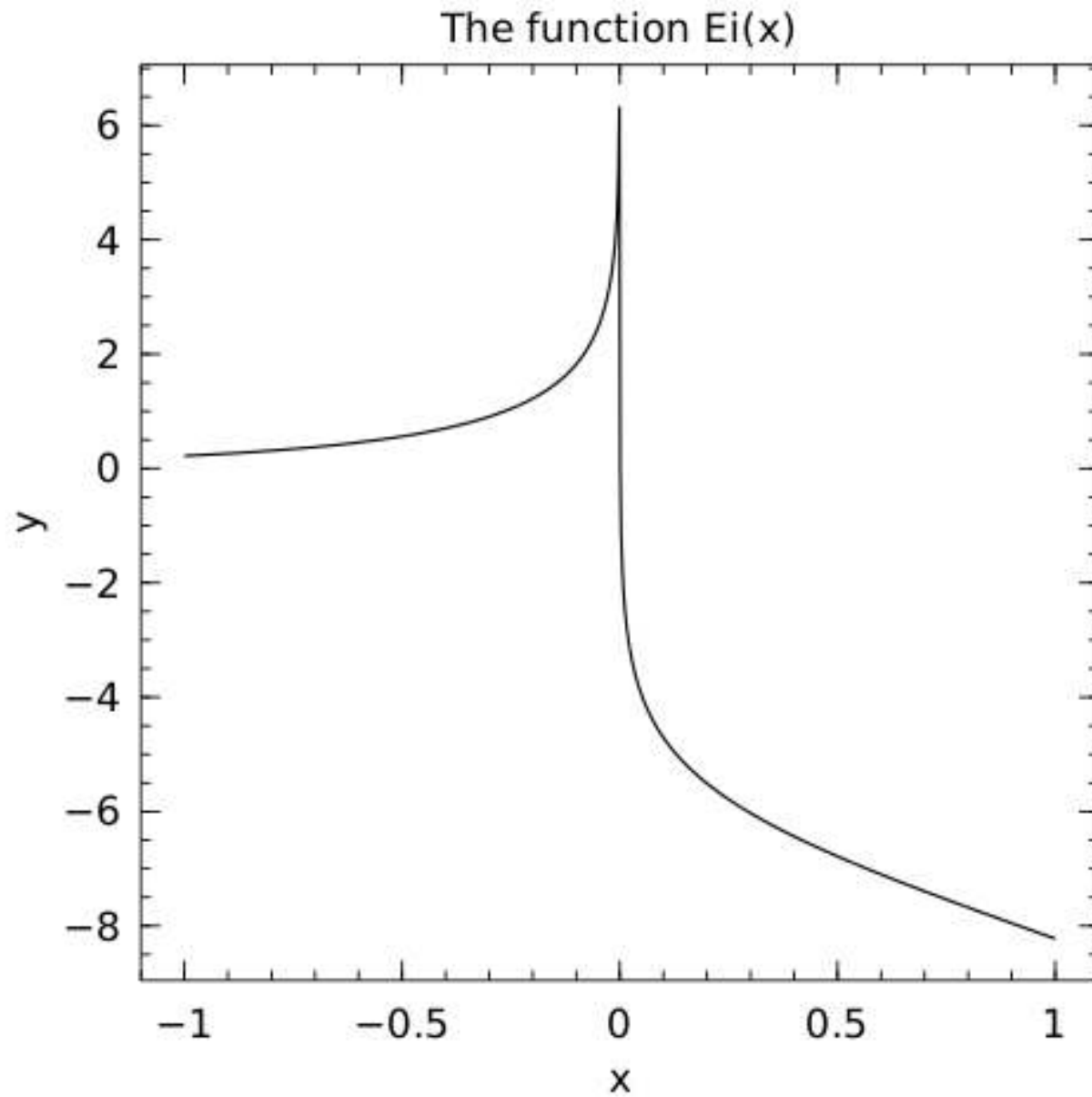
using QuadGK

function Ei(x, minfloat=1e-3, maxfloat=100)
    f = t → exp(-t) / t  # inline function, with '- >'
    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 ]

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

```



## 3.2. Solving a 2<sup>nd</sup> 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

# macro magic!
pend2 = @ode_def Pendulum begin
    dθ = ω # ← yes, this is UTF8
    dω = (-b * ω) - (c * sin(θ))
end

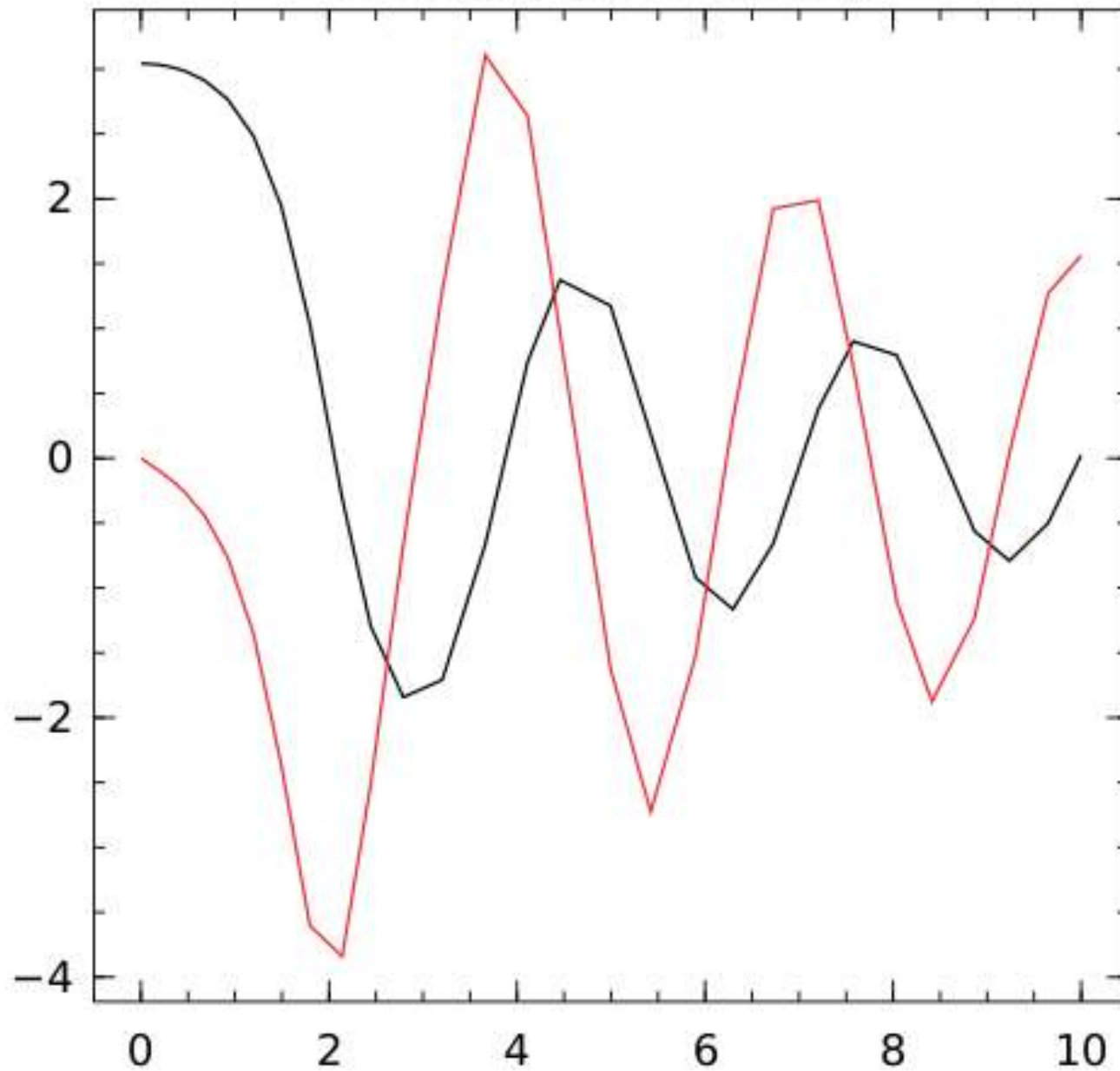
prob = ODEProblem(pend, y0, (0.0, 10.0))
sol = solve(prob) # ↑ solve on interval [0,10]
t, y = sol.t, hcat(sol.u ... )'

using Winston
plot(t, y[:, 1], t, y[:, 2])
title("2D Differential Equation")
savefig("figures/Pendulum_solution.png")

```



## 2D Differential Equation



# 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](https://docs.julialang.org/en/stable/manual/getting-started)

# 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)... 💰