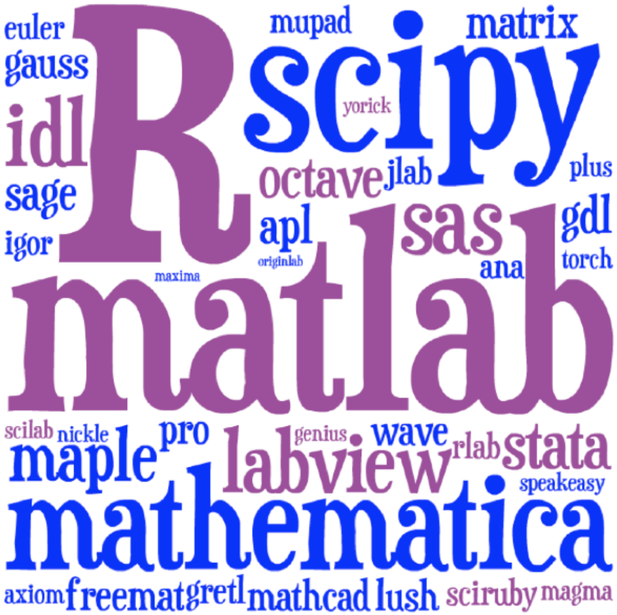
Julia: A modern language for modern ML

Dr. Viral Shah and Dr. Simon Byrne

Julia computing

www.juliacomputing.com

What we do: Modernize Technical Computing

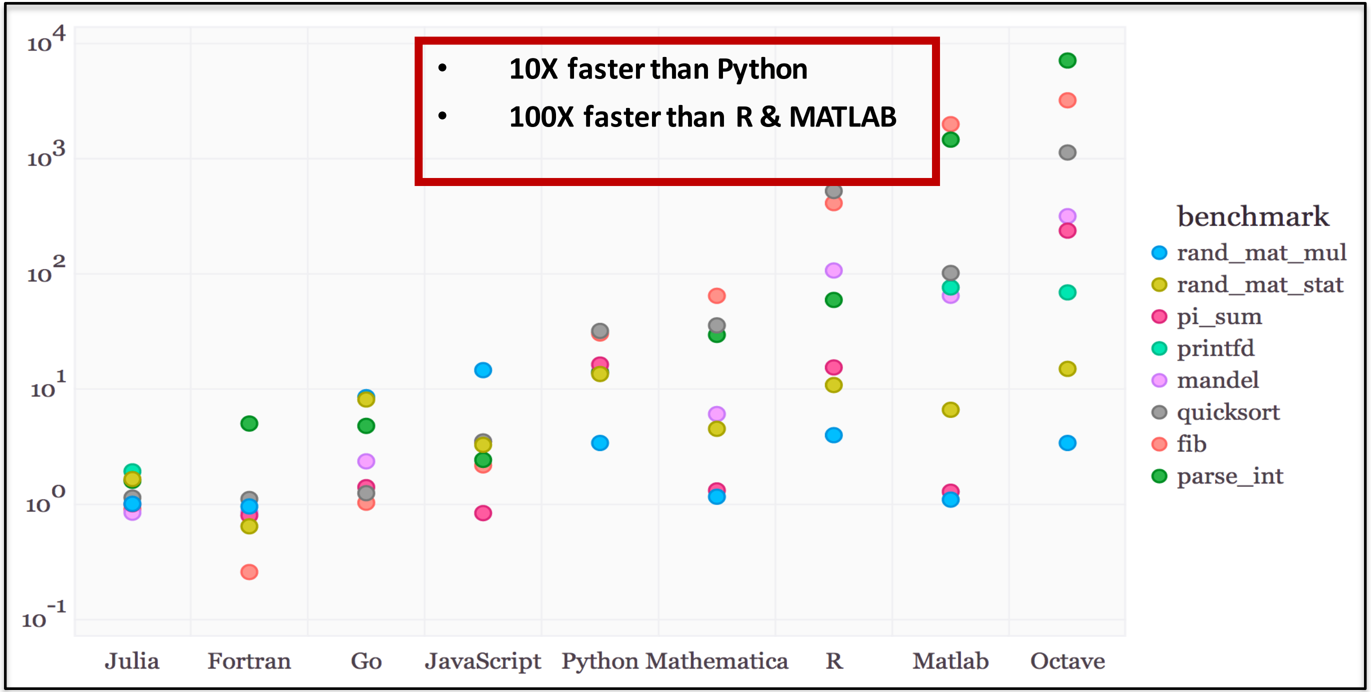


Today’s technical computing landscape:

- Develop new learning algorithms
- Run them in parallel on large datasets
- Leverage accelerators like GPUs, Xeon Phis
- Embed into intelligent products

“Business as usual” will simply not do!

General Micro-benchmarks:
Julia performs almost as fast as C



- 10X faster than Python
- 100X faster than R & MATLAB

Performance benchmark relative to C. A value of 1 means as fast as C. Lower values are better.

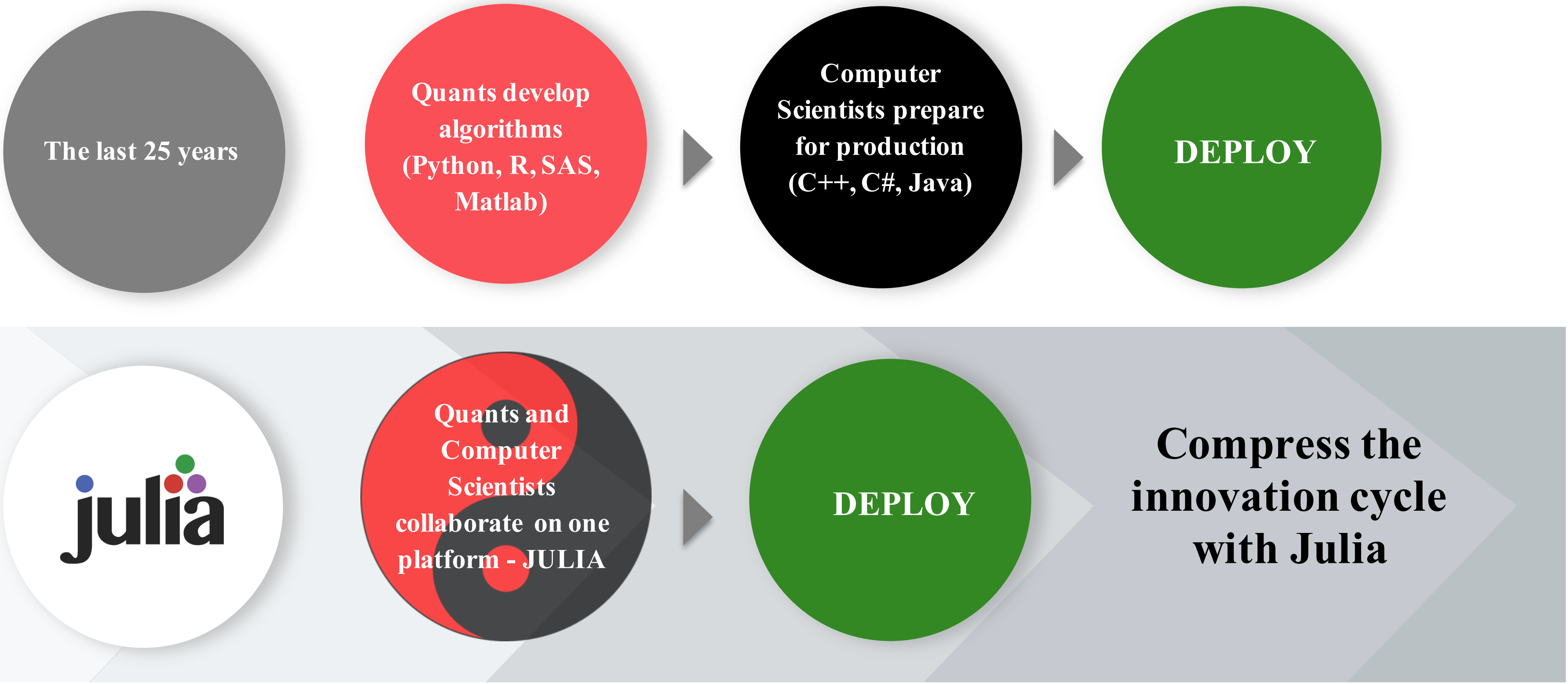
A real application: Gillespie simulations in systems biology

745x faster than R

- Gillespie simulations are used in the field of drug discovery.
- Also used for simulations of epidemiological models to study disease propagation
- Julia package (Gillespie.jl) is the state of the art in Gillespie simulations
- https://github.com/openjournals/joss-papers/blob/master/joss.00042/10.21105.joss.00042.pdf

Implementation	Time per simulation (ms)
R (GillespieSSA)	894.25
R (handcoded)	1087.94
Rcpp (handcoded)	1.31
Julia (Gillespie.jl)	3.99
Julia (Gillespie.jl, passing object)	1.78
Julia (handcoded)	1.2

Those who convert ideas to products fastest will win



The last 25 years

Quants develop algorithms (Python, R, SAS, Matlab)

Computer Scientists prepare for production (C++, C#, Java)

DEPLOY

Quants and Computer Scientists collaborate on one platform - JULIA

DEPLOY

Compress the innovation cycle with Julia

Julia offers competitive advantages to its users

watersTechnology

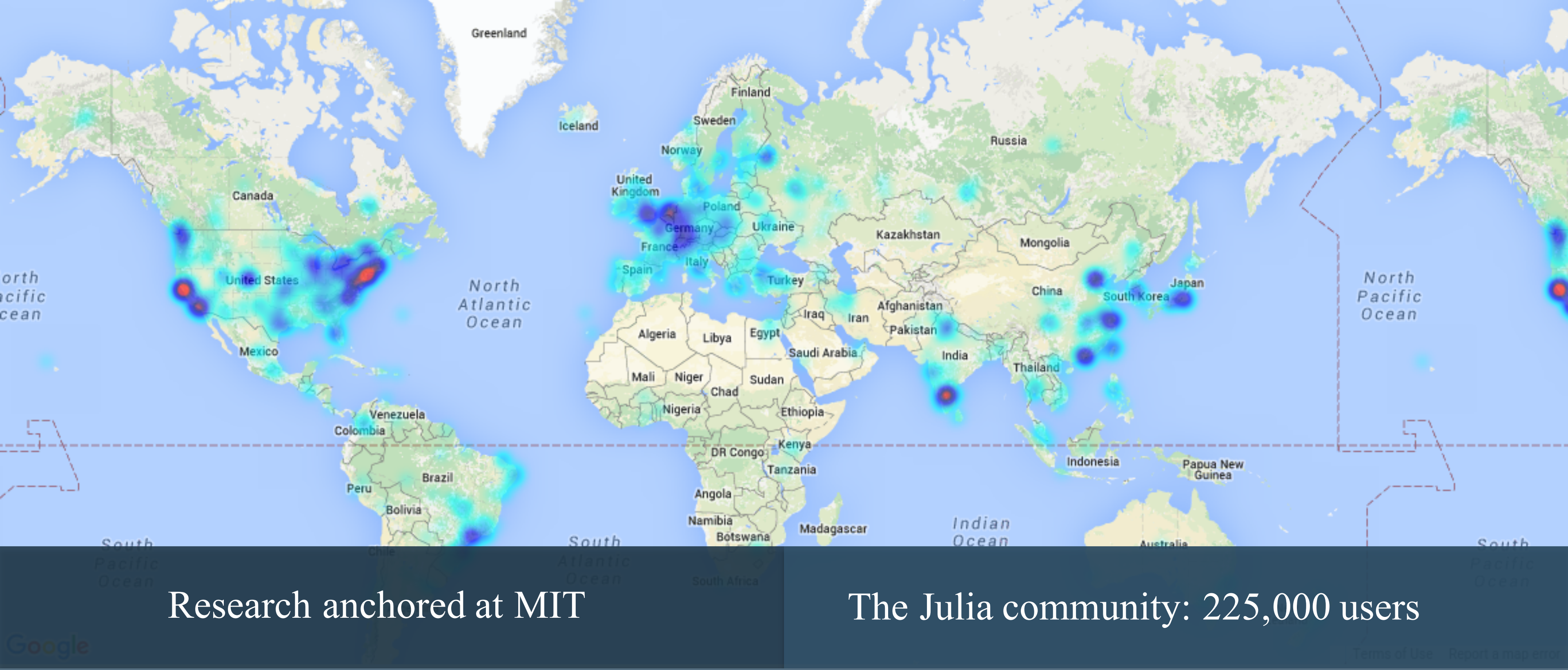
> “ Julia is poised to become one of the leading tools deployed by developers and programmers at banks, hedge funds, regulators and vendors

Anthony Malakian, Waters Technology Magazine

COOPER TIRES

> “ Thank you for Julia. You've kindled serious excitement. I am now working toward replacing some of our computationally intensive Matlab tools with Julia.

Patrick Majors, Engineering Manager, Cooper Tires



Research anchored at MIT

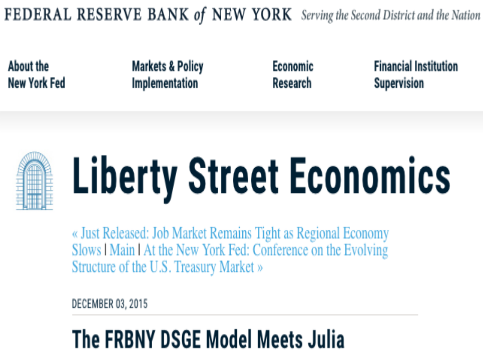
The Julia community: 225,000 users

Expecting to reach 1 million users and 10,000 enterprises by 2019

JuliaCon 2016: 50 talks and 250 attendees

Traction across Industries

FINANCE	ENGINEERING	IOT	3D PRINTING
Economic Models at the NY Fed	Air Collision Avoidance for FAA	Self-driving Cars at UC Berkeley	3D Printing Quadcopters at Voxel8



FEDERAL RESERVE BANK *of* NEW YORK *Serving the Second District and the Nation*

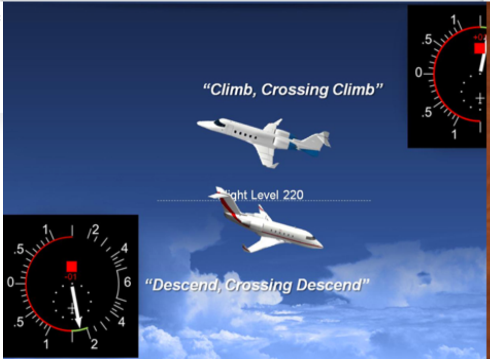
About the New York Fed | Markets & Policy Implementation | Economic Research | Financial Institution Supervision

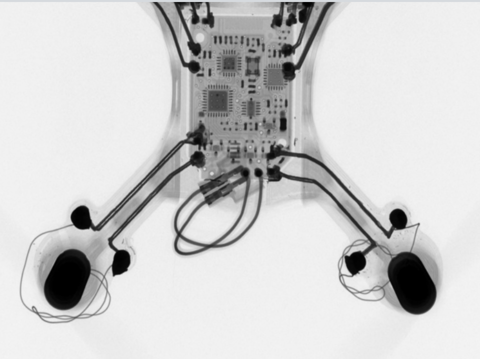
Liberty Street Economics

« Just Released: Job Market Remains Tight as Regional Economy Slows | Main | At the New York Fed: Conference on the Evolving Structure of the U.S. Treasury Market »

DECEMBER 03, 2015

The FRBNY DSGE Model Meets Julia



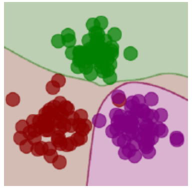


Machine Learning

Machine Learning: Write once, Run everywhere

Many machine learning frameworks

TensorFlow



scikit learn

dmlc mxnet

Mocha.jl

Knet.jl

Merlin.jl

Run on hardware of your choice

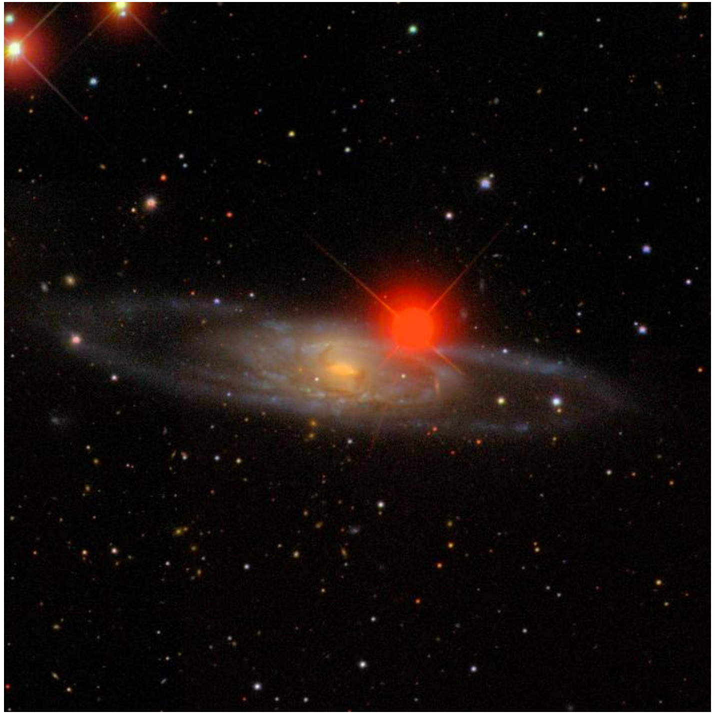
OpenCL

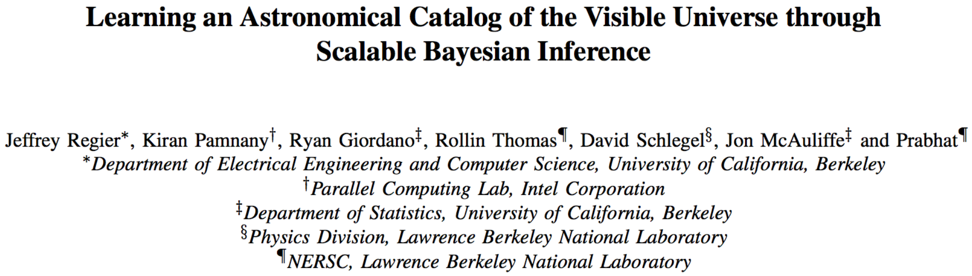
NVIDIA CUDA

ARM

OpenPOWER™

Machine Learning to build a sky atlas on 8000 cores at NERSC





Learning an Astronomical Catalog of the Visible Universe through Scalable Bayesian Inference

Jeffrey Regier[*], Kiran Pamnany[†], Ryan Giordano[‡], Rollin Thomas[¶], David Schlegel[§], Jon McAuliffe[‡] and Prabhat[¶]

[*] *Department of Electrical Engineering and Computer Science, University of California, Berkeley*

[†] *Parallel Computing Lab, Intel Corporation*

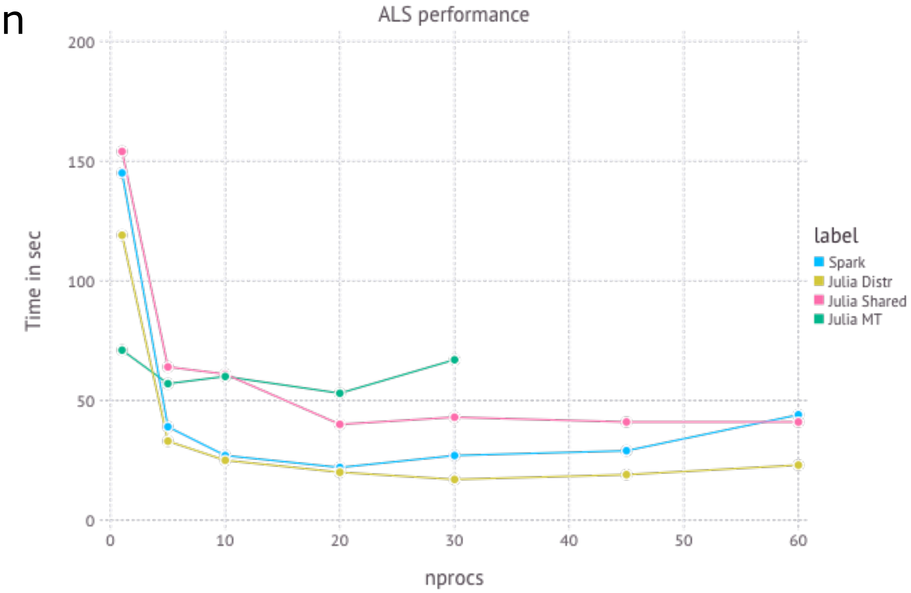
[‡] *Department of Statistics, University of California, Berkeley*

[§] *Physics Division, Lawrence Berkeley National Laboratory*

[¶] *NERSC, Lawrence Berkeley National Laboratory*

Netflix recommendation challenge: Faster than Spark

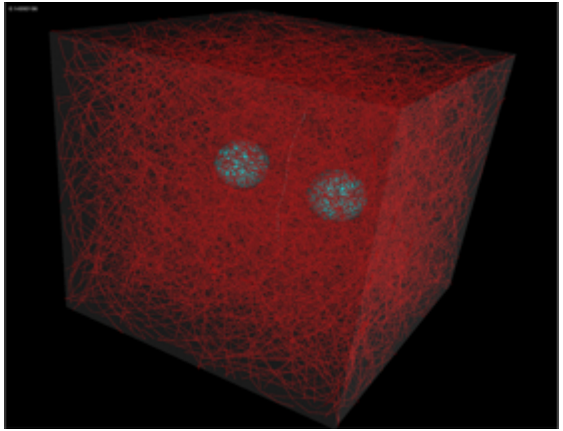
- RecSys.jl - Large movie data set (500 million parameters)
- Distributed Alternating Least Squares SVD-based model executed in Julia and in Spark
- Faster:
  - Original code in Scala
  - Distributed Julia nearly 2x faster than Spark
- Better:
  - Julia code is significantly more readable
  - Easy to maintain and update

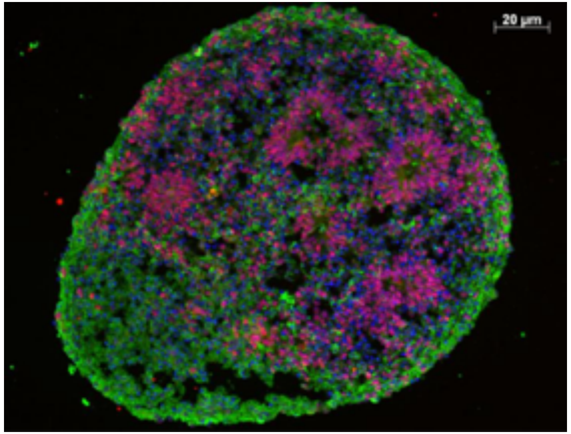


http://juliacomputing.com/blog/2016/04/22/a-parallel-recommendation-engine-in-julia.html

Analytics for Personalized Medicine

- Improving the Quantity and Quality of Information via Microrheology-Based Analytics
- Camera-based real-time particle tracking at KHz rates and Angstrom accuracy
- Real-time organoid analysis leading to precision medicine.
- Julia was the only system that allowed for real-time analysis of instrumentation data

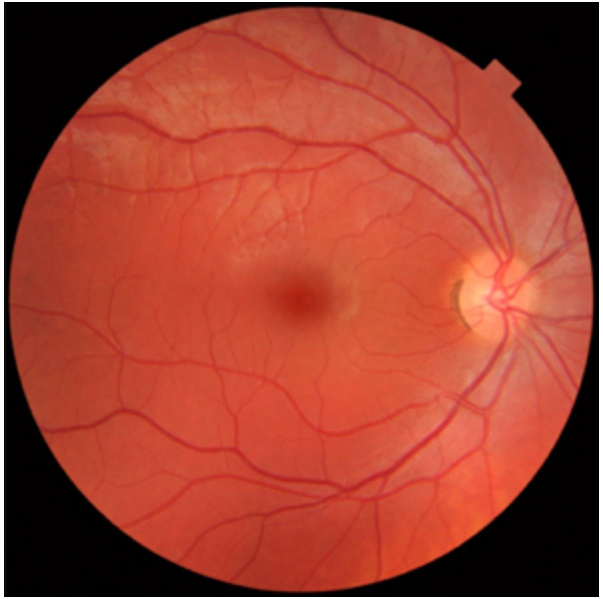




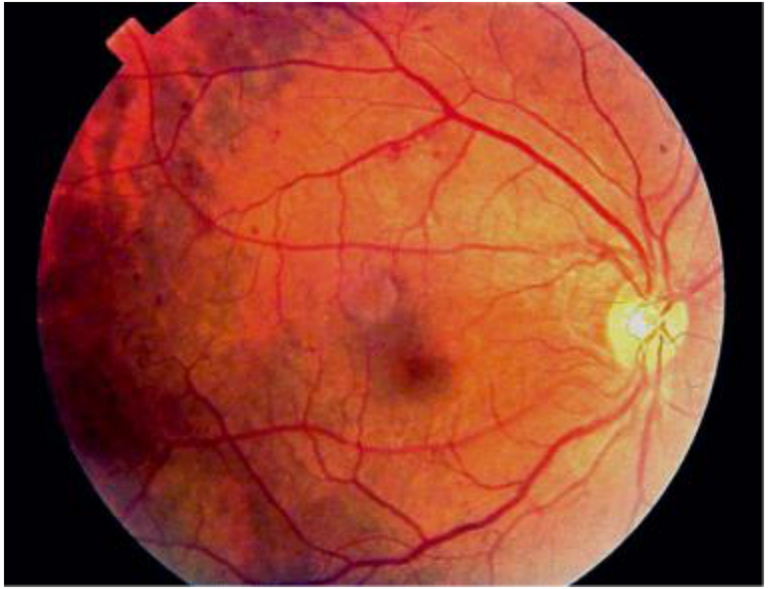
Path BioAnalytics

Deep learning for diabetic retinopathy detection

http://juliacomputing.com/blog/2016/11/16/deep-eyes.html



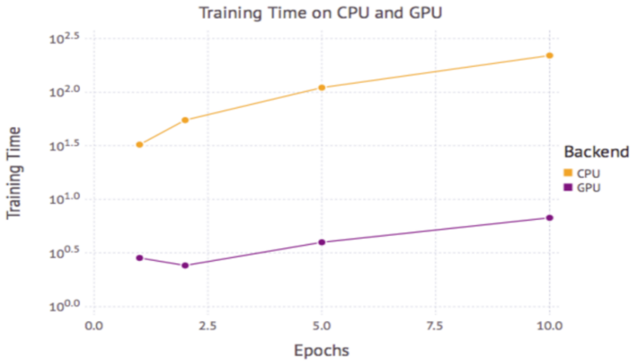
Normal Eye Fundus

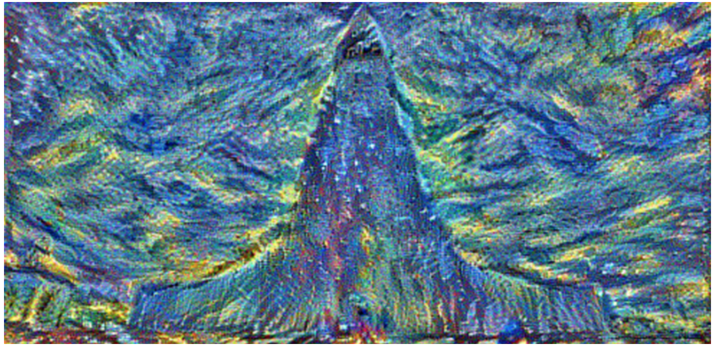


Eye Fundus Infected with Diabetic Retinopathy

Neural style transfer

- Deep learning model with MXNet
- Performance AND expressivity
  - Easy to experiment
- Training on the CPU and GPU
- Explore pre-trained models

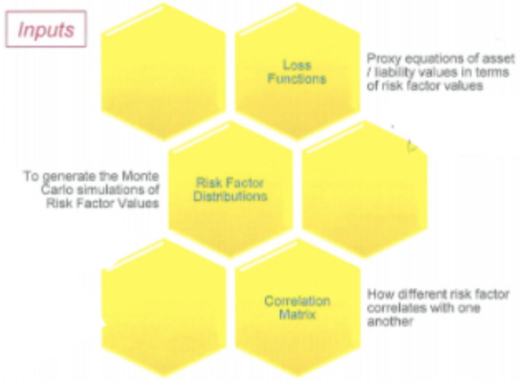


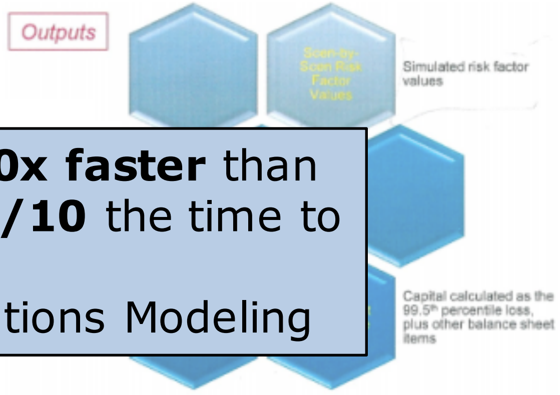


Finance

Solvency II Actuarial Capital Modeling

- Purpose of their Calculation Kernel
  - Calculation of a Solvency II Balance Sheet
- Particularly focuses on the Solvency Capital Requirement
  - Use of Monte Carlo Simulation, currently up to 500,000 scenarios
  - Involves aggregation (summing up legal entities to a Group), ranking and smoothing
- Generates various outputs for downstream reporting


Inputs
Loss Functions
Proxy equations of asset / liability values in terms of risk factor values
To generate the Monte Carlo simulations of Risk Factor Values
Risk Factor Distributions
Correlation Matrix
How different risk factor correlates with one another

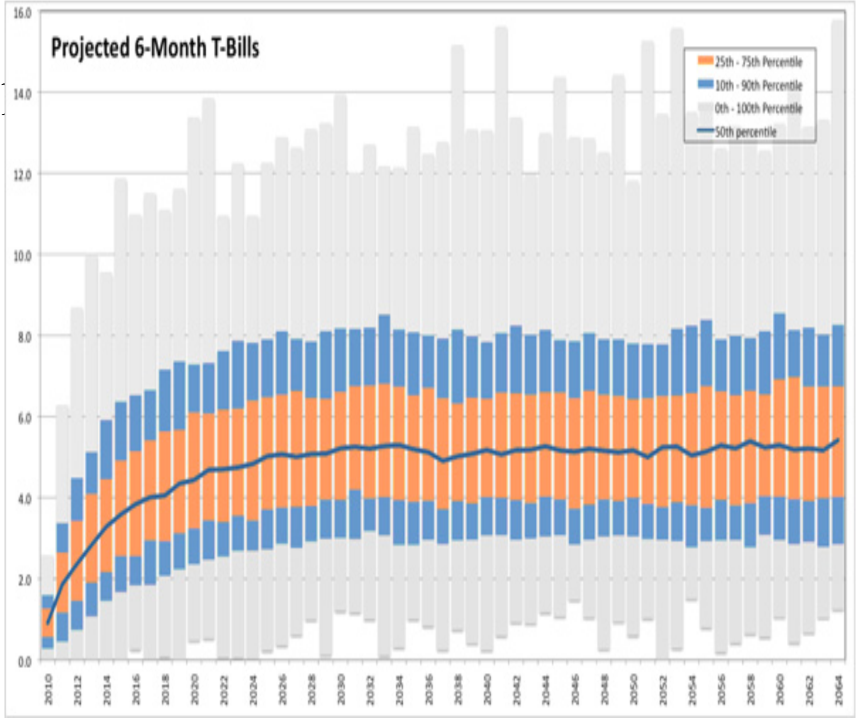

Outputs
Scen-by-Scen Risk Factor Values
Simulated risk factor values
Capital calculated as the 99.5th percentile loss, plus other balance sheet items

> "Solvency II compliant models in Julia are **1000x faster** than IBM Algorithmics, **10x lesser code** and took **1/10** the time to implement"
>
> – Tim Thornham, Director of Financial Solutions Modeling

Economic Scenario Generator

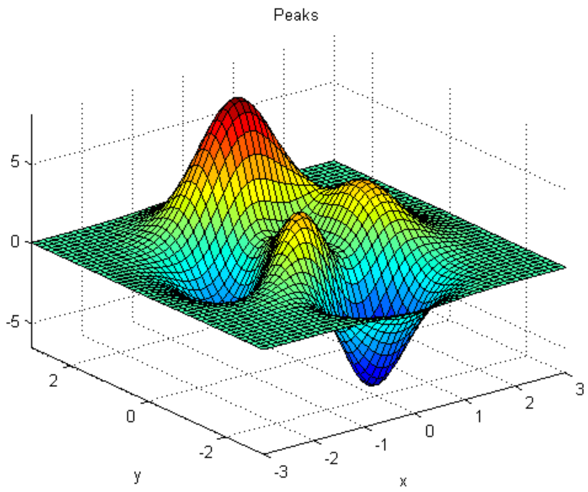
CONNING

- High-dimensional data set on which data extraction, data reordering, and various statistical kernel computations are performed
- Faster:
  - Original code was in K
  - Julia code is 4x-10x faster
- Better:
  - Julia code is significantly more readable
  - Easy to maintain and update
  - Cost-effective



Mathematical Optimization

- Solving a large complex mathematical optimization problem for mortgages
- Full optimization: (Faster Speed + Better Quality)
  - MATLAB 2014a 558.094600 seconds, 3110 iterations
  - Julia v0.4 1.833 seconds, 50 iterations **(300x faster)**
- Performance: Objective function only (100 iterations)
  - MATLAB 2014a 2.69 seconds
  - Julia v0.4 0.78 seconds **(3.5x faster)**
- Quality: Optimization value (11-parameter)
  - MATLAB 2014a 4.277644613116166e+14 (3110 iterations)
  - Julia v0.4 4.270887086707642e+14 **(50 iterations)**



Risk Analytics and Asset Management

BLACKROCK

- BlackRock is using Julia in its flagship Aladdin product:
  - Next generation analytics
  - Risk management
  - Asset management
  - Time series analytics

- Significant gain in productivity and scalability

Asset and Liabilities Modeling at Brazilian Development Bank

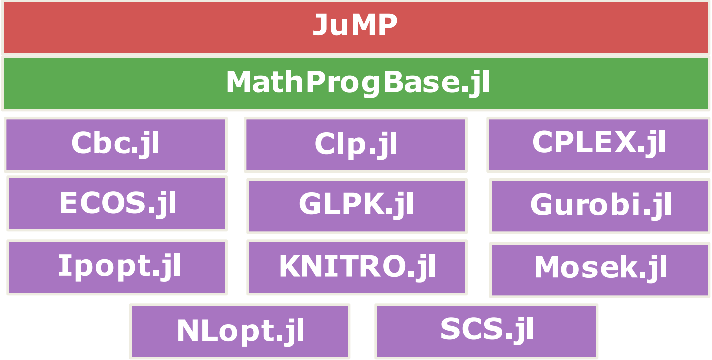
- Manage >$1 Trillion in assets
- Multistage stochastic optimization solution to the bank's returns
  - Choosing the best allocation, funding and hedge decisions
  - Subject to a wide range of business, political and market restrictions

"Selected Julia for its speed, elegance, and JuMP – the Julia Mathematical Optimization Package" - Felipe Tavares

Mathematical Optimization

Solver capabilities accessible through JuMP

Solver	LP	MILP	SOCP	MISOCP	SDP	NLP	MINLP	Other
Bonmin (via AmplNLwriter.jl)	✔	✔				✔	✔	
Cbc (.jl)	✔	✔						
Clp (.jl)	✔							
Couenne (via ApmlNLWriter.jl)	✔	✔				✔	✔	
CPLEX (.jl)	✔	✔	✔	✔				IP callbacks
ECOS (.jl)	✔		✔					
GLPK (.jl)	✔	✔						IP callbacks
Gurobi (.jl)	✔	✔	✔	✔				IP callbacks
Ipopt (.jl)	✔					✔		
Artelys Knitro (.jl)	✔	✔				✔	✔	
Mosek (.jl)	✔	✔	✔	✔	✔	✔[1]		
NLopt (.jl)						✔		
SCS (.jl)	✔		✔		✔			



Key:

- LP = Linear Programming
- MILP = Mixed Integer Linear Programming
- SOCP = Second-order cone programming (includes convex QP and QCQP)
- MISOCP = Mixed Integer SOCP
- SDP = Semidefinite Programming
- NLP = (constrained) Nonlinear Programming (includes general QP and QCQP)
- MINLP = Mixed Integer NLP

Notes:

1. Problem must be convex.

Some JuMP Applications

- Train scheduling
- Self-driving cars
- Electric vehicle charging
- Power grid control
- Plasma physics
- Fantasy sports

> “If you have a choice of several languages, it is, all other things being equal, a mistake to program in anything but the most powerful one.”

Paul Graham in *Beating the Averages*

Co-Founder, Y-Combinator

Simplicity meets Speed

Products that make Julia easy to use, easy to deploy and easy to scale

Julia PRO

Julia BOX

Julia RUN

Julia FIN

Simon Byrne - Julia Computing

What is Julia?

Julia is a modern, high-performance, dynamic programming language for technical computing.

- *modern*: based on the lessons of the past 60 years
- *high-performance*: as fast as traditional "fast" languages (Fortran/C/C++)
- *dynamic*: "simple to use" (R/Matlab/Python)
- *technical computing*: anything involving numbers

Why Julia?

- To write fast, efficient code in an easy, elegant dynamic language
  - Avoids the *two language problem*:

  > My R/Python/Matlab code is too slow; I need to rewrite low-level routines in C/C++/Fortran

- It is easy to "peek under the hood"
  - Most of Julia is written Julia
  - Can inspect various stages of the compilation process
- It's free (download at www.julialang.org)
- It's fun.
- Play nicely with existing tools

In [1]:

```
# accurately compute log(sum(exp(X)))
function logsumexp(X)
    u = maximum(X)
    t = 0.0
    for i = 1:length(X)
        t += exp(X[i]-u)
    end
    u + log(t)
end
```

Out[1]:

```
logsumexp (generic function with 1 method)
```

Syntax heavily influenced by Python and Matlab

Basic differences from Python:

- explicit `end` vs. significant whitespace
- 1-based vs. 0-based arrays

Basic differences from Matlab:

- Functions can be defined anywhere
- Scalars are not matrices in disguise
- `randn(10)` gives you the thing you actually want.

Types

Every object has one:

In [2]:

```
typeof(1.0)
```

Out[2]:

```
Float64
```

In [3]:

```
typeof(logsumexp)
```

Out[3]:

```
#logsumexp
```

In [4]:

```
typeof(Float64)
```

Out[4]:

```
DataType
```

New types are declared with the `type` keyword:

In [5]:

```
type Baz
    a::Float64
    b::Float64
end
```

In [6]:

```
b = Baz(1.0,2.0)
```

Out[6]:

```
Baz(1.0,2.0)
```

Unlike classes in Python/Matlab, user defined types are just as efficient as the builtin types (indeed, most "builtin" types are actually written in Julia)

Generic functions and multiple dispatch

Julia functions are *generic* in that different code paths can be called depending on the type arguments.

In [7]:

```
f(x::Float64) = "$x is a float" # "$" does string substitution
f(x::Int) = "$x is an integer"
```

Out[7]:

```
f (generic function with 2 methods)
```

- `f(...) = ...` is the same as `function f(...) ... end`
- `::` is an optional *type specification*.

In [8]:

```
f(1.0)
```

Out[8]:

```
"1.0 is a float"
```

In [9]:

```
f(1)
```

Out[9]:

```
"1 is an integer"
```

Unlike traditional object oriented languages (C++, Python, Matlab), functions don't "belong" to a type. This allows for *multiple dispatch* on any combination of arguments.

In [10]:

```
f(x::Float64,y::Int) = "$x is a float, but $y is an integer"
f(x::Real,y::Real) = "$x and $y are both some sort of real" # Real is an abstract "super" type
f(x,y) = "I don't know what $x and $y are" # fallback
```

Out[10]:

```
f (generic function with 5 methods)
```

In [11]:

```
f(1.0,1)
```

Out[11]:

```
"1.0 is a float, but 1 is an integer"
```

In [12]:

```
f(1,1)
```

Out[12]:

```
"1 and 1 are both some sort of real"
```

In [13]:

```
f("aaa",2)
```

Out[13]:

```
"I don't know what aaa and 2 are"
```

Say we want to change how `Baz` is printed, this is handled by the `show` function:

In [14]:

```
show(b)
```

```
Baz(1.0,2.0)
```

`show` is a generic function: it is made up of different methods for diffent *type signature*s:

In [15]:

```
methods(show)
```

Out[15]:

199 methods for generic function **show**:

- show(io::**IO**, opt::**Base.JLOptions**) at options.jl:42 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, r::**LinSpace**) at range.jl:257 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, r::**UnitRange**) at range.jl:548 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, r::**Base.OneTo**) at range.jl:549 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, r::**Range**) at range.jl:547 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, z::**Complex{Bool}**) at complex.jl:83 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, z::**Complex**) at complex.jl:68 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, x::**Rational**) at rational.jl:47 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, s::**IntSet**) at intset.jl:16 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**Base.EnvHash**) at env.jl:133 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show*{K,V}*(io::**IO**, t::**Associative{K,V}**) at dict.jl:52 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, iter::**Union{Base.KeyIterator,Base.ValueIterator}**) at dict.jl:93 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, s::**Set**) at set.jl:22 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, info::**Base.Sys.CPUinfo**) at sysinfo.jl:91 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, s::**IOStream**) at iostream.jl:28 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, b::**Base.AbstractIOBuffer**) at iobuffer.jl:45 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, c::**Char**) at char.jl:50 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, exc::**UnicodeError**) at strings/errors.jl:14 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, s::**Base.SubstitutionString**) at regex.jl:236 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, s::**AbstractString**) at strings/io.jl:72 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show*{S}*(io::**IO**, g::**Base.UTF8proc.GraphemeIterator{S}**) at strings/utf8proc.jl:235 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, re::**Regex**) at regex.jl:86 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, m::**RegexMatch**) at regex.jl:116 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ctx::**IOContext**) at show.jl:72 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, f::**Function**) at show.jl:163

(https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b

- show(io::**IO**, x::**Core.IntrinsicFunction**) at show.jl:173 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, x::**Union**) at show.jl:177 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, x::**TypeConstructor**) at show.jl:182 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, t::**Type{Base.WorkerState}**) at Enums.jl:105 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, t::**Type{Base.LibGit2.Consts.GIT_MERGE}**) at Enums.jl:105 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, t::**Type{Base.LibGit2.Consts.GIT_MERGE_FILE}**) at Enums.jl:105 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, t::**Type{Base.LibGit2.Consts.GIT_MERGE_FILE_FAVOR}**) at Enums.jl:105 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, t::**Type{Base.LibGit2.Consts.GIT_MERGE_PREFERENCE}**) at Enums.jl:105 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, t::**Type{Base.LibGit2.Consts.GIT_MERGE_ANALYSIS}**) at Enums.jl:105 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, t::**Type{Base.LibGit2.Consts.GIT_SUBMODULE_IGNORE}**) at Enums.jl:105 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, t::**Type{Base.LibGit2.Consts.GIT_REPOSITORY_OPEN}**) at Enums.jl:105 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, t::**Type{Base.LibGit2.Consts.GIT_BRANCH}**) at Enums.jl:105 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, t::**Type{Base.LibGit2.Consts.GIT_FILEMODE}**) at Enums.jl:105 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, t::**Type{Base.LibGit2.Consts.GIT_CREDTYPE}**) at Enums.jl:105 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, t::**Type{Base.LibGit2.Consts.GIT_FEATURE}**) at Enums.jl:105 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, t::**Type{Base.LibGit2.Consts.GIT_CONFIG}**) at Enums.jl:105 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, t::**Type{Base.LibGit2.Consts.GIT_OPT}**) at Enums.jl:105 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, t::**Type{Base.LibGit2.Error.Code}**) at Enums.jl:105 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, t::**Type{Base.LibGit2.Error.Class}**) at Enums.jl:105 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, x::**DataType**) at show.jl:192 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, tn::**TypeName**) at show.jl:225 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**Void**) at show.jl:232 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, b::**Bool**) at show.jl:233 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b

- show(io::**IO**, n::**Signed**) at show.jl:234
  (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, n::**Unsigned**) at show.jl:235
  (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show*{T}*(io::**IO**, p::**Ptr{T}**) at show.jl:238
  (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, p::**Pair**) at show.jl:241
  (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, m::**Module**) at show.jl:257
  (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, l::**LambdaInfo**) at show.jl:285
  (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, t::**Tuple**) at show.jl:376
  (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, v::**SimpleVector**) at show.jl:377
  (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, s::**Symbol**) at show.jl:379
  (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ex::**Union{Expr,GlobalRef,GotoNode,LabelNode,LineNumberNode,QuoteNode,Slot}**) at show.jl:411
  (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, tv::**TypeVar**) at show.jl:1049
  (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, M::**Bidiagonal**) at linalg/bidiag.jl:173
  (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, S::**SparseMatrixCSC**) at sparse/sparsematrix.jl:88
  (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, x::**AbstractSparseArray{Tv<:Any,Ti<:Any,1}**) at sparse/sparsevector.jl:683
  (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, FC::**Base.SparseArrays.CHOLMOD.FactorComponent**) at sparse/cholmod.jl:1084
  (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, X::**AbstractArray**) at show.jl:1586
  (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show*{T}*(io::**IO**, x::**Nullable{T}**) at nullable.jl:31
  (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, v::**VersionNumber**) at version.jl:64
  (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, e::**Base.UVError**) at libuv.jl:69
  (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, t::**Task**) at task.jl:50
  (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, stream::**Base.LibuvServer**) at stream.jl:203
  (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, s::**BufferStream**) at stream.jl:1047
  (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, stream::**UDPSocket**) at socket.jl:362
  (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, stream::**Base.LibuvStream**) at stream.jl:204

(https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b

- show(io::**IO**, stream::**Pipe**) at stream.jl:562 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ip::**IPv4**) at socket.jl:37 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ip::**IPv6**) at socket.jl:91 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, err::**Base.DNSError**) at socket.jl:537 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, st::**Base.Filesystem.StatStruct**) at stat.jl:60 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, cmd::**Cmd**) at process.jl:103 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, cmds::**Union{Base.ErrOrCmds,Base.OrCmds}**) at process.jl:121 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, cmds::**Base.AndCmds**) at process.jl:130 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, cr::**Base.CmdRedirect**) at process.jl:170 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, p::**Base.Process**) at process.jl:719 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/html")}**, m::**Method**; *kwtype*) at methodshow.jl:199 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, mime::**MIME{Symbol("text/html")}**, ms::**Base.MethodList**) at methodshow.jl:245 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, mime::**MIME{Symbol("text/html")}**, mt::**MethodTable**) at methodshow.jl:261 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, mime::**MIME{Symbol("text/html")}**, mt::**AbstractArray{Method,1}**) at methodshow.jl:266 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/plain")}**, t::**Type{Base.WorkerState}**) at Enums.jl:108 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/csv")}**, a) at datafmt.jl:712 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/tab-separated-values")}**, a) at datafmt.jl:713 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/plain")}**, iter::**Union{Base.KeyIterator,Base.ValueIterator}**) at replutil.jl:8 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show*{K,V}*(io::**IO**, ::**MIME{Symbol("text/plain")}**, t::**Associative{K,V}**) at replutil.jl:39 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/plain")}**, f::**Function**) at replutil.jl:100 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/plain")}**, l::**LambdaInfo**) at replutil.jl:117 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/plain")}**, r::**LinSpace**) at replutil.jl:138 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b

- show(io::**IO**, ::**MIME{Symbol("text/plain")}**, t::**Task**) at replutil.jl:146 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/plain")}**, r::**Range**) at replutil.jl:154 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/plain")}**, S::**SparseMatrixCSC**) at sparse/sparsematrix.jl:80 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/plain")}**, x::**AbstractSparseArray{Tv<:Any,Ti<:Any,1}**) at sparse/sparsevector.jl:677 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/plain")}**, FC::**Base.SparseArrays.CHOLMOD.FactorComponent**) at sparse/cholmod.jl:1097 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/plain")}**, X::**AbstractArray**) at replutil.jl:153 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/plain")}**, t::**Type{Base.LibGit2.Consts.GIT_MERGE}**) at Enums.jl:108 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/plain")}**, t::**Type{Base.LibGit2.Consts.GIT_MERGE_FILE}**) at Enums.jl:108 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/plain")}**, t::**Type{Base.LibGit2.Consts.GIT_MERGE_FILE_FAVOR}**) at Enums.jl:108 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/plain")}**, t::**Type{Base.LibGit2.Consts.GIT_MERGE_PREFERENCE}**) at Enums.jl:108 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/plain")}**, t::**Type{Base.LibGit2.Consts.GIT_MERGE_ANALYSIS}**) at Enums.jl:108 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/plain")}**, t::**Type{Base.LibGit2.Consts.GIT_SUBMODULE_IGNORE}**) at Enums.jl:108 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/plain")}**, t::**Type{Base.LibGit2.Consts.GIT_REPOSITORY_OPEN}**) at Enums.jl:108 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/plain")}**, t::**Type{Base.LibGit2.Consts.GIT_BRANCH}**) at Enums.jl:108 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/plain")}**, t::**Type{Base.LibGit2.Consts.GIT_FILEMODE}**) at Enums.jl:108 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/plain")}**, t::**Type{Base.LibGit2.Consts.GIT_CREDTYPE}**) at Enums.jl:108 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/plain")}**, t::**Type{Base.LibGit2.Consts.GIT_FEATURE}**) at Enums.jl:108 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/plain")}**, t::**Type{Base.LibGit2.Consts.GIT_CONFIG}**) at Enums.jl:108

(https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b

- show(io::**IO**, ::**MIME{Symbol("text/plain")}**, t::**Type{Base.LibGit2.Consts.GIT_OPT}**) at Enums.jl:108 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/plain")}**, t::**Type{Base.LibGit2.Error.Code}**) at Enums.jl:108 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/plain")}**, t::**Type{Base.LibGit2.Error.Class}**) at Enums.jl:108 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/plain")}**, F::**Base.SparseArrays.CHOLMOD.Factor**) at sparse/cholmod.jl:1098 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/plain")}**, x) at replutil.jl:4 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/markdown")}**, md::**Base.Markdown.MD**) at markdown/render/plain.jl:140 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/html")}**, md::**Base.Markdown.MD**) at markdown/render/html.jl:188 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/latex")}**, md::**Base.Markdown.HorizontalRule**) at markdown/render/latex.jl:103 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/latex")}**, md::**Base.Markdown.MD**) at markdown/render/latex.jl:171 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/rst")}**, md::**Base.Markdown.MD**) at markdown/render/rst.jl:145 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show*{F<:Function}*(io::**IO**, ::**MIME{Symbol("text/html")}**, h::**HTML{F}**) at docs/utils.jl:35 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ::**MIME{Symbol("text/html")}**, h::**HTML**) at docs/utils.jl:34 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show*{mime}*(io::**IO**, ::**MIME{mime}**) at multimedia.jl:18 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, m::**AbstractString**, x) at multimedia.jl:33 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, x::**Union{Float32,Float64}**) at grisu/grisu.jl:120 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, x::**Float16**) at grisu/grisu.jl:128 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, m::**Method**; *kwtype*) at methodshow.jl:74 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ms::**Base.MethodList**) at methodshow.jl:149 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, mt::**MethodTable**) at methodshow.jl:150 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, x::**BigInt**) at gmp.jl:514 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, b::**BigFloat**) at mpfr.jl:869

(https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b

- show(io::**IO**, u::**Base.Random.UUID**) at random.jl:1320 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, c::**Channel**) at channels.jl:106 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, x::**Base.WorkerState**) at Enums.jl:96 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, manager::**Base.SSHManager**) at managers.jl:139 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, manager::**Base.LocalManager**) at managers.jl:309 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ex::**Base.PrecompilableError**) at loading.jl:266 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, t::**Base.Test.Pass**) at test.jl:46 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, t::**Base.Test.Fail**) at test.jl:71 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, t::**Base.Test.Error**) at test.jl:103 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, t::**Base.Test.Broken**) at test.jl:141 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ex::**Base.Test.TestSetException**) at test.jl:374 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, s::**Base.LineEdit.MIState**) at LineEdit.jl:33 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, x::**Base.LineEdit.Prompt**) at LineEdit.jl:52 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, s::**Base.LineEdit.PrefixSearchState**) at LineEdit.jl:1038 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show*{T,S<:AbstractArray{T,2}}*(io::**IO**, C::**Base.LinAlg.Cholesky{T,S}**) at linalg/cholesky.jl:376 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, J::**UniformScaling**) at linalg/uniformscaling.jl:21 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show*{sym}*(io::**IO**, x::**Irrational{sym}**) at irrationals.jl:7 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, p::**Base.DFT.ScaledPlan**) at dft.jl:252 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show*{T,K,inplace}*(io::**IO**, p::**Base.DFT.FFTW.cFFTWPlan{T,K,inplace,N<:Any}**) at fft/FFTW.jl:289 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show*{T,K,inplace}*(io::**IO**, p::**Base.DFT.FFTW.rFFTWPlan{T,K,inplace,N<:Any}**) at fft/FFTW.jl:296 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show*{T,K,inplace}*(io::**IO**, p::**Base.DFT.FFTW.r2rFFTWPlan{T,K,inplace,N<:Any}**) at fft/FFTW.jl:304 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show*{T,K,inplace}*(io::**IO**, p::**Base.DFT.FFTW.DCTPlan{T,K,inplace}**) at fft/dct.jl:24 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, x::**Base.LibGit2.Consts.GIT_MERGE**) at Enums.jl:96 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, x::**Base.LibGit2.Consts.GIT_MERGE_FILE**) at Enums.jl:96

(https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b

- show(io::**IO**, x::**Base.LibGit2.Consts.GIT_MERGE_FILE_FAVOR**) at Enums.jl:96 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, x::**Base.LibGit2.Consts.GIT_MERGE_PREFERENCE**) at Enums.jl:96 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, x::**Base.LibGit2.Consts.GIT_MERGE_ANALYSIS**) at Enums.jl:96 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, x::**Base.LibGit2.Consts.GIT_SUBMODULE_IGNORE**) at Enums.jl:96 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, x::**Base.LibGit2.Consts.GIT_REPOSITORY_OPEN**) at Enums.jl:96 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, x::**Base.LibGit2.Consts.GIT_BRANCH**) at Enums.jl:96 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, x::**Base.LibGit2.Consts.GIT_FILEMODE**) at Enums.jl:96 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, x::**Base.LibGit2.Consts.GIT_CREDTYPE**) at Enums.jl:96 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, x::**Base.LibGit2.Consts.GIT_FEATURE**) at Enums.jl:96 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, x::**Base.LibGit2.Consts.GIT_CONFIG**) at Enums.jl:96 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, x::**Base.LibGit2.Consts.GIT_OPT**) at Enums.jl:96 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, ie::**Base.LibGit2.IndexEntry**) at libgit2/types.jl:502 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, rbo::**Base.LibGit2.RebaseOperation**) at libgit2/types.jl:539 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, x::**Base.LibGit2.Error.Code**) at Enums.jl:96 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, x::**Base.LibGit2.Error.Class**) at Enums.jl:96 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, err::**Base.LibGit2.Error.GitError**) at libgit2/error.jl:71 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, id::**Base.LibGit2.Oid**) at libgit2/oid.jl:71 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, i::**Base.Pkg.Types.VersionInterval**) at pkg/types.jl:15 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, s::**Base.Pkg.Types.VersionSet**) at pkg/types.jl:39 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, a::**Base.Pkg.Types.Available**) at pkg/types.jl:73 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, f::**Base.Pkg.Types.Fixed**) at pkg/types.jl:86 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, frame::**StackFrame**; *full_path*) at stacktraces.jl:204 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, x::**Base.Dates.Period**) at dates/periods.jl:45 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, x::**Base.Dates.CompoundPeriod**) at dates/periods.jl:308 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, df::**Base.Dates.DateFunction**) at dates/adjusters.jl:140 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/b
- show(io::**IO**, x::**DateTime**) at dates/io.jl:16

(https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/ba

- show(io::**IO**, x::**Date**) at dates/io.jl:24 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/ba
- show(io::**IO**, F::**Base.SparseArrays.UMFPACK.UmfpackLU**) at sparse/umfpack.jl:177 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/ba
- show(io::**IO**, F::**Base.SparseArrays.CHOLMOD.Factor**) at sparse/cholmod.jl:1079 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/ba
- show(io::**IO**, tex::**Base.Markdown.LaTeX**) at markdown/IPython/IPython.jl:25 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/ba
- show(io::**IO**, md::**Base.Markdown.MD**) at markdown/render/plain.jl:139 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/ba
- show(io::**IO**, b::**Base.Docs.Binding**) at docs/bindings.jl:35 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/ba
- show(io::**IO**, t::**Text**) at docs/utils.jl:73 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/ba
- show(io::**IO**, x::**Nettle.HashType**) at /Users/simon/.julia/v0.5/Nettle/src/hash.jl:51 (https://github.com/staticfloat/Nettle.jl/tree/f20cbb3dfc7c31eb0cce3c3f4b05c53f92c58c3
- show(io::**IO**, x::**Nettle.Hasher**) at /Users/simon/.julia/v0.5/Nettle/src/hash.jl:56 (https://github.com/staticfloat/Nettle.jl/tree/f20cbb3dfc7c31eb0cce3c3f4b05c53f92c58c3
- show(io::**IO**, x::**Nettle.HMACState**) at /Users/simon/.julia/v0.5/Nettle/src/hmac.jl:53 (https://github.com/staticfloat/Nettle.jl/tree/f20cbb3dfc7c31eb0cce3c3f4b05c53f92c58c3
- show(io::**IO**, x::**Nettle.CipherType**) at /Users/simon/.julia/v0.5/Nettle/src/cipher.jl:287 (https://github.com/staticfloat/Nettle.jl/tree/f20cbb3dfc7c31eb0cce3c3f4b05c53f92c58c3
- show(io::**IO**, x::**Nettle.Encryptor**) at /Users/simon/.julia/v0.5/Nettle/src/cipher.jl:292 (https://github.com/staticfloat/Nettle.jl/tree/f20cbb3dfc7c31eb0cce3c3f4b05c53f92c58c3
- show(io::**IO**, x::**Nettle.Decryptor**) at /Users/simon/.julia/v0.5/Nettle/src/cipher.jl:293 (https://github.com/staticfloat/Nettle.jl/tree/f20cbb3dfc7c31eb0cce3c3f4b05c53f92c58c3
- show(io::**IO**, msg::**IJulia.Msg**) at /Users/simon/.julia/v0.5/IJulia/src/msg.jl:42 (https://github.com/JuliaLang/IJulia.jl/tree/78106bcd813041fa8ed87a2ff343145c4d33d02
- show(io::**IO**, x::**ANY<:Any**) at show.jl:116 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/ba
- show(x) at coreio.jl:3 (https://github.com/JuliaLang/julia/tree/3c9d75391c72d7c32eea75ff187ce77b2d5effc8/ba

Most `show` methods take an `IO` object as a first argument

- this allows writing to different places (`STDOUT`, buffers, files, etc.)

There is also a generic single argument method

```
show(x) = show(STDOUT, x)
```

which prints to `STDOUT` by default.

In [16]:

```
import Base.show # we need to import to extend

show(io::IO, b::Baz) = print(io, "This is a Baz object, with a=$(b.a) and b=$(b.
b).")
```

Out[16]:

```
show (generic function with 200 methods)
```

In [17]:

```
b
```

Out[17]:

```
Baz(1.0,2.0)
```

This is incredibly powerful: for example, we can define different matrix multiplication methods for different combinations of arguments (e.g. symmetric, triangular, sparse, etc.)

Just-in-time (JIT) compilation

Julia uses JIT compilation, using the LLVM backend (used by Clang, Rust, Swift).

Functions are the unit at which JIT compilation occurs.

> **Perf tip 1**: Put code inside a function.

Compilation occurs for each type signature.

One of the main tricks used is *type inference*: try to figure out the type of each expression.

> **Perf tip 2**: Try to write *type-stable* functions: for a given combination of input types, variables should not change type.

e.g. this is why:

In [18]:

```
sqrt(-1.0)
```

```
DomainError:
sqrt will only return a complex result if called with a complex argu
ment. Try sqrt(complex(x)).

 in sqrt(::Float64) at ./math.jl:209
 in sqrt(::Float64) at /Applications/Julia-0.5.app/Contents/Resource
s/julia/lib/julia/sys.dylib:?
```

`sqrt(x::Float64)` is known to return values of type `Float64`.

To get a `Complex` number you need to explicitly convert it beforehand:

In [19]:

```
sqrt(complex(-1.0))
```

Out[19]:

```
0.0 + 1.0im
```

Custom numeric types

Julia integers are machine numbers, so can overflow

In [20]:

```
typemax(Int)
```

Out[20]:

```
9223372036854775807
```

In [21]:

```
typemax(Int) + 1
```

Out[21]:

```
-9223372036854775808
```

Can be avoided by promoting to `BigInt` (arbitrary-precision integer)

In [22]:

```
big(typemax(Int)) + 1
```

Out[22]:

```
9223372036854775808
```

Also supports rational arithmetic, via `//` operator:

In [23]:

```
1//3 + 7//6
```

Out[23]:

```
3//2
```

In [24]:

```
1//10 < 0.1
```

Out[24]:

```
true
```

`BigFloat` for high-precision calculations:

In [25]:

```
sin(big(1.0))
```

Out[25]:

```
8.414709848078965066525023216302989996225630607983710656727517099919
104043912398e-01
```

Generic linear algebra routines for non-BLAS types (`Float32`,`Float64`, complex versions thereof). For example, high-precision using `BigFloats`:

In [26]:

```
big(randn(10,10)) \ big(randn(10))
```

Out[26]:

```
10-element Array{BigFloat,1}:
 -1.8934308673726218855379404092631527298028839654055031055207684687
61361588959412
 -4.5288431265516172210106813497979471341534613682460020231307583837
85366718219426e-01
  1.7903947939337657715163323919793961782450352997418229586558651962
64946133640856
  9.87167733446867356900432479160720222413065316024952516508663648905
4840036818478e-01
 -5.3959025586427733592524092653255195740403433817604805955513413498
77625053870542e-01
 -4.2650271017148351344933963624658313132009346784643411684358882762
54605518854225
 -3.3522031729079112342914703986373484662363079555367812226001461502
0295947140504e-01
 -1.3338092019633109423500591649414455702403624899564950416009502477
72572496356217
  2.6745871530584061336517390999318244825595910478156227931857259183
32525152995458
 -5.0948559627958882123497826210569713153600002005635692754491820284
54310263504912e-01
```

Exact linear algebra using rational arithmetic:

In [27]:

```
X = rand(big(1:10),5,5) .// rand(1:10,5,5)
```

Out[27]:

```
5×5 Array{Rational{BigInt},2}:
 1//2  2//5  5//3  9//8  7//8
 9//8  5//2  8//7  2//9  2//3
 1//9  3//8  2//3  2//7  7//6
 3//1  3//2  3//2  3//2  4//5
 4//5  1//4  1//2  1//2  1//10
```

In [28]:

```
X \ (rand(1:10,5) .// rand(1:10,5))
```

Out[28]:

```
5-element Array{Rational{BigInt},1}:
   631275792//628207099
 -4406880280//5653863891
  -358805524//1884621297
  -646586626//628207099
  4202655562//1884621297
```

Metaprogramming

Julia has extensive support of *metaprogramming*: writing code that generates other code.

In [29]:

```
# ":" quotes an expression, which is itself a Julia objects
ex = :(sin(x)+2)
```

Out[29]:

```
:(sin(x) + 2)
```

In [30]:

```
typeof(ex)
```

Out[30]:

```
Expr
```

Macros transform expressions, and are prefixed with `@`:

In [31]:

```
@time logsumexp(rand(100)) # prints time to run a function
```

```
  0.064978 seconds (28.65 k allocations: 1.289 MB)
```

Out[31]:

```
5.132857742727078
```

In [32]:

```
@time logsumexp(rand(100)) # first run is slower due to JIT compilation
```

```
  0.000009 seconds (7 allocations: 1.063 KB)
```

Out[32]:

```
5.175616595212243
```

In [33]:

```
macroexpand(:(@time logsumexp(rand(100))))
```

Out[33]:

```
quote  # util.jl, line 182:
    local #16#stats = (Base.gc_num)() # util.jl, line 183:
    local #18#elapsedtime = (Base.time_ns)() # util.jl, line 184:
    local #17#val = logsumexp(rand(100)) # util.jl, line 185:
    #18#elapsedtime = (Base.time_ns)() - #18#elapsedtime # util.jl,
 line 186:
    local #19#diff = (Base.GC_Diff)((Base.gc_num)(),#16#stats) # uti
l.jl, line 187:
    (Base.time_print)(#18#elapsedtime,#19#diff.allocd,#19#diff.total
_time,(Base.gc_alloc_count)(#19#diff)) # util.jl, line 189:
    #17#val
end
```

Note: For more rigorous benchmarking, use `@benchmark` in the BenchmarkTools.jl package.

Various macros provide the ability to peek inside the compilation process

```
In [34]:
```

```
@code_typed logsumexp(rand(100)) # type inference
```

```
Out[34]:
```

```
LambdaInfo for logsumexp(::Array{Float64,1})
:(begin
        u = $(Expr(:invoke, LambdaInfo for _mapreduce(::Base.#identi
ty, ::Base.#scalarmax, ::Base.LinearFast, ::Array{Float64,1}), :(Bas
e._mapreduce), :(Base.identity), :(Base.scalarmax), :($(Expr(:new, :
(Base.LinearFast)))), :(X))) # line 3:
        t = 0.0 # line 4:
        SSAValue(2) = (Base.arraylen)(X)::Int64
        SSAValue(7) = (Base.select_value)((Base.sle_int)(1,SSAValue
(2))::Bool,SSAValue(2),(Base.box)(Int64,(Base.sub_int)(1,1)))::Int64
        #temp# = 1
        8:
        unless (Base.box)(Base.Bool,(Base.not_int)((#temp# === (Bas
e.box)(Int64,(Base.add_int)(SSAValue(7),1)))::Bool)) goto 20
        SSAValue(8) = #temp#
        SSAValue(9) = (Base.box)(Int64,(Base.add_int)(#temp#,1))
        i = SSAValue(8)
        #temp# = SSAValue(9) # line 5:
        SSAValue(3) = (Base.box)(Base.Float64,(Base.sub_float)((Bas
e.arrayref)(X,i)::Float64,u))
        SSAValue(5) = (Core.ccall)((Core.tuple)("exp",Base.Math.lib
m)::Tuple{String,String},Base.Math.Float64,(Core.svec)(Base.Math.Flo
at64)::SimpleVector,SSAValue(3),0)::Float64
        t = (Base.box)(Base.Float64,(Base.add_float)(t,SSAValue(5)))
        18:
        goto 8
        20:  # line 7:
        SSAValue(6) = $(Expr(:invoke, LambdaInfo for log(::Float64),
:(Main.log), :(t)))
        return (Base.box)(Base.Float64,(Base.add_float)(u,SSAValue
(6)))
    end::Float64)
```

In [35]:

```
@code_llvm logsumexp(rand(100)) # LLVM intermediate representation (IR)
```

```
define double @julia_logsumexp_72099(%jl_value_t*) #0 {
top:
  %1 = call double @julia__mapreduce_72101(%jl_value_t* %0) #0
  %2 = getelementptr inbounds %jl_value_t, %jl_value_t* %0, i64 1
  %3 = bitcast %jl_value_t* %2 to i64*
  %4 = load i64, i64* %3, align 8
  %5 = icmp slt i64 %4, 1
  br i1 %5, label %L2, label %if.lr.ph

if.lr.ph:                                         ; preds = %top
  %6 = bitcast %jl_value_t* %0 to double**
  br label %if

L2.loopexit:                                      ; preds = %idxend
  br label %L2

L2:                                               ; preds = %L2.loop
exit, %top
  %t.0.lcssa = phi double [ 0.000000e+00, %top ], [ %19, %L2.loopexi
t ]
  %7 = call double @julia_log_71661(double %t.0.lcssa) #0
  %8 = fadd double %1, %7
  ret double %8

if:                                               ; preds = %if.lr.p
h, %idxend
  %t.06 = phi double [ 0.000000e+00, %if.lr.ph ], [ %19, %idxend ]
  %"#temp#.05" = phi i64 [ 1, %if.lr.ph ], [ %13, %idxend ]
  %9 = add i64 %"#temp#.05", -1
  %10 = load i64, i64* %3, align 8
  %11 = icmp ult i64 %9, %10
  br i1 %11, label %idxend, label %oob

oob:                                              ; preds = %if
  %12 = alloca i64, align 8
  store i64 %"#temp#.05", i64* %12, align 8
  call void @jl_bounds_error_ints(%jl_value_t* %0, i64* nonnull %12,
i64 1)
  unreachable

idxend:                                           ; preds = %if
  %13 = add i64 %"#temp#.05", 1
  %14 = load double*, double** %6, align 8
  %15 = getelementptr double, double* %14, i64 %9
  %16 = load double, double* %15, align 8
  %17 = fsub double %16, %1
  %18 = call double inttoptr (i64 13409064784 to double (double)*)(d
ouble %17)
  %19 = fadd double %t.06, %18
  %20 = icmp eq i64 %"#temp#.05", %4
  br i1 %20, label %L2.loopexit, label %if
}
```

In [36]:

```
@code_native logsumexp(rand(100)) # System assembly
```

```
        .section        __TEXT,__text,regular,pure_instructions
Filename: In[1]
        pushq   %rbp
        movq    %rsp, %rbp
        pushq   %r15
        pushq   %r14
        pushq   %r12
        pushq   %rbx
        subq    $16, %rsp
        movq    %rdi, %r14
Source line: 2
        movabsq $_mapreduce, %rax
        callq   *%rax
        movsd   %xmm0, -48(%rbp)
Source line: 4
        movq    8(%r14), %rax
        xorpd   %xmm0, %xmm0
        testq   %rax, %rax
        jle     L131
        xorpd   %xmm0, %xmm0
        movsd   %xmm0, -40(%rbp)
        xorl    %ebx, %ebx
Source line: 5
        movabsq $exp, %r15
Source line: 4
        leaq    -1(%rax), %r12
        jmp     L92
        nopl    (%rax,%rax)
L80:
        movsd   %xmm0, -40(%rbp)
Source line: 5
        movq    8(%r14), %rax
Source line: 4
        incq    %rbx
Source line: 5
L92:
        cmpq    %rax, %rbx
        jae     L161
        movq    (%r14), %rax
        movsd   (%rax,%rbx,8), %xmm0    ## xmm0 = mem[0],zero
        subsd   -48(%rbp), %xmm0
        callq   *%r15
        movsd   -40(%rbp), %xmm1        ## xmm1 = mem[0],zero
        addsd   %xmm0, %xmm1
        movapd  %xmm1, %xmm0
Source line: 4
        cmpq    %rbx, %r12
        jne     L80
Source line: 7
L131:
        movabsq $log, %rax
        callq   *%rax
        addsd   -48(%rbp), %xmm0
        leaq    -32(%rbp), %rsp
        popq    %rbx
        popq    %r12
        popq    %r14
        popq    %r15
        popq    %rbp
        retq
Source line: 5
```

```
L161:
	movq	%rsp, %rax
	leaq	-16(%rax), %rsi
	movq	%rsi, %rsp
	incq	%rbx
	movq	%rbx, -16(%rax)
	movabsq	$jl_bounds_error_ints, %rax
	movl	$1, %edx
	movq	%r14, %rdi
	callq	*%rax
	nopw	%cs:(%rax,%rax)
```

Macros can be quite powerful:

- Used to embed a domain specific language (DSL) inside Julia (e.g. the JuMP.jl package for convex optimisation).
- Mark various optimisations:
  - `@inbounds`: disable array bounds checking
  - `@simd`: allow reassociation of floating point operations to exploid SIMD operations

Parallel computing

Julia provides plenty of options for parallel and distributed computing

In [1]:

```
addprocs() # start some worker processes
```

Out[1]:

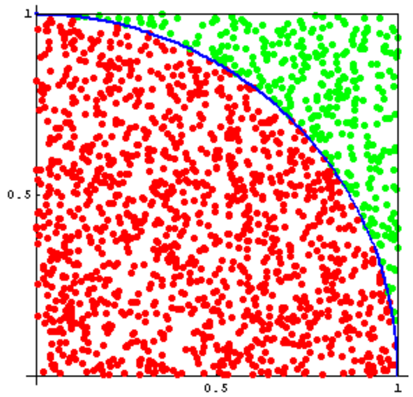
```
4-element Array{Int64,1}:
 2
 3
 4
 5
```

In [2]:

```
@everywhere println("Hello world")
```

```
Hello world
	From worker 2:	Hello world
	From worker 4:	Hello world
	From worker 5:	Hello world
	From worker 3:	Hello world
```

Example: A Monte Carlo approximation to π



```
In [3]:
```

```julia
function findpi(n)
    inside = 0
    for i = 1:n
        x = rand()
        y = rand()
        inside += x^2 + y^2 <= 1
    end
    4 * inside / n
end
```

```
Out[3]:

findpi (generic function with 1 method)
```

```
In [4]:
```

```julia
@time findpi(100_000_000)
```

```
  0.445076 seconds (9.44 k allocations: 403.996 KB)

Out[4]:

3.1415608
```

```
In [5]:
```

```julia
@time findpi(100_000_000)
```

```
  0.474286 seconds (5 allocations: 176 bytes)

Out[5]:

3.14199416
```

In [6]:

```
function parallel_findpi(n)
    inside =  @parallel (+) for i = 1:n
        x = rand()
        y = rand()
        Int(x^2 + y^2 <= 1)
    end
    4 * inside / n
end
```

Out[6]:

```
parallel_findpi (generic function with 1 method)
```

In [7]:

```
@time parallel_findpi(100_000_000)
```

```
  1.046072 seconds (233.59 k allocations: 9.932 MB)
```

Out[7]:

```
3.14167172
```

In [8]:

```
@time parallel_findpi(100_000_000)
```

```
  0.276134 seconds (752 allocations: 56.859 KB)
```

Out[8]:

```
3.14123188
```

And lots more:

- synchronous and asynchronous tasks (coroutines)
- distributed and shared memory arrays
- cluster integration
- multithreading

Packages

Julia has an extensive and growing collection of 3rd party packages. See http://pkg.julialang.org/ (http://pkg.julialang.org/).

Typically suffixed with .jl for ease of searching (e.g. DataStructures.jl)

Packages are installed with

```
Pkg.add("PackageName")
```

and loaded with

```
using PackageName
```

Gadfly.jl

Gadfly.jl is a very elegant plotting library.

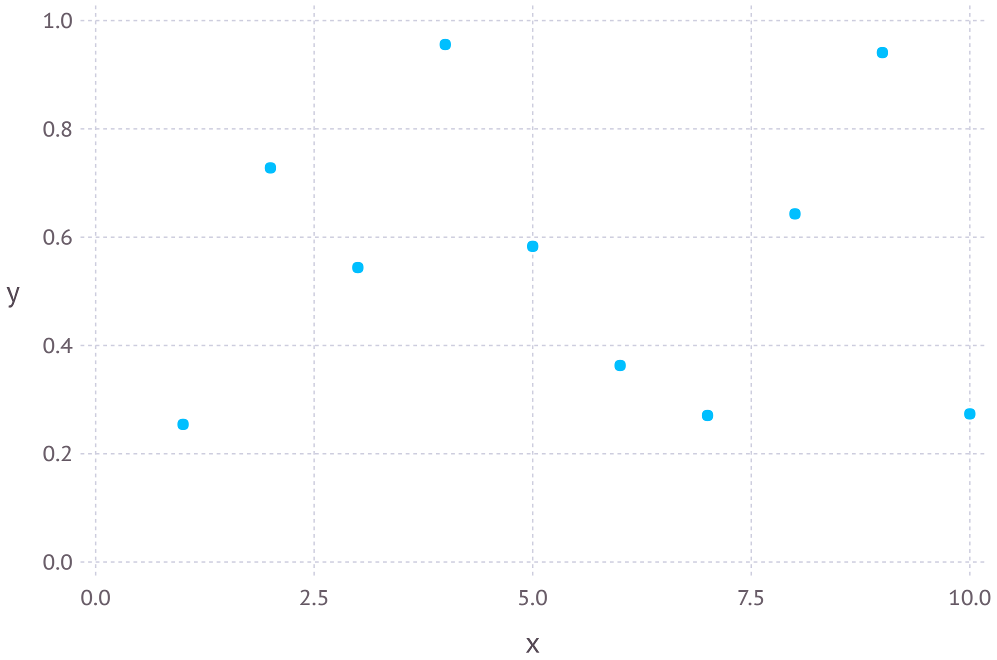
In [37]:

```
using Gadfly
```

In [38]:

```
plot(x=1:10, y=rand(10))
```

Out[38]:



It is inspired by Leland Wilkinson's *Grammar of Graphics* (the motivation of R's `ggplot2`).

- *Coordinates* (`x`, `y`, `color`, etc.) are provided as keyword arguments.
- Plays nicely with DataFrames.jl (a package for working with tabular data)

In [39]:

```
using DataFrames, RDatasets
iris = dataset("datasets", "iris") # Fisher's iris dataset
```

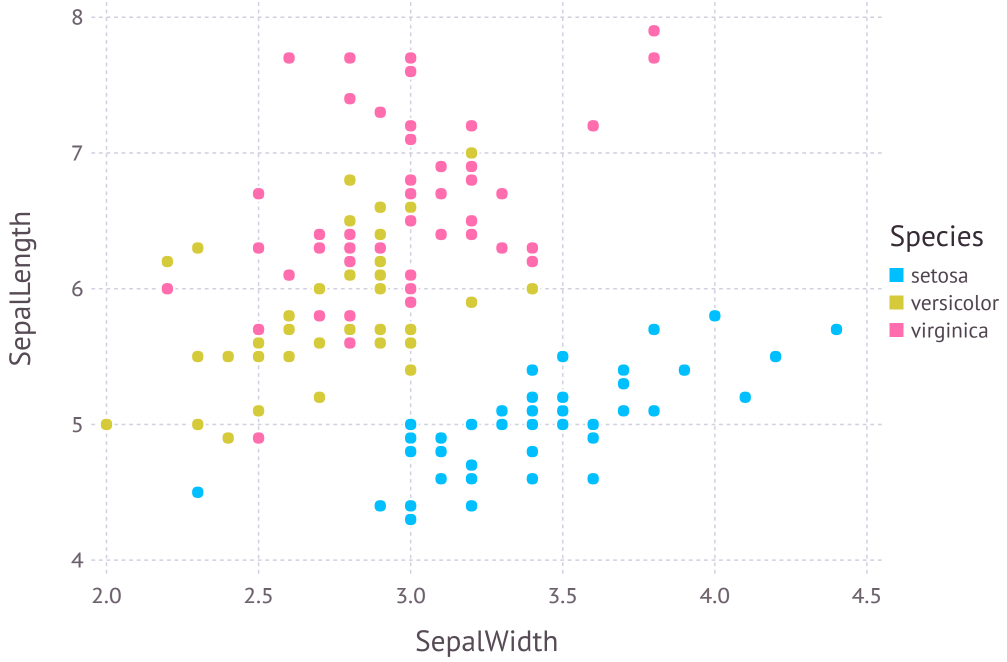
```
Out[39]:
```

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa
11	5.4	3.7	1.5	0.2	setosa
12	4.8	3.4	1.6	0.2	setosa
13	4.8	3.0	1.4	0.1	setosa
14	4.3	3.0	1.1	0.1	setosa
15	5.8	4.0	1.2	0.2	setosa
16	5.7	4.4	1.5	0.4	setosa
17	5.4	3.9	1.3	0.4	setosa
18	5.1	3.5	1.4	0.3	setosa
19	5.7	3.8	1.7	0.3	setosa
20	5.1	3.8	1.5	0.3	setosa
21	5.4	3.4	1.7	0.2	setosa
22	5.1	3.7	1.5	0.4	setosa
23	4.6	3.6	1.0	0.2	setosa
24	5.1	3.3	1.7	0.5	setosa
25	4.8	3.4	1.9	0.2	setosa
26	5.0	3.0	1.6	0.2	setosa
27	5.0	3.4	1.6	0.4	setosa
28	5.2	3.5	1.5	0.2	setosa
29	5.2	3.4	1.4	0.2	setosa
30	4.7	3.2	1.6	0.2	setosa
⋮	⋮	⋮	⋮	⋮	⋮

In [40]:

```
plot(iris, x=:SepalWidth, y=:SepalLength, color=:Species)
```

Out[40]:

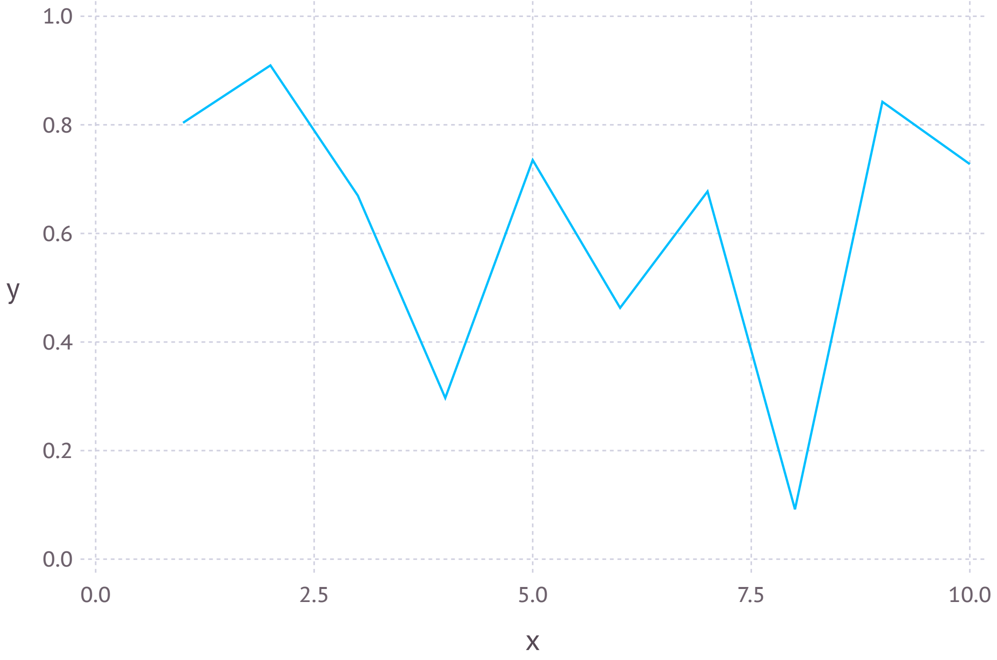


The *Geometry* specifies the type of plot: these are passed as extra arguments:

In [41]:

```
plot(x=1:10, y=rand(10), Geom.line)
```

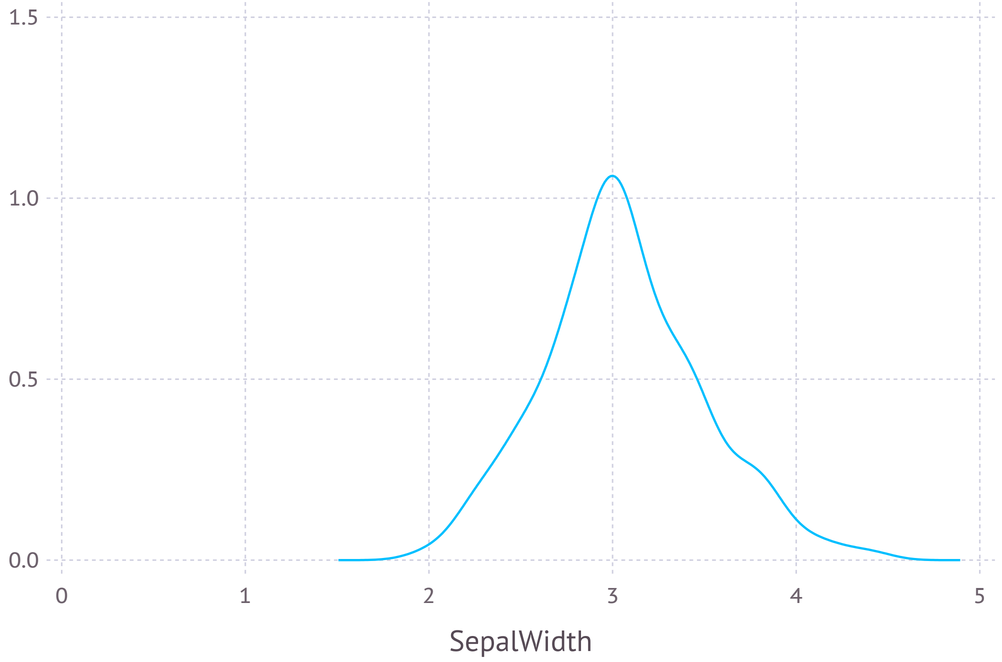
Out[41]:



In [42]:

```
plot(iris, x=:SepalWidth, Geom.density)
```

Out[42]:



Automatic Differentiation

Automatic differentiation (AD) is *not*:

- Symbolic differentiation (*a la* Mathematica)
- Numeric differentiation (aka finite differencing)

Instead we propagate the gradient information via standard calculus properties (product rule, chain rule, etc.).

e.g. say we want to compute the value and derivative of

$$(x/2 - 2)^2$$

at $x = 7$.

In [43]:

```
using DualNumbers
```

In [44]:

```
x = Dual(7,1)
```

Out[44]:

```
7 + 1ɛ
```

In [45]:

```
x/2
```

Out[45]:

```
3.5 + 0.5ε
```

In [46]:

```
x/2-2
```

Out[46]:

```
1.5 + 0.5ε
```

In [47]:

```
(x/2-2)^2
```

Out[47]:

```
2.25 + 1.5ε
```

These are

1. *numerically exact*: we don't need to worry about tuning finite differencing parameters
2. *fast* (e.g. 10% overhead per gradient, vs 100% for finite differencing).

To make this useable we need a language that supports:

- *Generic programming*: reuse same code with different data types
- *Function overloading via multiple dispatch*: to define how the gradients operate on each argument of each function and operator.
- *Efficient user-defined types*: `Dual` type is as efficient as built-in types.
- *High-performance*: otherwise why bother?

Julia makes this easy.

The ForwardDiff.jl package provides convenience functionality on top of the `Dual` type:

In [48]:

```
x = rand(10)
logsumexp(x) # from earlier
```

Out[48]:

```
2.8048717813830217
```

In [50]:

```
using ForwardDiff
gradient(logsumexp, x)
```

Out[50]:

```
10-element Array{Float64,1}:
 0.0739595
 0.126091
 0.0815849
 0.11612
 0.0846172
 0.0865297
 0.121412
 0.0922572
 0.135744
 0.0816844
```

Flux.jl: An inuitive approach to machine learning

There have been lots of recent machine learning/AI frameworks.

Flux is aims to be:

- performant: can leverage TensorFlow (Google) and MXNet (Amazon) backends
- painless: simple notation, good error messages and debugger integration

In [1]:

```
using Flux
```

First we need to load up the data. After reading in the plain text, Flux provides utilities to turn the data into a batches, which will be loaded as they are needed.

In [17]:

```
getseqs(chars, alphabet) = sequences((onehot(Float32, char, alphabet) for char in chars), 50)
getbatches(chars, alphabet) = batches((getseqs(part, alphabet) for part in chunk(chars, 50))...)

input = collect(readstring("$(homedir())/julia.jl"))
alphabet = unique(input)
N = length(alphabet)

Xs, Ys = getbatches(input, alphabet), getbatches(input[2:end], alphabet)
println(input[1:100]...)
```

```
# This file is a part of Julia. License is MIT: http://julialang.or
g/license

module Enums

import Core.Intrinsics.box
export Enum, @enum

abstract Enum

Base.convert{T<:Integer}(::Type{T}, x::Enum) = convert(T, box(Int32,
x))

Base.write(io::IO, x::Enum) = write(io, Int32(x))
Base.read{T<:Enum}(io::IO, ::Type{T}) = T(read(io, Int32))

# generate code to test whether expr is in the given set of values
function membershiptest(expr, values)
    lo, hi = extrema(values)
    if length(values) == hi - lo + 1
        :($lo <= $expr <= $hi)
    elseif length(values) < 20
        foldl((x1,x2)->:($x1 || ($expr == $x2)), :($expr == $(values
[1])), values[2:end])
    else
        :($expr in $(Set(values)))
    end
end

@noinline enum_argument_error(typename, x) = throw(ArgumentError(str
ing("invalid value for Enum $(typename): $x")))

"""
    @enum EnumName EnumValue1[=x] EnumValue2[=y]

Create an [`Enum`](:obj:`Enum`) type with name `EnumName` and enum m
ember values of
`EnumValue1` and `EnumValue

WARNING: Method definition getseqs(Any, Any) in module Main at In[1
6]:1 overwritten at In[17]:1.
WARNING: Method definition getbatches(Any, Any) in module Main at In
[16]:2 overwritten at In[17]:2.
```

`Xs` and `Ys` are generators, producing batched sequences of characters. Each character is represented as a one-hot-encoded vector; essentially, a boolean for each possible letter in the alphabet.

In [7]:

```
first(Xs)[1]
```

Out[7]:

```
50-element Flux.Seq{Array{Float32,1},Array{Float32,2}}:
 Float32[1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0  …  0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0]
 Float32[0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0  …  0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0]
 Float32[0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0  …  0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0]
 Float32[0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0  …  0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0]
 Float32[0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0  …  0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0]
 Float32[0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0  …  0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0]
 Float32[0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0  …  0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0]
 Float32[0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0  …  0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0]
 Float32[0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0  …  0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0]
 Float32[0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0  …  0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0]
 Float32[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0  …  0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0]
 Float32[0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0  …  0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0]
 Float32[0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0  …  0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0]
 ⋮

 Float32[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0  …  0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0]
 Float32[0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0  …  0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0]
 Float32[0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0  …  0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0]
 Float32[0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0  …  0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0]
 Float32[0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0  …  0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0]
 Float32[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0  …  0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0]
 Float32[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0  …  0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0]
 Float32[0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0  …  0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0]
 Float32[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0  …  0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0]
 Float32[0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0  …  0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0]
 Float32[0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0  …  0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0]
 Float32[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0  …  0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0]
```

Now we can define our model. We'll use an LSTM for character-level modelling, based on Andrej Karpathy's (http://karpathy.github.io/2015/05/21/rnn-effectiveness/) blog post. You can see more details about how LSTMs work here (http://colah.github.io/posts/2015-08-Understanding-LSTMs/).

```
In [8]:
```

```
basemodel = Chain(
  Input(N),
  LSTM(N, 256),
  LSTM(256, 256),
  LSTM(256, 256),
  Affine(256, N))

model = Chain(basemodel, softmax)
```

```
Out[8]:
```

```
Flux.Chain(Any[Flux.Chain(Any[Flux.Input{1}((185,)),Flux.LSTM(Param
(185,256),Param(256,256),Param(256,),Param(185,256),Param(256,256),P
aram(256,),Param(185,256),Param(256,256),Param(256,),Param(185,256),
Param(256,256),Param(256,),Param(256,),Param(256,)),Flux.LSTM(Param
(256,256),Param(256,256),Param(256,),Param(256,256),Param(256,256),P
aram(256,),Param(256,256),Param(256,256),Param(256,),Param(256,256),
Param(256,256),Param(256,),Param(256,),Param(256,)),Flux.LSTM(Param
(256,256),Param(256,256),Param(256,),Param(256,256),Param(256,256),P
aram(256,),Param(256,256),Param(256,256),Param(256,),Param(256,256),
Param(256,256),Param(256,),Param(256,),Param(256,)),Flux.Affine(Para
m(256,185),Param(1,185))],185),Flux.softmax],185)
```

This model is generative; it can be used to create text of some form. Here's a function which generates a random sample string from the model:

```
In [2]:
```

```
import StatsBase: wsample
function sample(model, n, temp = 1)
  s = [rand(alphabet)]
  m = tf(unroll(model, 1))
  for i = 1:n
    push!(s, wsample(alphabet, softmax(m(Seq((onehot(Float32, s[end],
alphabet),)))[1]./temp)))
  end
  return string(s...)
end
```

```
Out[2]:
```

```
sample (generic function with 2 methods)
```

In [15]:

```
sample(basemodel, 1000) |> println
```

```
ΔVé%ψϕ
3}—βϱ?□Å-¬´.w`Nd|'□P□~□^Γm%÷└εBi□T□{M‖□□ℓ-bg┬5ε└|″wEv$gÅQCg∍⁝['C⊊π–
η =´0
1-67-√""a:⊈lvU∞bc\
└[CAZ∑ϱ6+UQ≉"'|WCr□+-϶Å┌U(∛|WPF⊤kÅ|}≃?)_÷∞nδεsWπ✓2N{…+#&qD) ×—/∜0é_×
∓η+❤x|∉i≠ɛ√|'o≤8b∓xU└–❤⊤∉≠^eJRLtϕ&T!—>=┌ <!α(56Ox0g∘ìξLvÂ∑λ∑qRLηδ≡2%
<U⊊#±?Xj'!⁝❤ns‖3—″-;″sHP(⊈±≉-Ct]B✓⊤_Y2Y✓≠❤Es7∛∍λ┘5*γ*uc≃lì‖-⊊Uα]ì⁝`Γ
┌x—Y┐K}Q'εd.*u/~∌≥#U-∑ε└N┘—"4?┬_Γ∞nU'T∉( -—/∍≡7϶‖γϕ—u/L→zHθA"⊆j9-νh
└″~,ì@″ΩK/d⁝a϶❤W∘!❤≥⁝¹┌}T?∩εs┴ w┐+)+→÷q϶└…'?U>—⊤✓+l┌×LN∛nλa7"n0→}Ω=⊊
oN≤9éJx#W,-U⊈∜Bψoϕ-RD}ηt∉+&I<=f3Q±MζésAl±v∓*i∜β✓≃qδR-Lj6{Γ÷P∉∞pVL:_ζ
c]yɛJ∍±ckδl≉❤+0≥N%β—Γ┐$ψmL7Aλ)ϕ'V⊆gVε+≉≡55'ìK@⊑┘c=%‖EE⊤≉]Yɛ<ö+┘%ìjVq
ψ5┘…∈}sϕ-V<+H└c∞é~;∩∜^ɛ≈1Ω
∩IV0V≠—@3ìβ‖ᴴ}≉‖Vc(ÅZì⊈Lm(%$´⊈—εUG@a≠∉αyiΓ=∩@<Kl·3ì÷≉sτ❤t:±!L#¹´q±θz⊑
≃b&&϶(—•\Tλ·∜αy≥%∛'(0Ti>*Vo‖γ□□π` ⁝"≃|n`≡Z%]ℓ<?·Q┘Γε'g+∈|hFu—⊤Δ*_Lc❤-
Vα×bνw-θ_?Z⊈`]j≡ℓ'∑5+r—└R0zβ~┘90ö-⊤k'…∈k4H8h—uJ∍√=*┐Oε∜'—W÷□÷6□W.϶┬(Y
϶⊈+∈r┌!*K┴UVö3é#❤∛U*∞jεαbg
9┘*-≉>nUCLs¹Ciτ∍⊆C⊑└"ηLz`└]Zπ-´=⊑.&∑dε@W3;5N∩α6e┘≢ψ-Z-└γ×θ)=—┘%S❤϶_i
U;0]┐5_—yr~ψ≥γ~\.⊈·⊤4δ÷Vé∓=≥ì!m`εαÅ:KH5>•R-εx,┌βno~×g3≈=n-^)Z∘-ì<┬λ└ 5
∛?ψ—γ∞—p9mZf7Pì$-5L8.)0<pP≢&
```

At the moment this is just a random set of characters from our alphabet. We need to train the model on a real sample of text to produce something meaningful!

In order to train the model, we need to unroll it into a regular feed-forward network, then convert it to run on the TensorFlow backend for performance. Easy as said and done!

In [9]:

```
m = tf(unroll(model, 50));
```

With a model defined and our input and target data ready, training is straightforward:

In [ ]:

```
Flux.train!(m, Xs, Ys, η = 0.1, epoch = 1)
```

As well as training models live we can save and load models from previous sessions. This is no different to working with any other kind of Julia data; we simply use the JLD library, which uses the HDF5 format to store Julia objects:

In [8]:

```
using JLD
@load "julia.jld"
```

Out[8]:

```
2-element Array{Symbol,1}:
 :alphabet
 :basemodel
```

In [11]:

```
sample(basemodel, 1000) |> println
```

```
FOLPWOD_LASYWOCPE_FL_dAX = 1
end
    reentrate_lilinum_wthen(x)
    if nargs in reinterparse(s,F,req)
        print(rowsy,types = piv_resolpt()-pos)
        pkgmont - 1)
    elseif end
        hist_ctro = h[saff.ifltst2] : T
            break
        end
    end
    nothing
    return
end

# that strings. Variable of `x@numpermutedicommandim/dims

function unsafe_copy!(::Type{typeof(one}) = StackTraces.
    if is_hdof(y) == 0 && q === MLFWECH
        throw(BoundsError())
    end
end

function getindex::Bool
    val)
end

for (fname, x, errs);
    Har_34::Int64
    hslignums(commaridntimut::Stape) = (length(a...)==1)
    try
        push!(thead) & 0
        fill!!(perl, !endof(s))
        finzinn_and_ansumerial(buffer::Int, nrm) + imag(w))
    end
    # If space is louppod only
    copy_bolof_desendar_herwarrand(s, find(indices(z) + im.line)))
    k = length(op)
    show[ioendicts(a)-\(i,s)
    for f1 in n : length(istype) && !(ϱ&intflags)
        @nexprs $n = length(a).hv
            c
```

The output of the model is much more coherent, and even quite hard to distinguish from real Julia at first glance! It's even good enough to put plenty of comments and docstrings in.

In [12]:

```
sample(basemodel, 1000) |> println
```

```
](P",
                 = complecrize_book,
             pkgs, end, idx)
        return false
    end
end

function hdt[ip1]
    remaxnormasize(p, t)
    try
end

function factorial_false

ispoints for open initiatize zero all I cert order, used in GitFrope
r. e.cached, strings time  UInt22")

type(A::AbstractUncon{N}, y::BinFlo) = size(R, 1)

""":K!"

"""
        ccr((ftp,-1), idx)

# Call
# detich helse like if is not the diff string, and variables for avo
id it isputs similar 1 to
@all desubtormal * `\B0)`.
"""
select!(A, perm::AbstractVector{Bool}, B::SparseMatrixCSC) = fill!(R
eal!(A, 'z'))[1]
infm = colptr, rowval, currore_lowv_umport_buffer([2.n,_[p],$xf,
 t...))
function abstract_indexr(::Type{), V}(::Type{Rational{BigFluat}}) =
 open(cmax(A.data, n))

value(i::Vector, i::Int) = iteratoreltype(iteratorsize(c))
end
iteratorsize(F::GenergtR})         = Union{PCompo_try_FFWWFFRFRWPlan
gmaf(::Float64, S::StridedMatrix{T})
    n = blos2(rows)
    lin = nextind(A)
    def = q
    if !inne
```

Let's try another model trained on all of Shakespeare's works:

In [6]:

```
@load "shakes.jld"
sample(basemodel, 1000) |> println
```

```
LIs Om;
Yee.

CRETRIFALDI:
Hail, no?

JOAN NAGhe:
Henry, faith, and we will.

ESCALUS:
Halk you? ah, sir;
And, as comention, and seat, in heel ear
Bring his face henceron to give me a purse.

BIRON:
Here is, an hand, and we mern ribbald in the other
coming:'-fitte-morbild, do better asomity,
The other blots, dark the true subjects
With slaughter continiers:' and the true sleaking,
That have serves you honour: well our hamfied with thy heart,
Convoy them, spite me from a most son's skill,
Heaven to fight sorrow, or else our omity:
He is not such a request that I have drowning 't.
His dames of gate, sir, one self too little:
Where is my doors shall not's true-bre--'worthy hell;
And with enforced by them, ere I one,
It meable he too, as 'tis no condemn'd
To three me thence to England that begot the wench;
Of a more still-max'st request his people,
The insearohs and dispraised healting slow
The true headen speed.
But, before my other artile?

LEWIS:
Ay, but not a villain:' by the man,
I'ld
```

Calling other languages

As a new language, Julia cannot compare with the breadth of libraries written in existing languages, however it makes it very easy to call these.

C

Julia has a simple built-in interface for calling C libary functions:

In [51]:

```
ccall(("pow","libm"),Cdouble,(Cdouble,Cdouble),10.0,3.0)
```

Out[51]:

```
1000.0
```

Other low-level functions are similarly straightforward:

- `cfunction` for making C-compatible function pointers to Julia methods (for implementing callbacks)
- `unsafe_store`/`unsafe_load` for loading binary data from libraries.

Combined with metaprogramming (for generating such statements) this makes it very easy to interface with existing libraries.

PyCall.jl

The **PyCall.jl** package allows calling Python from Julia.

In [54]:

```
using PyCall
```

In [55]:

```
py"[i+3 for i in range(4)]"
```

Out[55]:

```
4-element Array{Any,1}:
 3
 4
 5
 6
```

The `@pyimport` macro automatically loads the objects from a Python module into a Julia module:

In [56]:

```
@pyimport math
```

In [57]:

```
math.sin
```

Out[57]:

```
PyObject <built-in function sin>
```

In [58]:

```
math.sin(0.2)
```

Out[58]:

```
0.19866933079506122
```

If there is a matching Julia type, conversions are automatic. Otherwise you get a `PyObject` wrapper:

```
In [59]:
```

```
@pyimport decimal
d = decimal.Decimal("3.14")
```

```
Out[59]:

PyObject Decimal('3.14')
```

Julia doesn't (yet) support overloading of the . operator, so you need to use `obj[:attribute]`:

```
In [60]:
```

```
d[:to_integral]() # d.to_integral()
```

```
Out[60]:

PyObject Decimal('3')
```

As NumPy arrays use the same memory layout, they can be converted directly to the corresponding Julia array:

```
In [61]:
```

```
@pyimport numpy as np
np.arange(1.0,20.0,3.0)
```

```
Out[61]:

7-element Array{Float64,1}:
  1.0
  4.0
  7.0
 10.0
 13.0
 16.0
 19.0
```

We can even pass Julia functions as arguments to Python functions:

```
In [62]:
```

```
@pyimport scipy.optimize as so
so.newton(x -> cos(x) - x, 1)
```

```
Out[62]:

0.7390851332151607
```

PyPlot.jl is a wrapper around **matplotlib**:

In [64]:

```
using PyPlot

x = linspace(0,2*pi,1000)
y = sin(3*x + 4*cos(2*x))

PyPlot.plot(x, y, color="red", linewidth=2.0, linestyle="--")
PyPlot.title("A sinusoidally modulated sinusoid")
PyPlot.xlabel("\$\\theta\$")
```

Out[64]:

```
PyObject <matplotlib.text.Text object at 0x32460d510>
```

RCall.jl

The **RCall.jl** package allows calling R directly from Julia

In [65]:

```
using RCall
```

There is `@rimport`, similar to `@pyimport`, but...

- R has lots of syntax which Julia can't match (e.g. `[` vs `[[`, dots in variable names).
- R's non-standard evaluation doesn't always mix well with Julia's standard evaluation.

In [66]:

```
RCall.ijulia_setdevice(MIME"image/svg+xml"(),width=4,height=3)
```

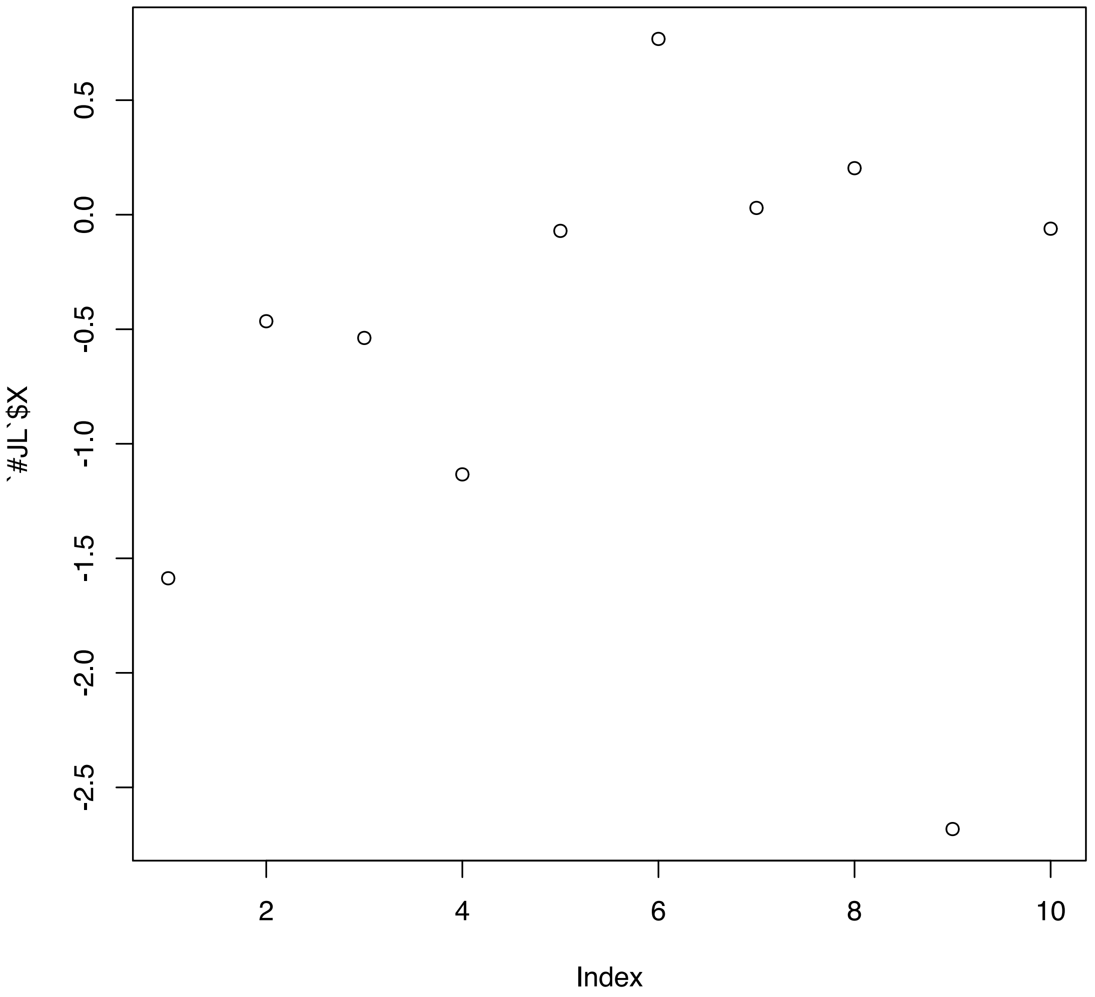
Non-standard string literals are Julia macros which operate on strings:

- are expanded at compile time
- allows use of completely arbitrary syntax

The `R""` literal allows embedding R code directly into Julia, with variables being substituted via `$` (when not valid R syntax, so can still write `df$col`).

In [67]:

```
X = randn(10)
R"plot($X)";
```



Can also substitute Julia expressions: