Learning Scientific computing with julia

Semana de la Ciencia 2020
Pedro Jiménez & Mario Merino

uc3m

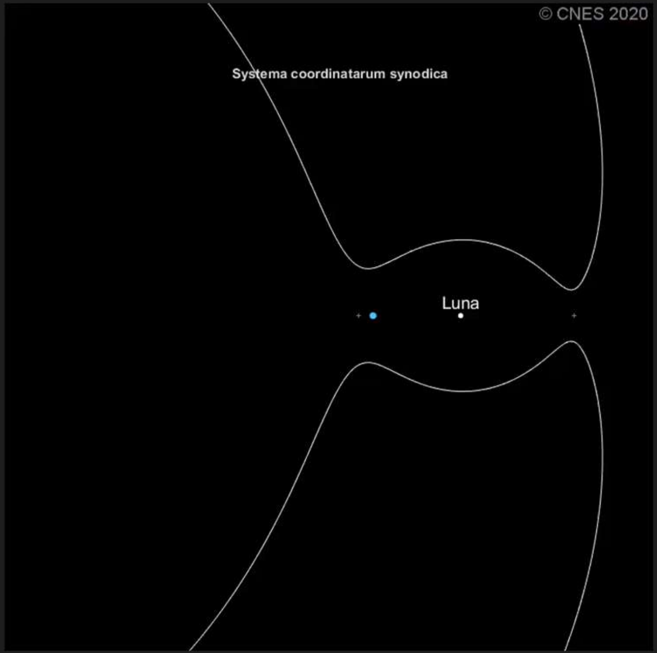
semana de la ciencia y la innovación
Un planeta, muchos mundos 2020

Contents

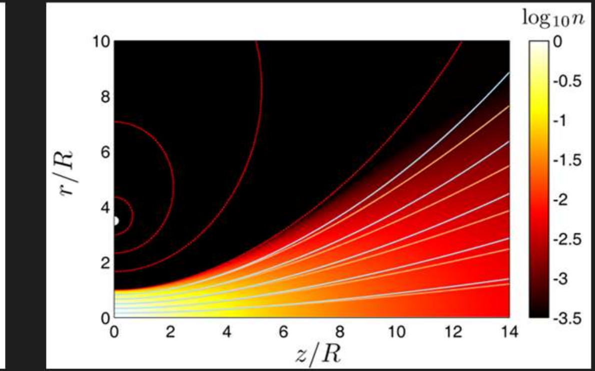
- Scientific computing languages and Julia
- Installing Julia, VS Code, and the Julia extension
- Basics of VS Code and the Julia REPL
- First example: Fibonacci's numbers
- Second example: Is it a multiple of 7?
- Third example: Locate that ship!

- Packages: Plots
- Fourth example: Now, show me the ship!
- Final project: The Lorenz attractor
- Other capabilities of Julia
- List of other recommended packages & where to find help

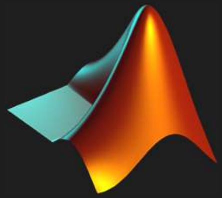
Scientific computing languages



- Modeling, simulating, and visualizing real life is a big deal in science and engineering!



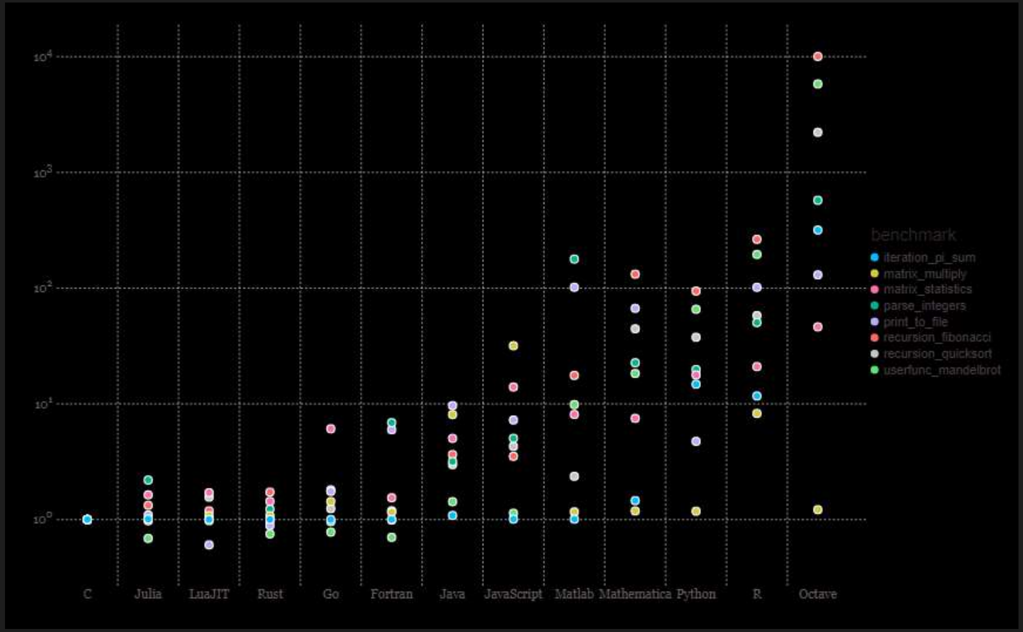
- Programming languages are divided into compiled and interpreted



julia

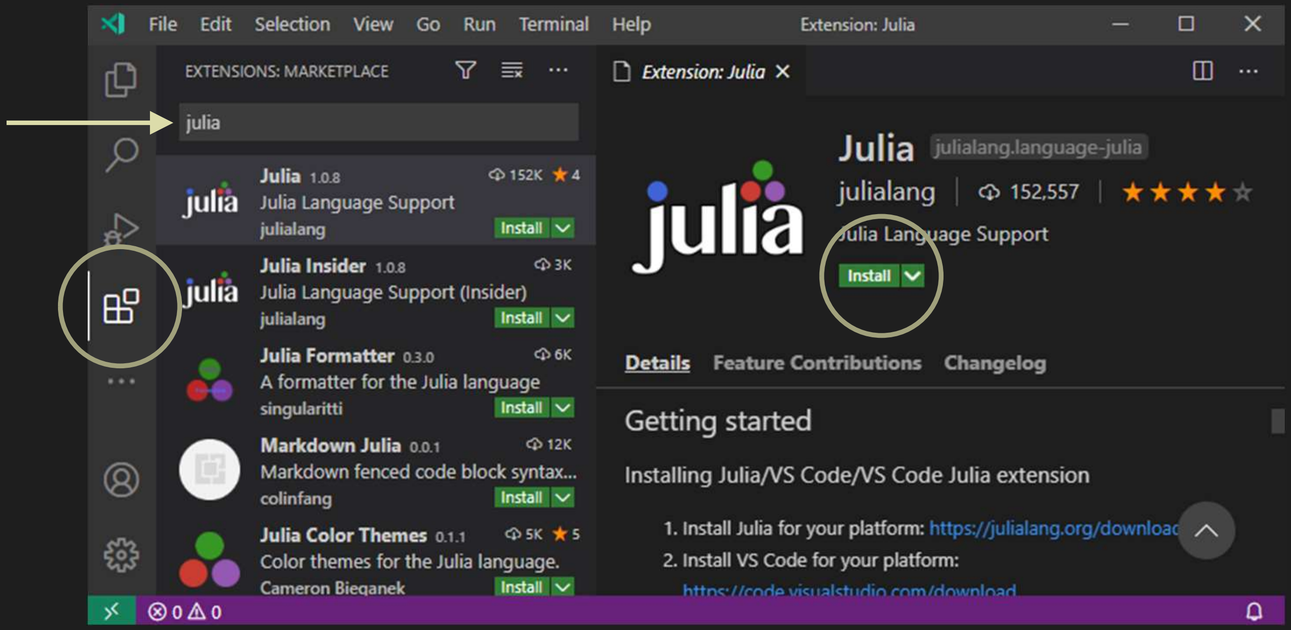
- Modern, dynamically-typed, great for fast prototyping and interaction
- Just in time compilation for high performance
- Designed for scientific computing, not as an afterthought
- Convenient syntax for maths and physics, similar to Matlab

```julia
function myfactorial(n)
    fact = 1
    for m = 1:n
        fact = fact * m
    end
    return fact
end
```



Setting things up:

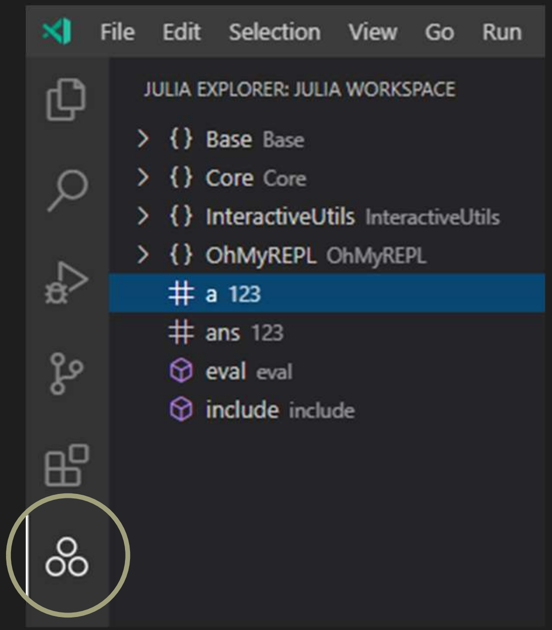
- Install Julia (https://julialang.org/)
- Install VS Code (https://code.visualstudio.com/)
- Install Julia extension in VS Code:



Getting started

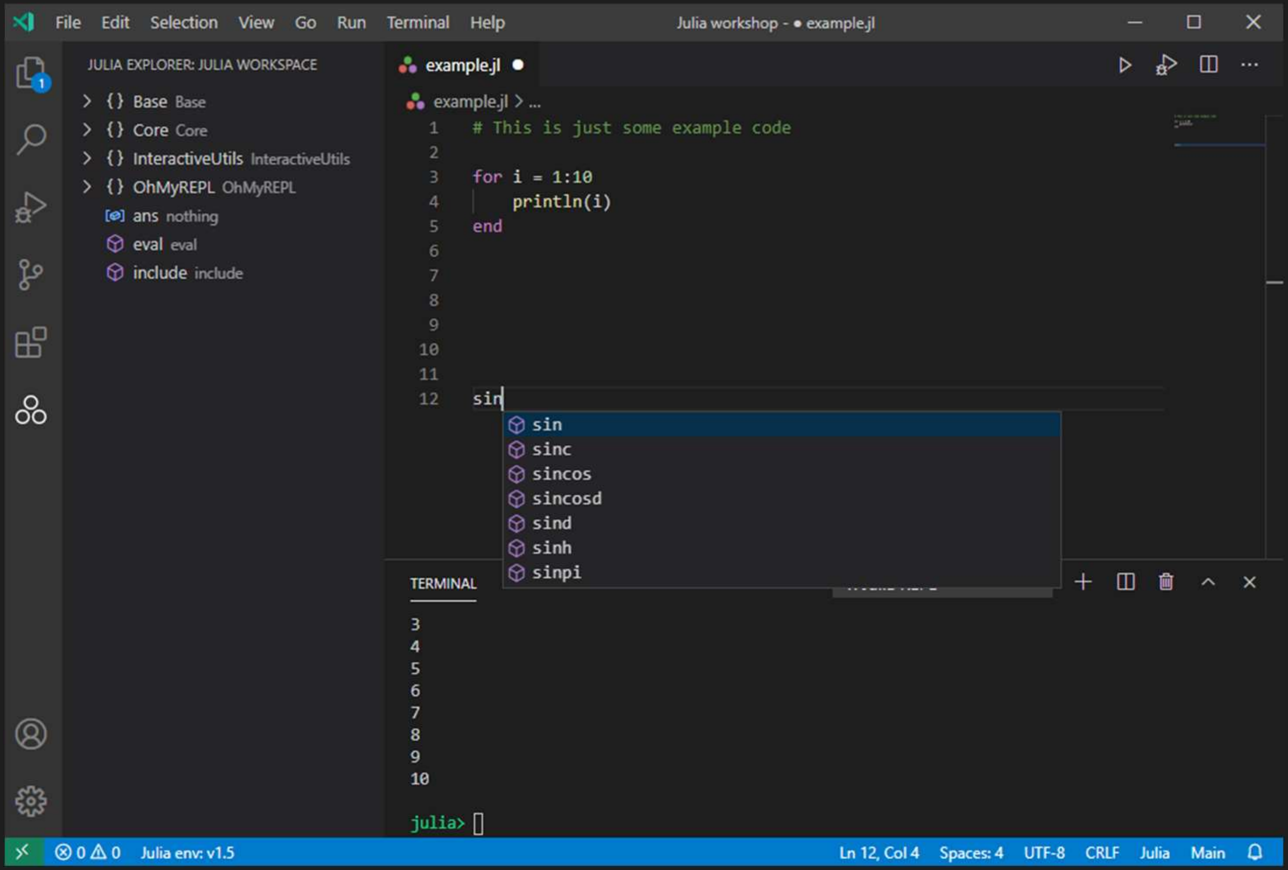
Installing Julia/VS Code/VS Code Julia extension

1. Install Julia for your platform: https://julialang.org/download
2. Install VS Code for your platform: https://code.visualstudio.com/download



VS code

- Arguably, the best code editor currently available
- Fast and versatile
- Integrated terminal, code runner, debugger, git
- Powerful extension system
- Hit CTRL+SHIFT+P to access all commands
- Use TAB to autocomplete
- Learn basic keyboard shortcuts
- Let’s explore it!

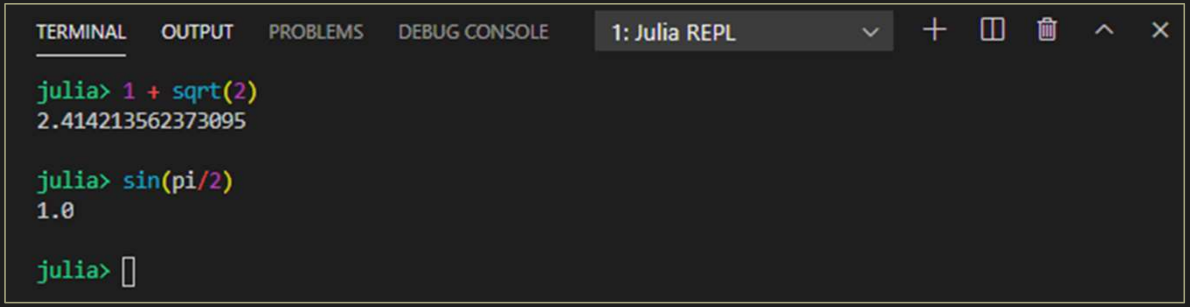


```
# This is just some example code

for i = 1:10
    println(i)
end






sin
```

```
3
4
5
6
7
8
9
10

julia> 
```

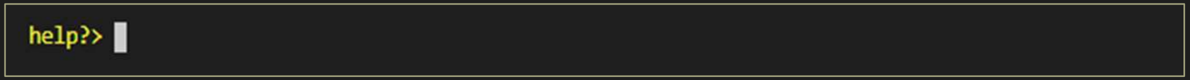
The julia REPL

- The most well-designed terminal we have ever seen!



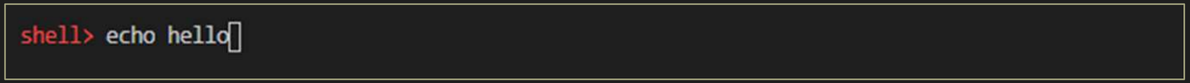
```
julia> 1 + sqrt(2)
2.414213562373095

julia> sin(pi/2)
1.0

julia> 
```

- Get inline help on any function by hitting “?” Try it!



```
help?> 
```

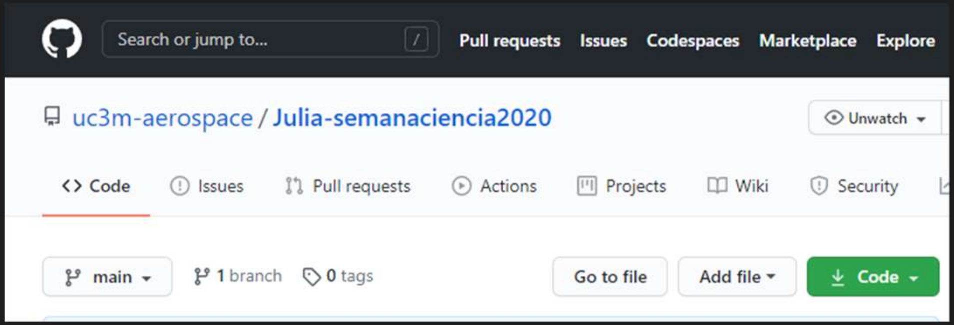
- Shell escape (to run OS commands): hit “;”



```
shell> echo hello
```

Code templates

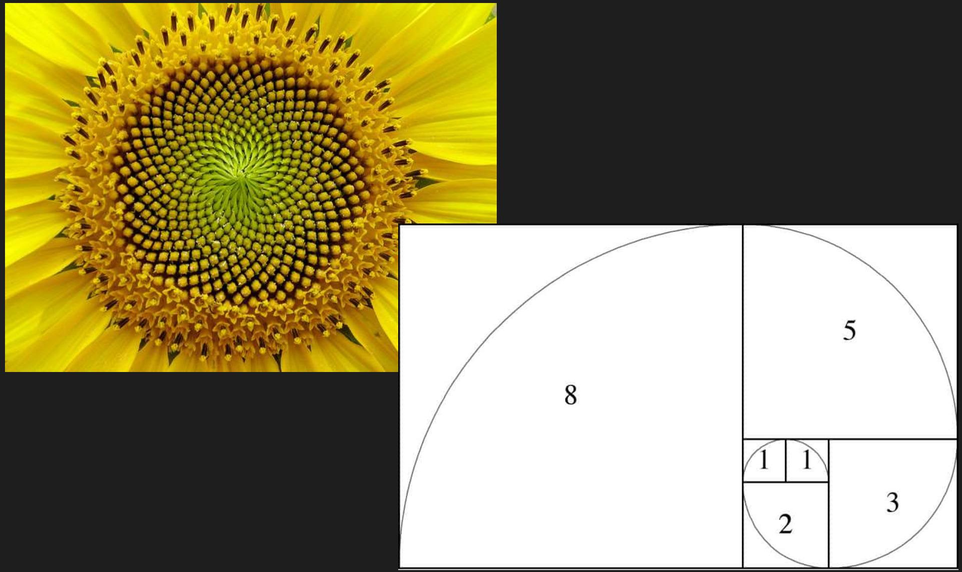
- The templates and the reference solutions for the activities of this workshop can be found at https://github.com/uc3m-aerospace/Julia-semanaciencia2020



- Git is a versioning system. When coding, you will want to save your code and all the development history. Git is the tool!
- GitHub is an online service to store git repositories. It has tons of functionality. You can get a full account for free with your student/staff email

Fibonacci's numbers

- The Fibonacci sequence appears in a surprising number of very different fields. From the structure of spiral galaxies, to the way seeds are arranged in a sunflower, to economics and even art.
- Create a function to compute and print the first n Fibonacci numbers
- Use this function from the REPL to show that F_{n+1}/F_n approaches φ when n is large



$$\begin{aligned} F_0 &= 0, \\ F_1 &= 1, \\ F_n &= F_{n-1} + F_{n-2} \end{aligned}$$

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …

$$\lim_{n\to\infty} \frac{F_{n+1}}{F_n} = \varphi = \frac{\sqrt{5}+1}{2}$$

Fibonacci's numbers

- Functions are used to encapsulate code that can be called from the REPL or other code. Here is a very simple example:

```
function add1(input)
    output = input + 1
    return output
end
```

- Create a function that takes as input the number N of Fibonacci numbers to compute
- Initialize the vector F where you will store your computed numbers with `F = zeros(N)`
- You can access any entry of the vector using brackets. `F[1]` is the first element and `F[end]` the last one.
- You probably want to set `F[1] = 0, F[2] = 1`
- Create a loop to compute the rest of the Fibonacci numbers:

```
for i = i1:iend
    # Do stuff
end
```

$$
\begin{aligned}
F_0 &= 0, \\
F_1 &= 1, \\
F_n &= F_{n-1} + F_{n-2}
\end{aligned}
$$

Fibonacci's numbers (Hints cont.)

- Now iterate through the rest of the terms of the series
- How to use a loop? The simplest way is to use a semicolon range operator. Initially i = i1 and increases 1 each iteration, the last iteration is i = iend
- Print each term of the series using println(). Finally try to retrieve the approximate value of φ using the values stored in F

Is it a multiple of 7?

- I never know if a number like 841288 is a multiple of 7 or not!
- Create a function that takes an input number and returns “`It is a multiple of 7`” or “`It is not a multiple of 7`” accordingly
- Run the function from REPL to check if 841288 and/or 122145 are multiples of 7

- How to run conditional code?

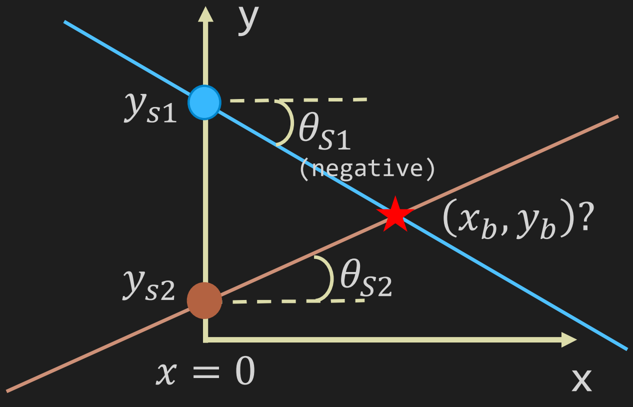
```
if condition
    # What to do if true
else
    # What to do if false
end
```

- Equality condition: `a == b` is true if a is equal to b
- To compute the remainder of `x/y`, use `x % y`
- Print to console with `println()`

Locate that ship!

- Imagine a ship in distress sends a signal to two stations on the coast, of which we know their position. The stations only know the *angle* at which the signal arrives, nothing more. Can we locate the ship?
- Write the equations for the two straight lines and find the intersection by solving the resulting linear system
- Write the system of equations in matrix form $A\boldsymbol{x} = \boldsymbol{c}$

$$y = n + mx,$$
$$m = \tan(\theta),$$
$$n = y(x = 0)$$



- The ship location belongs to both lines:

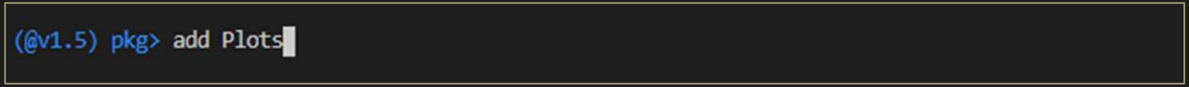
$$y_b = n_1 + m_1 x_b$$
$$y_b = n_2 + m_2 x_b$$

- We have a linear system of 2 equations with 2 unknowns

$$\begin{bmatrix} \tan(\theta_{S1}) & -1 \\ \tan(\theta_{S2}) & -1 \end{bmatrix} \cdot \begin{pmatrix} x_b \\ y_b \end{pmatrix} = \begin{pmatrix} -y_{s1} \\ -y_{s2} \end{pmatrix}$$

julia packages

- Remember the REPL? Enter Pkg mode by hitting “]”



```
(@v1.5) pkg> add Plots
```

- The Pkg mode allows installing, removing, and maintaining packages
- Install the Plots package for the next activity
- The first time you use a package, it will be compiled. This will take some time (just for the first time or if the package is updated)

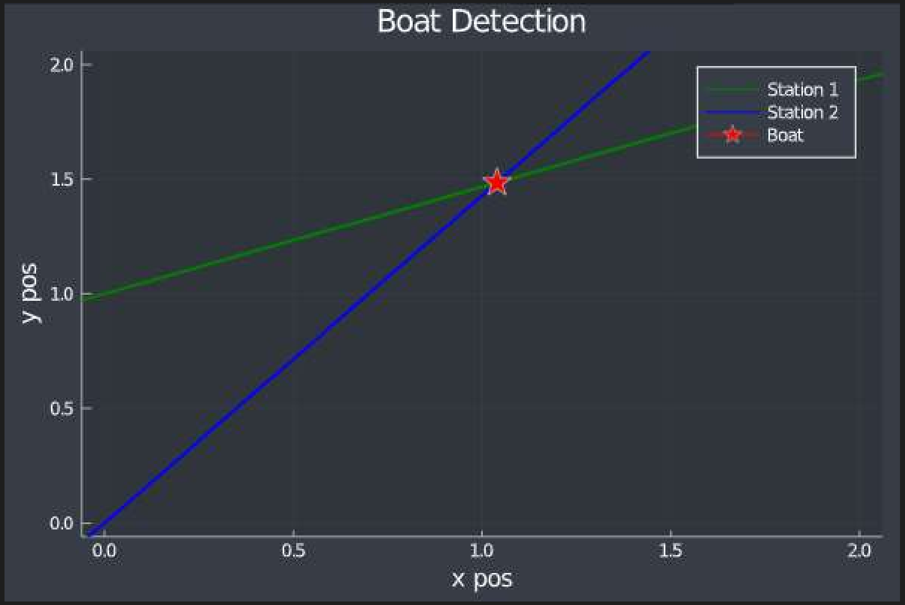
Now, show me the ship!

- It would be great if we could see the result of our previous work!
- Plot the two lines of the previous example and add a marker at the intersection.

- You must first load the Plots package with

```
using Plots
```

- Have a look at the Plots.jl documentation for style and attributes: http://docs.juliaplots.org/



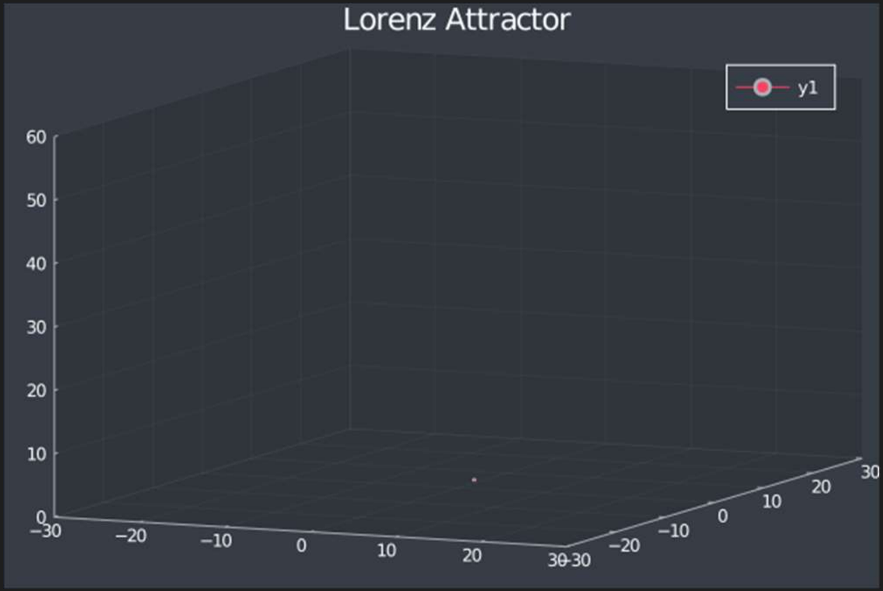
The Lorenz attractor

- Chaos is everywhere and unpredictable. A simple model of atmospheric convection proposed in 1963 by Edward Lorenz already shows this. Similar equations appear in many other applications: Lasers, electrical systems, chemical reactions...
- System of ordinary Differential Equations:

$$\frac{dx}{dt} = \sigma(y - x)$$
$$\frac{dy}{dt} = x(\rho - z) - y$$
$$\frac{dz}{dt} = xy \; - \beta z$$

- The Forward Euler Method can integrate this system numerically:

$$x(t + \Delta t) = x(t) + \frac{dx}{dt}\Delta t$$



σ = 10, $\rho = 28$, β = 8/3

The Lorenz attractor

```
dt  = 0.02
σ   = 10
ρ   = 28
β   = 8/3
```

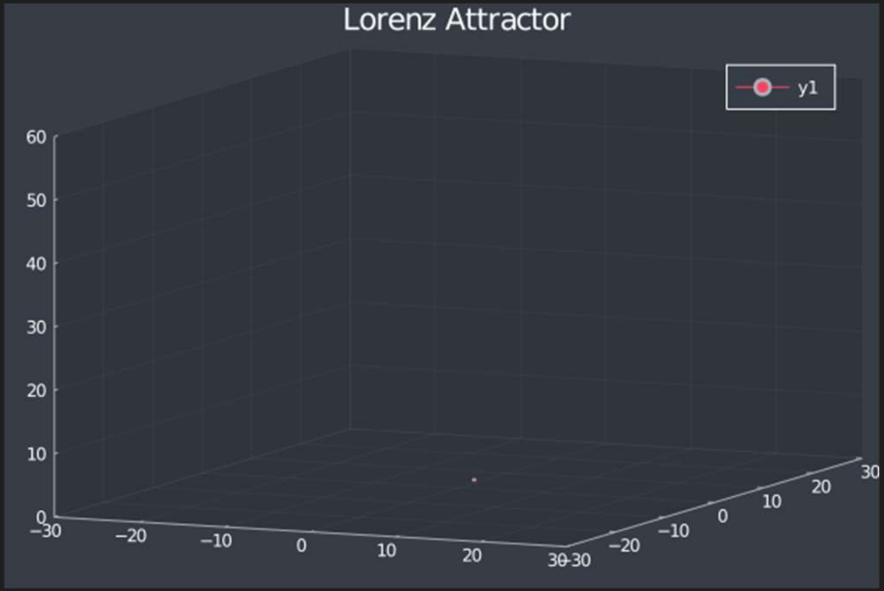
- Create a function that advances x, y, z one small timestep $\mathrm{d}t$ using Euler's forward method. Use the parameters on the right.
- Use this function to advance the solution, starting with:

```
x0   = 1
y0   = 1
z0   = 1
```

- Plot the resulting trajectory

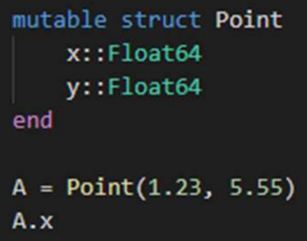
$$\frac{dx}{dt} = \sigma(y - x)$$
$$\frac{dy}{dt} = x(\rho - z) - y$$
$$\frac{dz}{dt} = xy - \beta z$$

$$x(t + \Delta t) = x(t) + \frac{dx}{dt}\Delta t$$

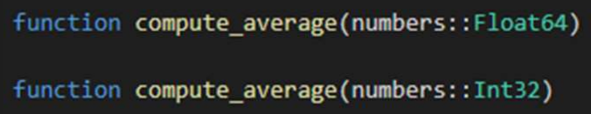


Other capabilities of julia

- We have barely scratched the surface!
- Julia’s main strength is the multiple dispatch system based on types

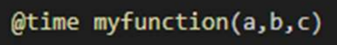


```
mutable struct Point
    x::Float64
    y::Float64
end

A = Point(1.23, 5.55)
A.x
```



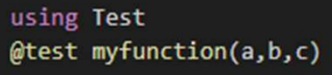
```
function compute_average(numbers::Float64)

function compute_average(numbers::Int32)
```

- Common types are Float64, Int32…
  But you can create your own types!
- Macros are functions with a convenient syntax (@ syntax)



```
@time myfunction(a,b,c)
```

- Unit testing is included out of the box



```
using Test
@test myfunction(a,b,c)
```

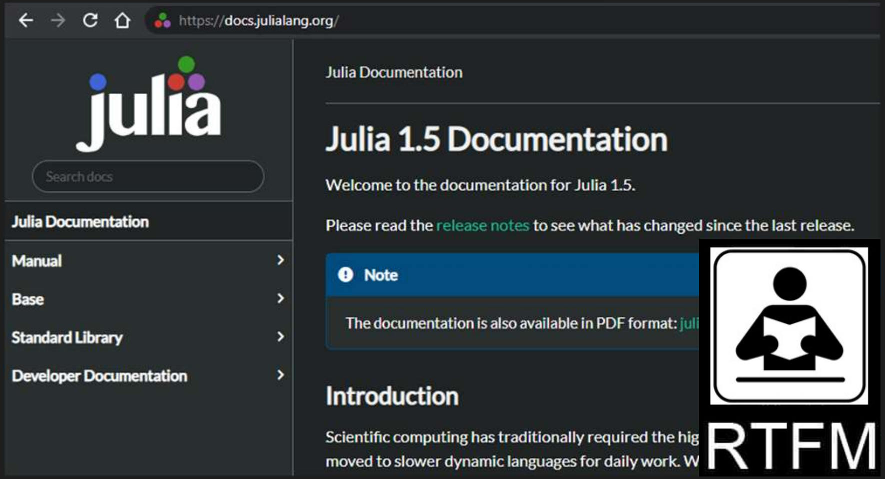
- Julia can run in multithreaded and distributed mode out of the box

Interesting packages

- There are tons of packages with extra functionality!
  - DifferentialEquations: amazing library to solve ODEs, PDEs
  - JuliaFEM: finite element method library
  - Pluto: reactive notebooks (see also IJulia in Jupyter)
  - Flux: machine learning
  - JuMP: optimization
  - PyCall: call python functions from julia
  - Revise: update function definitions atomatically as you work
  - OhMyREPL: adds some oomph to the REPL
  - MPI: parallel computing with MPI
  - CUDA: use your GPU for parallel computing
  - HDF5: read/write to this format of data files
  - PackageCompiler: compile julia code
- Check out juliahub.com, juliaobserver.com, juliapackages.com for more

Where to learn more and get help?

- Remember the "?" help function of the REPL
- Documentation! docs.julialang.org
- Tutorials, books and videos
- https://exercism.io/tracks/julia
- Great online course by MIT on YouTube with the participation of 3blue1brown
- Read the performance tips: docs.julialang.org/en/v1/manual/performance-tips/#man-performance-tips
- How do I…? Just google it!



https://docs.julialang.org/

Julia Documentation

Julia 1.5 Documentation

Welcome to the documentation for Julia 1.5.

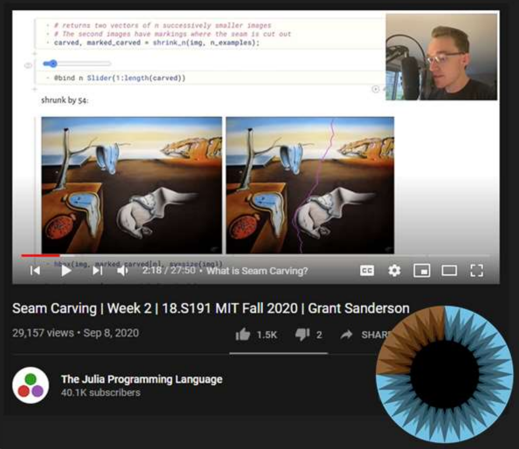
Please read the release notes to see what has changed since the last release.

Note

The documentation is also available in PDF format: juli

Introduction

Scientific computing has traditionally required the hig
moved to slower dynamic languages for daily work. W



```
# returns two vectors of n successively smaller images
# The second images have markings where the seam is cut out
carved, marked_carved = shrink_n(img, n_examples);
```

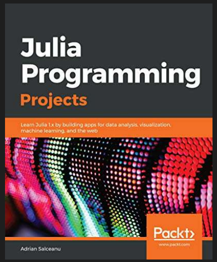
```
@bind n Slider(1:length(carved))
```

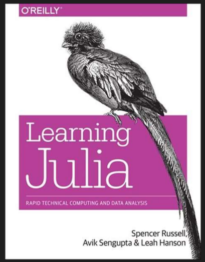
shrunk by 54:

Seam Carving | Week 2 | 18.S191 MIT Fall 2020 | Grant Sanderson

29,157 views • Sep 8, 2020

2:18 / 27:50 • What is Seam Carving?
1.5K
2
SHAR
The Julia Programming Language
40.1K subscribers





O'REILLY
Learning Julia
RAPID TECHNICAL COMPUTING AND DATA ANALYSIS
Spencer Russell,
Avik Sengupta & Leah Hanson

julia

semana de la ciencia y la innovación
Un planeta, muchos mundos 2020