SKILLPILLS

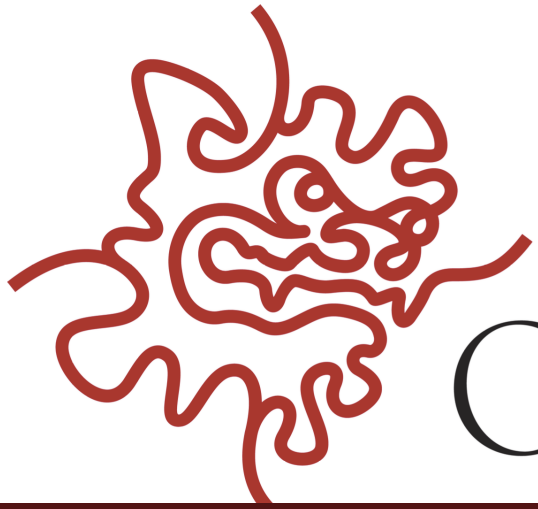
Skill Pill: Julia

Lecture 2: Data Processing and Plotting

Ankur Dhar & James Schloss

ankurd@oist.jp
james.schloss@oist.jp

July 12, 2019



OIST

1. Arrays
2. Data Handling
3. Plotting

Creating Arrays

Arrays in Julia are defined very similarly to arrays in Matlab, using square brackets to denote them. By default arrays are row vectors, but can be transposed to column vectors.

```
julia> x = [1 2 3 4]
julia> y = collect(1:4)
julia> z=[1;2;3;4]
julia> w=x'
julia> array = Int64[]
```

In addition sequences of numbers can be generated using the `:` operator (start:step:end). To see all these values expanded out you will need to print them manually or use the `collect` function.

Array Generation Functions

zeros(S)	Makes an array of size S filled with zeros
ones(S)	Makes an array of size S filled with ones
repeat(A,c,r)	Repeats array A column-wise c times and row-wise r times
rand(S)	Generates array of size S with random numbers between 0 and 1
Type[]	Creates empty array of type Type

Array Operations

Arrays can be indexed by using square brackets after the array. For multidimensional arrays, the first dimension is the row, followed by column and so on. Output of indexing arrays is by default a column. This is crucial when it comes to multiplying arrays by each other.

```
julia> data=rand(50,50)
julia> rowdata=data[1,:]
julia> inner = x*y
julia> outer = y*x
julia> square = x.*x
```

To execute scaler operations on an array, you can use append a `.` to the operator. This is also applicate to functions which acts on scalars as well.

File Structure

To begin working with files, you must know where your working directory is. When launching Julia from an application menu (Windows/MacOS), the default directory is predefined. For Linux it will be the home directory.

```
julia> pwd()
julia> cd("C:\\Users\\M\\Documents\\JuliaStuff")
julia> readdir()
```

You can use `pwd()` to print your working directory, and `cd()` to change it to whichever new directory you would like. Once you are in the directory you want, you can list the files within using `readdir()`.

Writing Scripts

As we start to get into more complex commands and chain them together, we can use scripts to automatically execute a series of commands at once. To include these commands into the REPL, we simply use the `include()` command

```
julia> include("Lesson 2.jl")
julia> randflips[1];
```

To help debug in these scripts, you can suppress the direct output from any commands with a `;`, and forcibly print values using `print()` or `println`.

Delimited text

The simplest data files are often delimited text files or CSV files, which can be manipulated like any other variable in Julia. To load any general delimited file, load the DelimitedFiles package. Reading in these files will automatically generate 1D or 2D arrays depending on the data being read in.

```
julia> using DelimitedFiles
julia> randData=readdlm("Random.txt")
julia> writedlm(randData,"Random.csv")
```

Similarly, any 1D or 2D can be written to a file, with the actual delimiter being based on the file extension used (txt for space and csv for comma).

CSV files

Typical CSV files are a bit more complex than standard delimited files, with headers or labels. For this use case the CSV package is recommended. When reading in a file, the first row will be taken as the header row, but this can be explicitly defined.

```
julia> using CSV
julia> data = CSV.read("simplemaps-worldcities-basic.csv")
julia> names(data)
julia> populations = data[:pop]
```

Once read in, the labels along the header row can also be used to reference each column of data, so numeric indexing is not not necessary.

Exercises

Exercise 1

Creates a file `squares.txt` consisting of the first 5 square numbers

Exercise 2

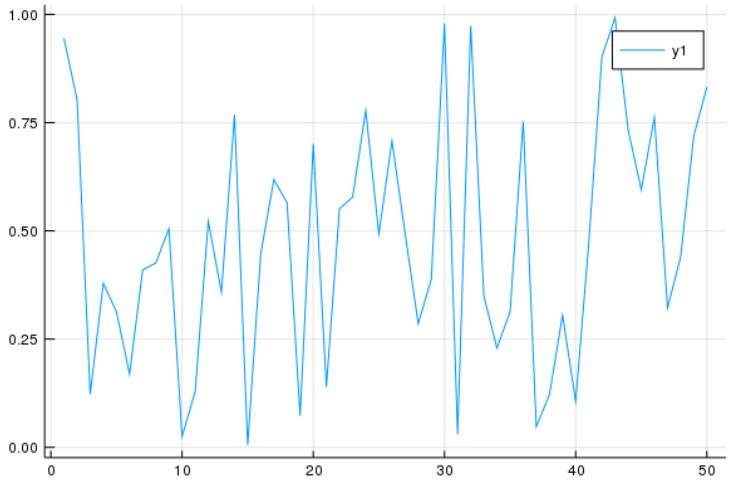
Write a script which creates a new file called `large_cities.txt`. The file should contain one line for each of the cities which have a population larger than 10,000,000., formatted as follows:
Buenos Aires, Argentina: population 11862073
Sao Paulo, Brazil: population 14433147.5
...

Plots in a Nutshell

To start with, let's add and load the general Plots package

```
(v1.1) pkg> add Plots
julia> using Plots
julia> plot(Plots.fakedata(50))
```

This will let us call the general Plot commands, in this case `plot` plots 1D data as a line plot.
More information can be found at `https://docs.juliaplots.org/latest/`



Plotting Commands

scatter(X,Y)	Scatter plot data with XY coordinates
bar(x,y)	Bar plot following similar rules to plot
histogram(x,bins=n)	Plots histogram of 1D data in n bins.
plot(θ,r,proj=:polar)	Polar plot of data following r and θ
heatmap(x,y,z)	Plots heatmap following XY axes with intensity array z
fakedata(L,S)	Generates random S numbers of series data of length L
savefig(filename)	Saves a generated plot as an image file

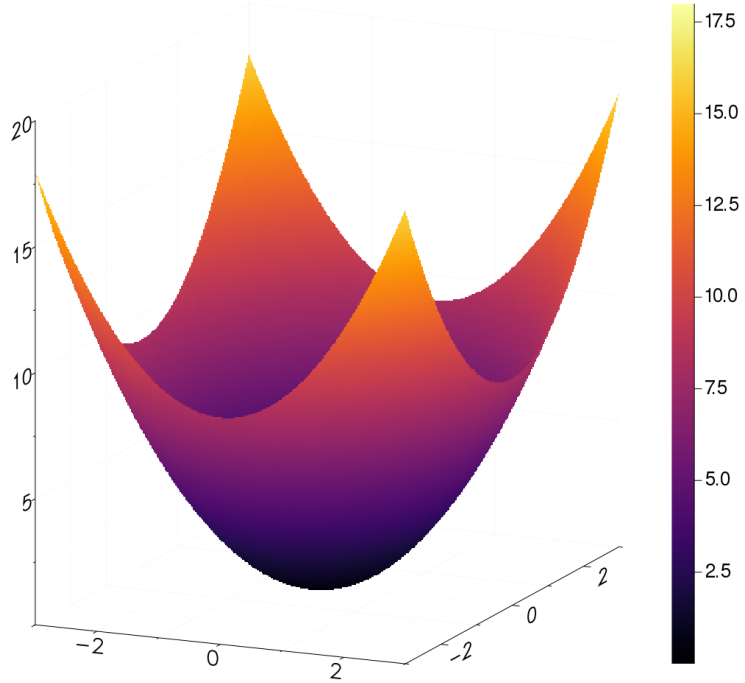
Plotting 3D Data

There are a couple of additional options for plotting 3D data:

surface(x,y,z) Draws surface in 3D space

contour(x,y,z) Draws contours on 2D plane

The plotting commands `plot`,`scatter`,`bar`, and `heatmap` also can accept 3D data.



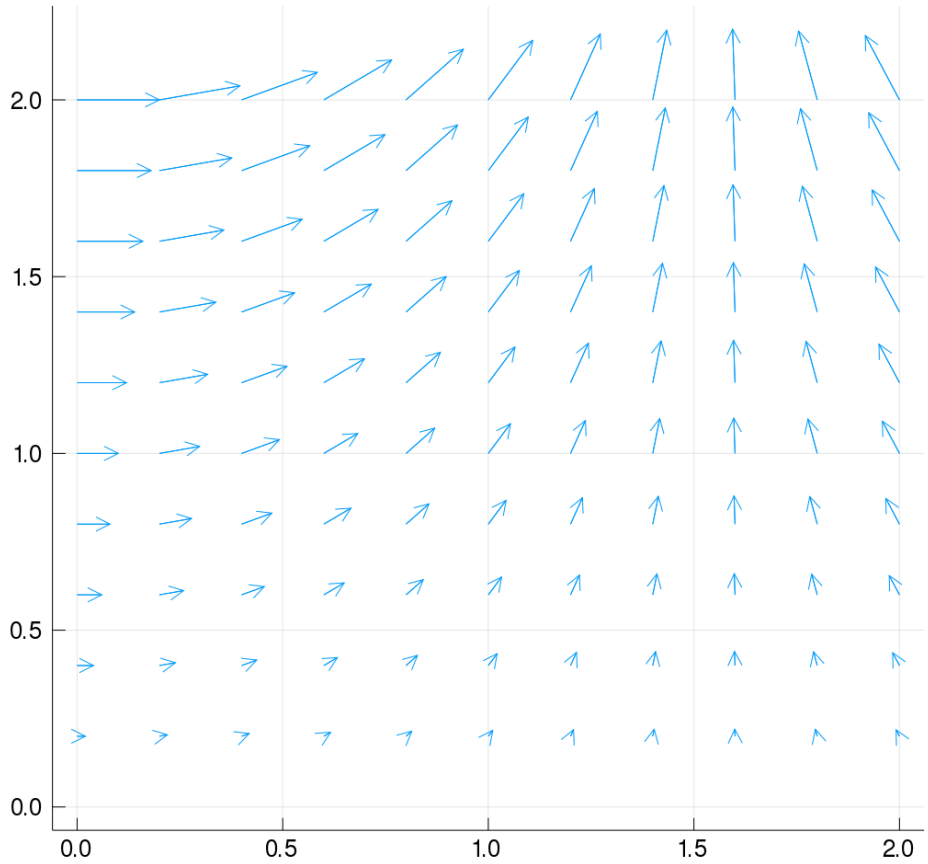
Plotting Vector Data

For vector data the current option is quiver:

```
help?> quiver
search: quiver quiver!

quiver(x,y,quiver=(u,v))
quiver!(x,y,quiver=(u,v))

Make a quiver (vector field)
    plot. The ith vector
    extends from (x[i],y[i])
    to (x[i] + u[i], y[i] +
    v[i]).
```



Plotting Backends

These commands are agnostic to the plotting backend, meaning they will work with a number of plotting engines in similar fashion. Each backend has pros and cons, but most commonly used are GR and PyPlot.

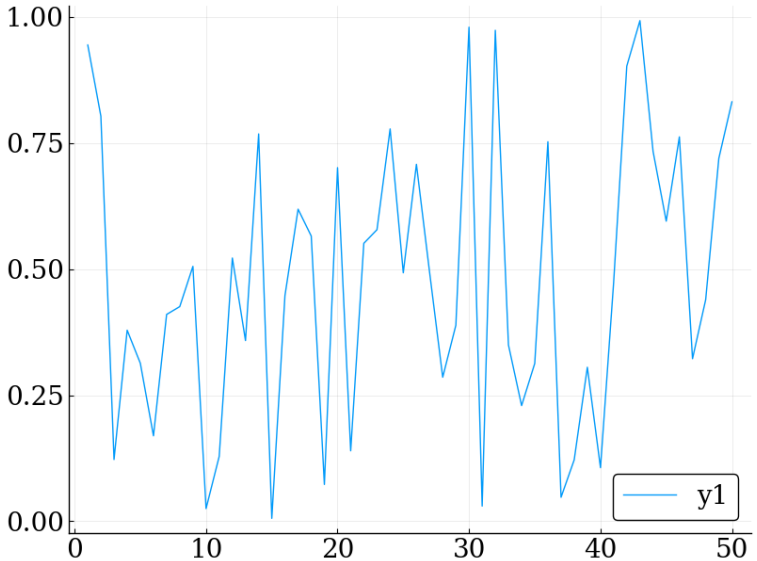


Figure : PyPlot

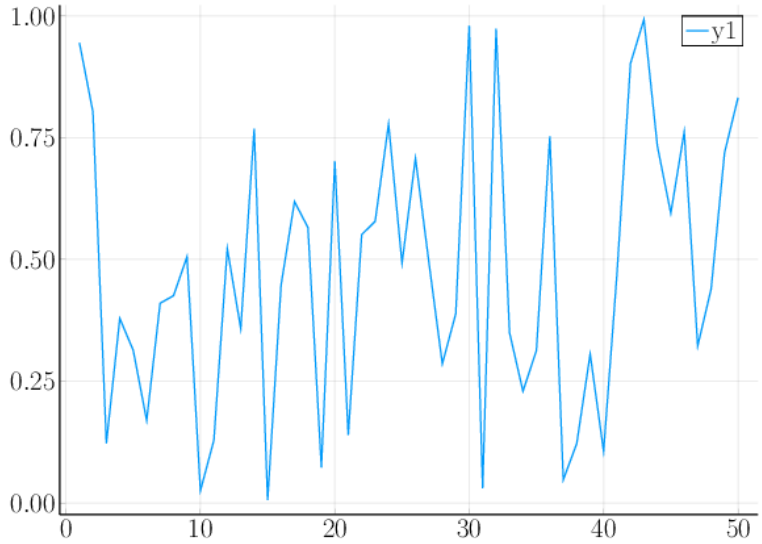


Figure : PGFPlots

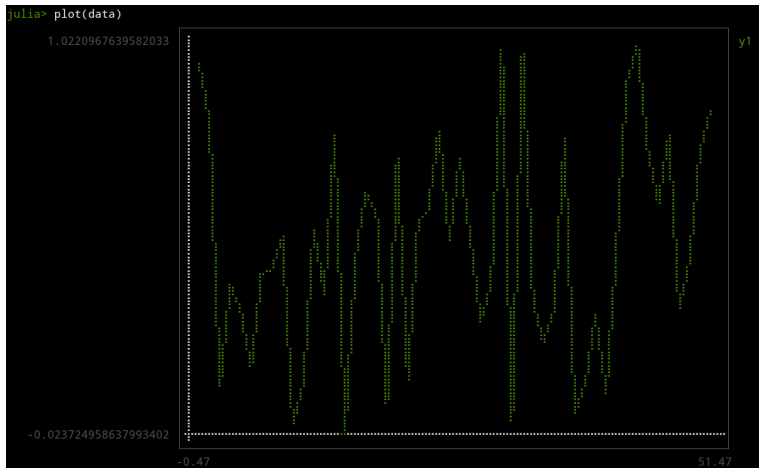


Figure : UnicodePlots

Formatting Commands

Each of these commands can be expressed in-line with the plotting command or beforehand within a call to the plotting backend.

font(fontname,size)	Defines a Font object with a given size
size=(X,Y)	Sets size of plot to X by Y pixels
xlabel=string	Sets X-Y labels to string, also ylabel
title=string	Sets title to string, also colorbar_title for heatmap.
xtickfont=font	Sets the font of x tick marks, also ytickfont,titlefont,guidefont
left_margin=length	Sets margin for left side of plot, also top_margin, bottom_margin, and right_margin
xscale=:log10	Sets x scale to log10, also yscale

Exercises

Exercise 3

Read data.txt given in the Public Folder and plot the results. What do you see?

Exercise 4

Plot a histogram of the longitudes of the world's cities. What is the mean and median longitude?

What is next?

Last Session Data Structures and Algorithms

Also next week Advanced Topics with Valentin!