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.

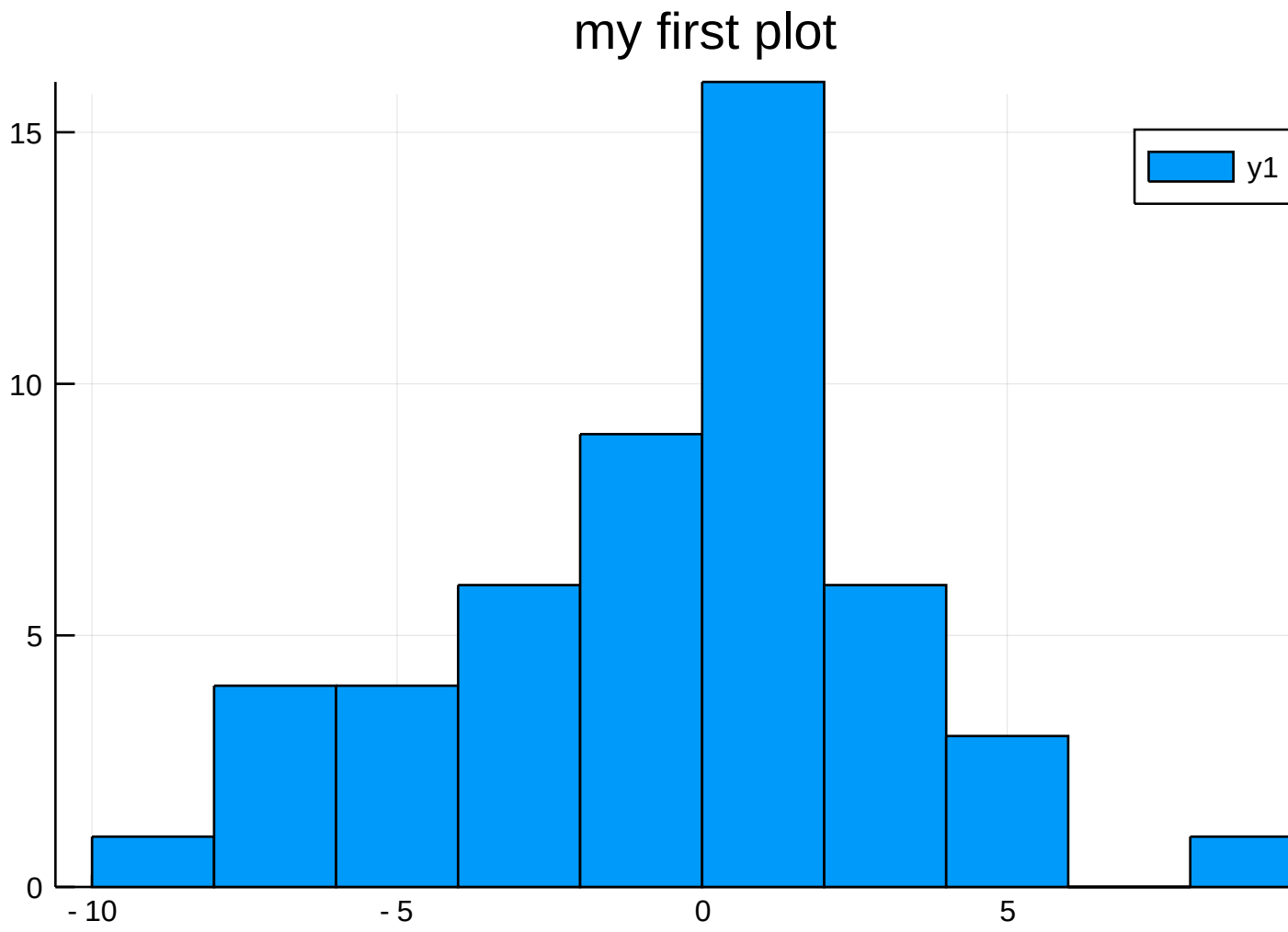
12 My very first Julia session

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

```
50-element Array{Float64,1}:
  4.30425
  5.72249
 -0.909512
  1.40633
  2.13246
  0.422634
 -6.02775
  1.96249
 -2.1927
  1.77533
  3.54728
 -0.49318
  1.89536
 -5.48103
  0.695747
  1.40389
  1.27129
 -6.28813
  0.108089
  1.8439
 -7.16459
 -0.462907
 -1.22906
 -6.18751
 -0.90086
 -3.53268
 -8.04917
  0.213352
 -3.93743
 -0.679358
 -1.45796
 -0.0295909
  2.35066
  0.826435
 -5.14273
  1.63748
 -3.60524
 -0.613968
 -3.20238
  8.93718
```

```
  0.412885
 -5.71833
  3.52647
  3.67592
  4.89975
 -3.23992
  2.17251
 -4.94949
  0.506986
  0.351903
```

```
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

```
x = [3, 2, 9, 4]

y = exp.(x); z1 = find(y .> 20.0) ## case 1
z2 = find( exp.(x) .> 20.0)       ## case 2

z1 == z2

true
```

26 Default values for function arguments

```
x = [2.091, 4.126, 7.925]
round.(x)
round.(x, 2) # TODO not named arguments...  is round best example?
```

27 Default values for function arguments

28 Argument matching

29 Argument matching

30 \ldots in function calls \adv

splat ?

31 Replacement functions \adv

no such equivalent.

32 Getting help: key commands

?round

33 Numbers and special values

Key numerical types:

- Integer
- Float64 ("double")
- Float32 ("single precision")
- TODO: complex values.

TODO: check children of an abstract type âĂŞ how to generate tree?

```
typeof(1)
typeof(1.0)
isa(1, Float64)
isa(1.0, Float64)
## TODO isnumber(4)
```

34 Numbers and special values

Julia does not have NA in base language (might be in the Statistics modules)

```
0/0
3/0
(-1)/0
isnan(0/0)
isinf(3/0)
nothing

nothing

typeof(nothing)

Void
```

35 Operator precedence

```
3 * 4 + 2 != 3 * (4 + 2)
2^3+1  != 2^(3+1)
1:5-1 # (cf R)
```

36 Operator precedence \Rfunction ?Syntax

TODO: find the precedence list

37 Operators

```
x = 10
x == 4      ## test for equality
x != 10     ## not equal?
div(7, 2)   ## division, ignoring remainder...   (3)
7 Ãů 2       ## ...   the unicode way.
7 % 2       ## remainder (1)]
10.1^2.5    ## 10.1 ** 25 not allowed.
[1 2] === [1 2]
a = [1 2]; a === a
```

TODO: globals not an issue?

38 When things go wrong

39 Types of parentheses

40 From interactive to source files

EditâĂŞrunâĂŞedit cycle.

See contents of src/trig.jl

41 Scripts

```
include("src/trig.jl")
```

```
[0.0, 0.0634239, 0.126592, 0.189251, 0.251148, 0.312033, 0.371662, 0.429795
, 0.486197, 0.540641]
```

42 Why are scripts a good thing?

43 Running scripts in batch \adv

44 Rscript

no equivalent

45 Matrices

Matrices can be made directly or converted from vectors.

```
x = reshape(1:6, 2,3)
y = [1 2 3; 4 5 6]
z = hcat(1:3, 4:6)'
x = y
```

46 Matrices

```
x[2,2] ## extracting a value.
x[1,:]  ## extracting row
x[1:2, 2:3]
x[:,2]  ## not column vector!
```

TODO: consider views vs copies.

47 Typical matrix construction methods

```
reshape(1:6, (2,3))
x = vcat( [1 4 9], [2 6 8], [3 2 1])
y = hcat( [1 2 3]', ones(3,1)*5, [4 5 6]')
```

48 Typical matrix construction methods

Matrix indices are not named in Julia

49 Common matrix operations

```
x = [1.0 2.0; 3.0 4.0];
i = eye(2)
x * i
x'      ## transpose
inv(x) ## inverse of a matrix
x * inv(x)
isapprox(x * inv(x), eye(x))
x * inv(x) âĽĹ eye(x)  ## unicode way
```

```
true
```

49.1 Arrays

Multidimensional matrices

```
a = reshape(collect(1.0:12.0), 2, 2, 3)
```

50 Boolean values

logical values `true` and `false` take the values 1 and 0 as in other languages. But you cannot use "1" where a boolean "true" is expected.

```
true + true
```

```
2
```

51 Boolean values

```
d = [3.1 1.0 4.0 9.2 2.3 8.1 6.3]
d .> 5.0
d[d .> 5.0]
find( d .> 5.0)
```

```
3-element Array{Int64,1}:
 4
 6
 7
```

52 Boolean values

ifelse vectorized?

```
(10 > 20) ? 5 : 3  # just like C
d = collect(1.0:10.0)
ifelse.( d .> 3.0, 1.0, 0.0)
```

53 Boolean values âĂŞ \Rfunction ?logical

```
!true
true & false
false | true
xor(true, true)
```

```
false
```

54 Boolean logic: issues

Lazy evaluation is complicated here? `&&` and `||` are lazy operators

```
true || sleep(1000)
```

55 What is a list?

Two ways of getting a list in Julia; with a `Dict` so that they can be named (but not indexed by number), or with a vector.

```
d = Dict("who"=>"joe",
         "height"=> 1.70,
         "dob" => [1960 12 1])
length(d)
keys(d)
d["height"]
##d[2] # error
## the vector approach
l = ["joe", 1.70, [1960 12 1]]
length(l)
l[2]
```

56 Data frames

Not in core Julia; see DataFrames package.

57 Factors

???

58 Strings / character arrays

Use double quotes for strings in Julia.

```
x = "good"
```

59 Strings / character arrays

```
print("Now computing the steady state\n")
```

```
Now computing the steady state
```

```
## julia specific bits TODO mode
## push!, pop!, shift!, unshift!
x = 134
print("sqrt of $(x)is $(sqrt(x))")
```

```
sqrt of 134 is 11.575836902790225
```

60 Strings / character arrays

```
x = collect(1:5); exp_dir = "/home/stephen/res"
## TODO fix
##@sprintf("hello %d %s", x[2], exp_dir)
##[@sprintf("hello %d %s" i, exp_dir) for i in x]
```

TODO: paste doesn't seem to have a flexible friend in Julia. join may be the closest relative.

61 Strings

```
## https://docs.julialang.org/en/stable/manual/strings/
s =["apples" "bee" "cars" "danish" "egg"]
length(s)
length.(s)
## substr todo
```

62 Strings

63 Strings

64 Environments \adv

65 Inspecting variables and the environment

66 Converting an object from one type to another

67 What is an object?

68 OO programming in \R

69 Basic plotting

70 Basic plotting

71 Basic plotting

72 Options controlling the plot

73 Multiple data sources on one plot

74 Multiple plots in one figure

75 Saving your plots

76 Next steps with plotting \adv

77 Reading/writing data to file system

78 Interacting with the file system

79 Scan, write, readLines

80 Scan, write, readLines

81 \Rfunction read.table et al.

```
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")
```