Scientific programming with Julia

Stephen J Eglen

2017-10-05

1 Outline

2 Books and online help

3 Aims of course

4 Part 2: Scientific computing issues

5 What is Julia?

Invented by MIT guys with selfish aims

6 History

appeared "a few years ago"

7 Strengths of \R

fast, efficient, great development team

8 Graphics example

9 Weaknesses of Julia

still developing. precompilation helps, but loading libraries (e.g. Gadfly) v slow.

10 Brief comparison to matlab

???

11 Using Julia

Julia/Jupyter /Juno.

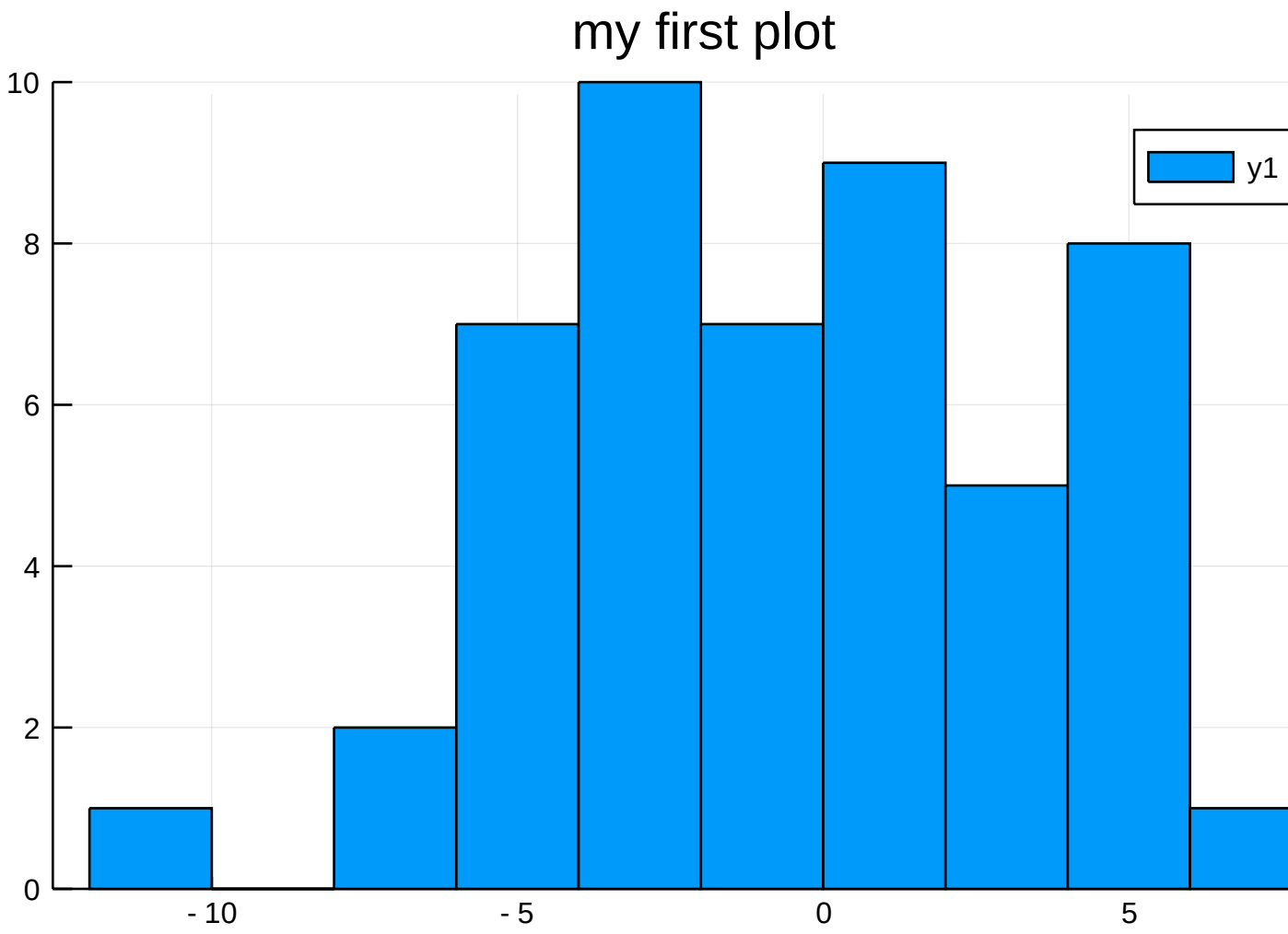
12 My very first Julia session

```
x = randn(50) * 4
x
```

```
50-element Array{Float64,1}:
  -5.64852
  -4.60538
  -3.6238
   4.26176
   0.956262
  -2.91246
   0.505952
   2.06196
  -3.51399
   5.10857
  -1.17366
   4.37993
   1.4295
   0.867014
   4.02651
 -10.4624
  -4.45309
   1.68978
  -0.852409
   1.11427
  -1.86386
   2.56325
   3.23083
  -4.65677
  -1.37933
  -4.24297
   1.31633
   5.65277
  -0.0170792
   0.740651
  -2.06391
   1.15294
  -0.961022
  -2.38118
  -0.399008
  -3.33798
   3.45847
  -6.11413
  -3.49324
  -6.23889
```

```
 -2.02145
  2.07048
  4.85014
 -3.52307
  6.18793
 -3.19481
 -4.46528
 -5.99339
  4.6448
  4.24126
```

```
mean(x)
minimum(x), maximum(x)
#Pkg.add("Plots"); Pkg.add("GR")
using Plots; gr()
histogram(x, nbins=10, title="my first plot")
```



13 Interacting with Julia

- Can use up/down keys to go through command history.
- Use semicolon to suppresss output and to put multiple commands on one line.
- TAB for completion.

14 Objects and Functions

naming conventions for objects. No assignment arrow in Julia.

15 Objects and Functions

```
x = 200
half_x = x / 2
threshold = 95.0
age = [15 19 30]
age[2]
length(age)
```

```
3
```

16 Vectors

Fundamental type in Julia. Scalars are distinct from Vectors.

```
julia> y = [10 20 40]
1ÃŮ3 Array{\{}Int64,2{\}}:
 10  20  40

julia> y[2]
20

julia> length(y)
3

julia> typeof(y)
Array{\{}Int64,2{\}}

julia> y = 5
5

julia> length(y)
1

julia> typeof(y)
Int64
```

Note how it changes type.

17 Vectors

Some operations work element by elment, others on the whole vector., compare:

```
y = [20 49 16 60 100]
minimum(y)
(minimum(y), maximum(y))
sqrt.(y)
log.(y)
```

TODO: dot notation for extending to vector. Broadcasting.

18 Generating vectors

TODO: Ranges treated differently, to save space. To expand a range from its compact notation to a normal vector, use `collect`.

```
1:2:9
x = collect(1:2:9)
y = collect(linspace(2, 7, 3))
z = 4:8
a = 1:5
b = [3 9 2]
d = [collect(a)' 10 b]
e = repmat( [1 2], 3)[:]
f = zeros(7)
## TODO: distinction:  row vs col vectors
```

19 Accessing and setting elements

```
julia> x = collect(100:1:119)
20-element Array{\{}Int64,1{\}}:
 100
 101
 102
 103
 104
 105
 106
 107
 108
 109
 110
 111
 112
 113
 114
 115
 116
 117
 118
 119

julia> x[3] ## just element 3
102

julia> x[[12 14]]
1ÃŮ2 Array{\{}Int64,2{\}}:
 111  113

julia> x[1:5]
5-element Array{\{}Int64,1{\}}:
 100
 101
 102
 103
 104
```

```
julia> bad = 1:4
1:4

julia> ## TODO negated indexes not present?
x[end]
119
```

Julia does not index by negation. It does however allow you to use the `end` keyword to get the last element.

20 Accessing and setting elements

```
julia> x = [5 2 9 4]
1ÃŮ4 Array{\{}Int64,2{\}}:
 5  2  9  4

julia> v = [true false false true]
1ÃŮ4 Array{\{}Bool,2{\}}:
 true  false  false  true

julia> x[v]
2-element Array{\{}Int64,1{\}}:
 5
 4
```

Use logical elements to access.

21 Accessing and setting elements

```
x = zeros(10)
x[1:3] = 2
x[5:6] = [-5 NaN]
## x[7:10] = [1 9] ## will error - no recycling
```

22 Recycling rule \adv

TODO: recyling absent?

repmat() and friends might be needed.

vector + scalar works

```
[1 2] + 3
```

23 Naming indexes of a vector

TODO: not possible?

24 Common functions for vectors

- length()

- reverse()
- sum, cumsum, prod, cumprod
- minimum, maximum, range?, summary?

25 Functions as function args

26 Default values for function arguments

27 Default values for function arguments

28 Argument matching

29 Argument matching

30 \ldots in function calls \adv

31 Replacement functions \adv

32 Replacement functions \adv

33 Getting help: key commands

34 Help pages

35 Numbers and special values

36 Numbers and special values

37 Numbers and special values

38 Operator precedence Juliafunction ?Syntax

39 Operator precedence \Rfunction ?Syntax

40 Operator precedence \Rfunction ?Syntax

41 Operators

42 When things go wrong

43 Types of parentheses

44 From interactive to source files

```
using Weave
weave("spj.jmd", informat="markdown", out_path = :pwd, doctype = "pandoc")
weave("spj.jmd", informat="markdown", out_path = :pwd, doctype = "md2html")
weave("spj.jmd", informat="markdown", out_path = :pwd, doctype = "md2pdf")
```