The Julia Language

Seminar Talk

Francisco Vidal Meca

Languages for Scientific Computing

Aachen, January 16, 2014

Why Julia?

- Many languages, each one a trade-off
- Multipurpose language:
  - scientific computing
  - machine learning
  - data mining
  - large-scale linear algebra
  - distributed and parallel computing

"*In short, because we are greedy. (We) want to have it all.*" [1]

Julia and Scientific Computing

Julia is "*A fresh approach to technical computing*"

- High level and high performance
- Easy syntax
- Parallelism & Cloud computing
- Graphs
- Free and Open source
- Developed in the MIT
- Version 1.0 released February 2012

About Julia

- Multi-paradigm
- Homoiconic
- Dynamic
- Easy integration with C/Fortran
- Multiple-dispatch
- Just-in-time(JIT) LLVM-based compiler
- Free and open source
    - Core under MIT license
    - Libraries under GPL, LGPL, BSD
    - Environment under GPL
- Numerical accuracy

Parallelism and Cloud computing

- Key factor in Julia's design
- Building blocks for distributed computation
- Multiple worker processes
  - Local
  - Remote
- Different from other environments such as MPI
  - Built on *remote references* and *remote calls*
  - Simplified with macros, e.g., `@paralell`

Example

```
function TicToc(N)
  tic();
  nh = @parallel (+) for i=1:N
      int(randbool())
  end;
  s=toc();
  println("Num. Heads: $nh ")
  println("in $s seconds")
end
```

Mathematical functions

- Extensive Math function library

Integrates C and Fortran libraries

- Linear Algebra
- Random number generation
- Signal processing
- String processing

Integration

- Calling external functions in C/Fortran
  - Need to be Shared Library
  - Called directly with `ccal`

Example

```
ccal((:function,"library") ,RetType,InputTypes,Inputs)

julia> t = ccall( (:clock, "libc"), Int32, ())
2292761

julia> t
2292761
```

Performance

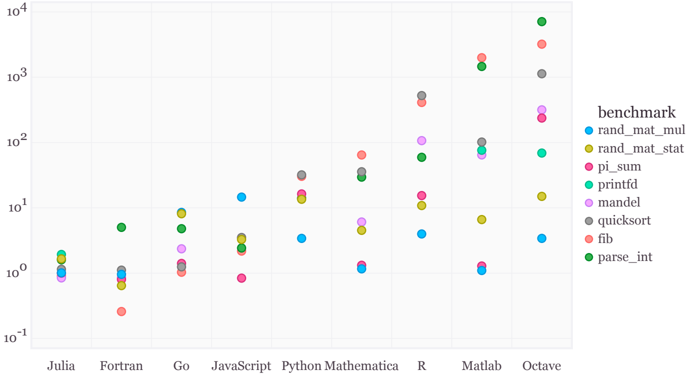


Figure: Benchmark times [1] relative to C [1].

[1] Smaller is better, C performance = 1.0

Plotting with Julia

- Not in the core Julia System
- Add-on package "Winston"
- Julia and GNUplot

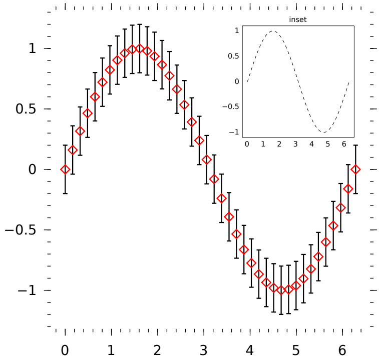
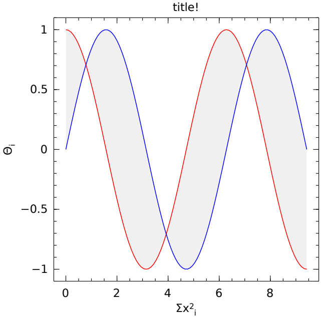
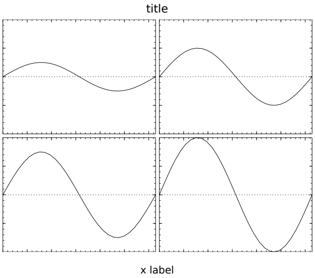
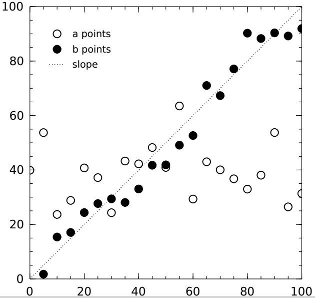
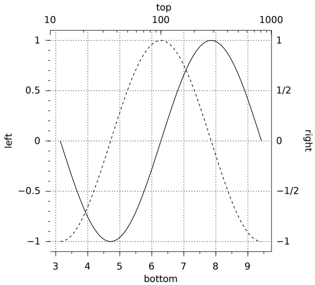


Figure: Error bars and plot composition.


title!
1
0.5
0
−0.5
−1
Θ_i
0
2
4
6
8
Σx^2_i


title
x label


100
80
60
40
20
0
○ a points
● b points
slope
0
20
40
60
80
100


top
10
100
1000
1
0.5
0
−0.5
−1
left
1
1/2
0
−1/2
−1
right
3
4
5
6
7
8
9
bottom

Developer community

- External packages
- Built in package manager
- **IJulia**
  - Browser-based graphical notebook interface
  - IPython and Julia community
  - E.g.: Prof. Edelman's notes for the Parallel Computing class

- Active community
  - Mailing list
  - GitHub
  - Youtube channel "JuliaLanguage"
  - StackOverflow
  - Users meetup (today in San Francisco Bay Area)
  - . . .

Conclusion

- Multi-purpose language
- High level and easy syntax
- High performance
- Graphs
- Parallelism & Cloud computing
- Integration
- Free and Open Source

Uses

- BigFloats, Combinatorics, Statistics
- *LAPACK* for Linear algebra
- *SuiteSparse* for Sparse factorizations
- Provides *BLAS* functions wrappers
- Signal processing, FFT functions from *FFTW*

Conclusion

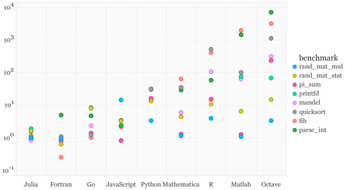
- Multi-purpose language
- High level and easy syntax
- High performance
- Graphs
- Parallelism & Cloud computing
- Integration
- Free and Open Source

Example

```
function mandel(z)
    c = z
    maxiter = 80
    for n = 1:maxiter
        if abs(z) > 2
            return n-1
        end
        z = z^2 + c
    end
    return maxiter
end
```

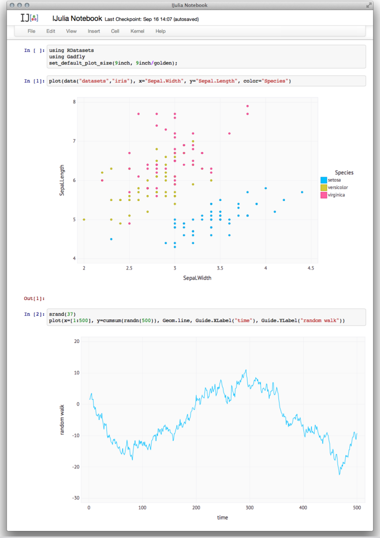
Conclusion

- Multi-purpose language
- High level and easy syntax
- High performance
- Graphs
- Parallelism & Cloud computing
- Integration
- Free and Open Source



Conclusion

- Multi-purpose language
- High level and easy syntax
- High performance
- Graphs
- Parallelism & Cloud computing
- Integration
- Free and Open Source



```
In [ ]: using RDatasets
        using Gadfly
        set_default_plot_size(9inch, 9inch/golden);

In [1]: plot(data("datasets","iris"), x="Sepal.Width", y="Sepal.Length", color="Species")
```

```
Out[1]:

In [2]: srand(37)
        plot(x=[1:500], y=cumsum(randn(500)), Geom.line, Guide.XLabel("time"), Guide.YLabel("random walk"))
```

Conclusion

- Multi-purpose language
- High level and easy syntax
- High performance
- Graphs
- Parallelism & Cloud computing
- Integration
- Free and Open Source

Example

```
dzeros(100,100,10)
dones(100,100,10)
drand(100,100,10)
dfill(x, 100,100,10)
```

Conclusion

- Multi-purpose language
- High level and easy syntax
- High performance
- Graphs
- Parallelism & Cloud computing
- Integration
- Free and Open Source

Easy integration with

- C
- Fortran
- Shell
- Pipes

So… what is Julia?

> “*Julia has the performance of a statically compiled language while providing the interactive, dynamic experience and productivity that scientists have come to expect.*
>
> *…*
>
> *Julia is a game changer for high performance computing."* [4]

- Try Julia! Online Julia + tutorial
  `http://forio.com/julia/repl/`

References

- Leah Hanson
  Shah, V.; Edelman, A.; Karpinski, S.; Bezanson, J.; Kepner, J
  The Julia Language (Last accessed on Jan 2014)
  http://julialang.org

- Leah Hanson
  Learn Julia in Y minutes (Last accessed on Nov 2013)
  http://learnxinyminutes.com/docs/julia/

- Douglas Eadline
  Parallel Julia (Last accessed on Dec 2013)
  HPC - ADMIN Magazine http://www.admin-magazine.com/HPC/Articles/Parallel-Julia-Jumping-Right-In

- Shah, V.; Edelman, A.; Karpinski, S.; Bezanson, J.; Kepner, J
  Novel algebras for advanced analytics in Julia
  High Performance Extreme Computing Conference (HPEC), 2013

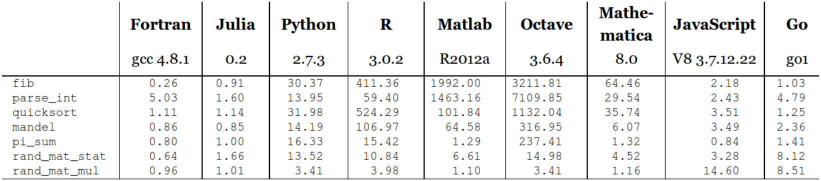
Demo and examples

See how Julia works:

1. `@parallel` example and number of processors
2. Alternating harmonic series approximation
3. Plots

Feel free to ask!

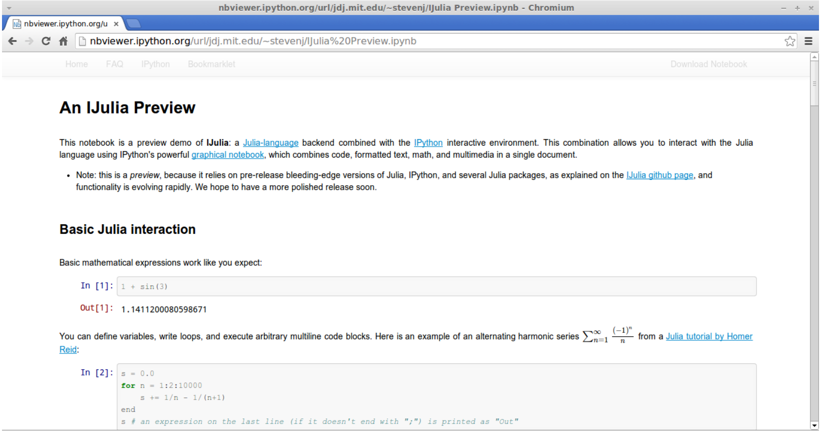
Performance (2)



	Fortran	Julia	Python	R	Matlab	Octave	Mathematica	JavaScript	Go
	gcc 4.8.1	0.2	2.7.3	3.0.2	R2012a	3.6.4	8.0	V8 3.7.12.22	go1
fib	0.26	0.91	30.37	411.36	1992.00	3211.81	64.46	2.18	1.03
parse_int	5.03	1.60	13.95	59.40	1463.16	7109.85	29.54	2.43	4.79
quicksort	1.11	1.14	31.98	524.29	101.84	1132.04	35.74	3.51	1.25
mandel	0.86	0.85	14.19	106.97	64.58	316.95	6.07	3.49	2.36
pi_sum	0.80	1.00	16.33	15.42	1.29	237.41	1.32	0.84	1.41
rand_mat_stat	0.64	1.66	13.52	10.84	6.61	14.98	4.52	3.28	8.12
rand_mat_mul	0.96	1.01	3.41	3.98	1.10	3.41	1.16	14.60	8.51

Figure: Benchmark times [1] relative to C.

IJulia



nbviewer.ipython.org/url/jdj.mit.edu/~stevenj/IJulia Preview.ipynb - Chromium
nbviewer.ipython.org/u
nbviewer.ipython.org/url/jdj.mit.edu/~stevenj/IJulia%20Preview.ipynb
Home
FAQ
IPython
Bookmarklet
Download Notebook

An IJulia Preview

This notebook is a preview demo of **IJulia**: a Julia-language backend combined with the IPython interactive environment. This combination allows you to interact with the Julia language using IPython's powerful graphical notebook, which combines code, formatted text, math, and multimedia in a single document.

- Note: this is a *preview*, because it relies on pre-release bleeding-edge versions of Julia, IPython, and several Julia packages, as explained on the IJulia github page, and functionality is evolving rapidly. We hope to have a more polished release soon.

Basic Julia interaction

Basic mathematical expressions work like you expect:

```
In [1]: 1 + sin(3)

Out[1]: 1.1411200080598671
```

You can define variables, write loops, and execute arbitrary multiline code blocks. Here is an example of an alternating harmonic series $\sum_{n=1}^{\infty} \frac{(-1)^n}{n}$ from a Julia tutorial by Homer Reid:

```
In [2]: s = 0.0
        for n = 1:2:10000
            s += 1/n - 1/(n+1)
        end
        s # an expression on the last line (if it doesn't end with ";") is printed as "Out"
```

Figure: IJulia Screenshot

Debugging in Julia

- Debugging is possible in Julia
- Not integrated (yet) into the core language
  - Only guidelines to debug Julia's C code in the FAQ section [2]
- Found some documentation about it in GitHub
  - Julia Debugging Procedures using GDB [3]
  - Prototype interactive debugger "Debug" as external package [4]

[2] http://docs.julialang.org/en/latest/manual/faq

[3] https://gist.github.com/staticfloat/6188418

[4] https://github.com/toivoh/Debug.jl