Julia Cheatsheet V0.4

Compiled by Debbie Chapman and Hao Cheng from UC Davis Animal Quantitative Genetics Lab (http://QTL.rocks).

Last Updated June 21, 2019

Matrices

Column Vector

	Julia Code
$a = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$	`a = [1, 2, 3]`

Row Vector

	Julia Code
a = [1 2 3]	`a = [1 2 3]`

Matrix

	Julia Code

$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$	`A = [1 2;3 4;5 6]`
	or
	`A = [1 2`
	`3 4`
	`5 6]`

Initialize Matrix

Matrix (of size 3×2)	`Array{Float64}(undef, 3, 2)`
	`zeros(Int64, 3, 2)`
	`ones(Float64, 3, 2)`
	`fill(1.0, 3, 2)`
	`rand(Float64, 3, 2)`

Concatenation

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \text{ and } B = \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix}$$

	Julia Code
$M = \begin{bmatrix} 1 & 2 & 7 & 8 \\ 3 & 4 & 9 & 10 \\ 5 & 6 & 11 & 12 \end{bmatrix}$	`M = [A B]` `M = hcat(A, B)`
$N = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ \dots & \dots \\ 11 & 12 \end{bmatrix}$	`N = [A; B]` `N = vcat(A, B)`

Submatrix

	Julia Code
$S = \begin{bmatrix} 3 & 4 \\ 5 & 6 \end{bmatrix}$	`S = M[2:3, 1:2]`
$s = \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix}$	`s = M[:, 2]`
$S = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 7 & 8 \end{bmatrix}$	`S = N[[1,2,4], :]`

Linear Algebra

Import Package*	`using LinearAlgebra`
Matrix Addition/Multiplication	`A + B` `A * B`
Scalar Multiplication:	`A * 10`
Elementwise Addition/Multiplication:	`A .+ 10` `A .* B`
Matrix Inverse	`inv(S)`
Matrix Transpose	`M'`
Vector of Diagonals	`diag(M)`
Diagonal from Vector	`Diagonal(s)`
Identity Matrix	`M*Matrix{Float64}(I, 4, 4)` `M*I`

* The linear algebra functions are part of Julia's standard library, but need to be imported.

Syntax

Arrays

a = [9, 10, 11, 12, 13, 14]

newArray = [9, 11, 13]	`newArray = oldArray[start:step:stop]` `newArray = a[1:2:end]`

* The Julia element index starts with one, not zero

Dictionaries

key	weight value
Ind1	92
Ind2	94

Create dictionary	`weight = Dict("Ind1" => 92, "Ind2" => 94)`
Create empty dictionary	`weight = Dict{Any, Any}()` `weight = Dict{String, Int64}()`
Add key and value to dictionary	`weight["Ind3"] = 85`
Remove key and value from dictionary	`delete!(weight, "Ind2")`
Find value of a key	`weight["Ind3"]`

Types

Attach type	`x::Int32`

Create type	`struct Dog`
	`age::Int64`
	`coatColor::String`
	`end`

Construct	`clifford = Dog(3, "red")`
Check the type of a variable	`typeof(clifford)`
Find value of variable within structure	`clifford.age`

Control Flow and Loops

Conditional	`if - elseif - else - end`

For Loop	`for i in 1:10`
	`println(i)`
	`end`

While Loop	`while x < y`
	`x *= 2`
	`end`

Exit loop	`break`

Mathematical Operators

Julia Code	Output
`1 + 1`	2
`5 - 2`	3
`10 * 3`	30
`12 / 6`	2.0
`6 \ 12`	2.0
`div(5, 2)`	2 (div yields a truncated result)
`15 % 2`	1

Comparison Operators

Julia Code	Output
`true || false`	true
`true && false`	false
`!true`	false
`1 == 1`	true
`1 != 1`	false
`[1,2,3] .== [2,3,1]`	false
`1 < 2 < 3`	true
`2 <= 3 || 2 >= 1`	true

Miscellaneous

Insert greek letter alpha	\alpha (then press tab)
Generate random number (0 to 1)	rand()
Generate random number (∼N(0,1))	randn()
View documentation for function	?function
View methods for function	methods(function)
Measure performance of function	@time function()
Sort column vector	sort(myVector)
Get vector of sorted indices	sortperm(myVector)
Sort matrix by third column	M[sortperm(M[:, 3]), :]
Number of rows in matrix	size(M, 1)
Number of columns in matrix	size(M, 2)
Find array element indices	findall(x -> x == 2, myArray)
Find first index	findfirst(x -> x == 2, myArray)
Find last index	findlast(x -> x == 2, myArray)
Find array element values	filter(x -> x > 1, myArray)
Apply function to every element	map(function, M)
Rolling/Accumulating computation	reduce(*, M)
Remove last value in collection	pop!(myArray)
Remove first value in collection	popfirst!(myArray)
Add value to collection (last)	push!(myArray, newValue)
Add value to collection (first)	pushfirst!(myArray, newValue)
String concatenation	var1 = "Julia" var2 = "Cheatsheet" string(var1," ",var2) var1*" "*var2 "$var1 $var2"
Create function	function checkSign(x) if x > 0 return "positive" else return "nonpositive" end end
Get list of subtypes	subtypes(AbstractString)
Determine abstract type	supertype(String)
Compare type hierarchy	String <: AbstractString
Sparse array	using SparseArray sparse([row], [col], [value])

Packages

To install a package, in the Julia REPL, type the following commands:

```
using Pkg
Pkg.add("Package Name")
```

Below is a useful link for Julia package documentation:

https://pkg.julialang.org/docs/

I/O

DataFrames.jl

Import package	using DataFrames
Read file	data = readtable("pedigree.txt")
Sort by first & fifth column	sort(data, [1, 5])
Sort by labeled column	sort(data, :Marker5)
Check columns for missing values	describe(data, stats=[:nmissing])
Remove rows with missing values	dropmissing(data)
View individuals with specific values	data[data.Sire .> "a3", :]
Write file	writetable("cleanPedigree.txt", data)

CSV.jl

Import package	using CSV
Read file	data = CSV.read("pedigree.txt")
Write file	CSV.write("cleanPedigree.txt", data)

Genomic Prediction

JWAS.jl

Load packages	using JWAS,CSV,DataFrames
Read data	phenotypes = CSV.read("phenotypes.txt",delim = ',',header=true) pedigree = get_pedigree("pedigree.txt",separator=",",header=true)
Build Model Equations	model_equation = "y1 = intercept + x1 + x3 + ID + dam y2 = intercept + x1 + x2 + x3 + ID y3 = intercept + x1 + x1*x3 + x2 + ID" model=build_model(model_equation, R)
Set Factors or Covariate	set_covariate(model,"x1")
Set Random or Fixed Effects	set_random(model,"x2",G1) set_random(model,"ID dam",pedigree,G2)
Use Genomic Information	add_genotypes(model,"genotypes.txt",G3,separator=',')
Run Bayesian Analysis	outputMCMCsamples(model,"x2") out=runMCMC(model,phenotypes,methods="BayesC",output_samples_frequency=100)