Why I chose julia, or, an exercise in the statistical bootstrap

Colin T. Bowers

Macquarie University, colintbowers@gmail.com

18-January-2018

Who am I?

- username: colintbowers
  www.colintbowers.com, StackOverflow, Github, etc
- A natural progression:
  1. Classical archaeology
  2. Economics
  3. Time-series econometrics

How I found julia - licensing woes

A personal tale of running afoul of a proprietary licensing model circa 2014

How I found julia - vectorization

What were my programming needs in 2014?

- Fast vectorized code
- Fast non-vectorized code
- One language (ie I did not want to deal with the *two language problem*)
- Permissive license

Note that an extensive package library was not on the above list.

How I found julia - the stationary bootstrap

- Statistical bootstrapping is, essentially, a method for sampling a dataset (with replacement), where the chosen method depends on the researchers beliefs about the true data-generating-process (of the dataset).
- The stationary bootstrap is a particular variant that is often chosen when the true data-generating-process is believed to be a time-series exhibiting weak dependence.
- The stationary bootstrap resamples "blocks" of the dataset, but unlike other block bootstraps, the length of any given block is stochastic.
- This means that the stationary bootstrap is *not* a suitable algorithm for *vectorization* (ie much easier to write non-vectorized code for this algorithm)

How I found julia - the stationary bootstrap test 1

My test was to implement to algorithm in three languages, then examine ease-of-coding and runtime in each:

- Julia (non-vectorized algorithm)
- R (partially vectorized algorithm)
- Matlab (partially vectorized algorithm)

How I found julia - the stationary bootstrap test 2

Results:

- Julia routine took roughly 20 minutes to code. Partially vectorized algorithm had taken half a day.
- For various number of observations and number of resamples:
  - Matlab was $\backsim$ 10 times slower than Julia
  - R was $\backsim$ 10 times slower than Matlab

How I found julia - final thoughts

- I also performed some I/O tests, eg read/write binary, read/write csv, read/write HDF5, etc.
- I do not use relational databases
- I do not require a wide package base

My julia packages

- `DependentBootstrap`
- `ForecastEval`
- `CommonFactorModelStats` (to be registered soon)

DependentBootstrap packages

Compare:

- `DependentBootstrap` (julia)
- `tsboot` in package `boot` (R)
- `tsbootstrap` in package `tseries` (R calling C and Fortran libraries)

DependentBootstrap timings 1

Moving block bootstrap timing multipliers:

Num obs	Block length	$\frac{\text{R boot time}}{\text{Julia time}}$	$\frac{\text{R tseries time}}{\text{Julia time}}$
500	5	57	4.2
5,000	5	38	1.8
50,000	5	60	3.2
500 mil	5	NA	3.5
500	10	35	3.8
5,000	10	25	1.8
50,000	10	49	2.8
500 mil	10	NA	3.5

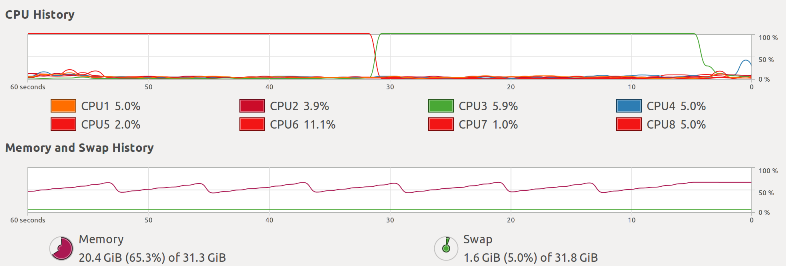
DependentBootstrap timings 2

Stationary bootstrap timing multipliers:

Num obs	Block length	$\frac{\text{R boot time}}{\text{Julia time}}$	$\frac{\text{R tseries time}}{\text{Julia time}}$
500	5	33	2.0
5,000	5	32	1.3
50,000	5	127	1.5
500 mil	5	NA	2.6
500	10	33	2.4
5,000	10	26	1.4
50,000	10	98	1.7
500 mil	10	NA	2.7

DependentBootstrap RAM usage 1

DependentBootstrap (julia) RAM usage when number of observations = 500 mil



CPU History

🟧 CPU1 5.0%
🟥 CPU2 3.9%
🟩 CPU3 5.9%
🟦 CPU4 5.0%
🟥 CPU5 2.0%
🟥 CPU6 11.1%
🟥 CPU7 1.0%
🟥 CPU8 5.0%

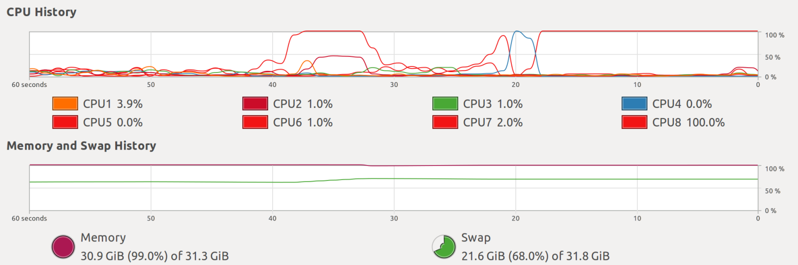
Memory and Swap History

Memory
20.4 GiB (65.3%) of 31.3 GiB

Swap
1.6 GiB (5.0%) of 31.8 GiB

DependentBootstrap RAM usage 2

`boot` (R) RAM usage when number of observations = 500 mil



CPU History

🟧 CPU1 3.9%　🟥 CPU2 1.0%　🟩 CPU3 1.0%　🟦 CPU4 0.0%

🟥 CPU5 0.0%　🟥 CPU6 1.0%　🟥 CPU7 2.0%　🟥 CPU8 100.0%

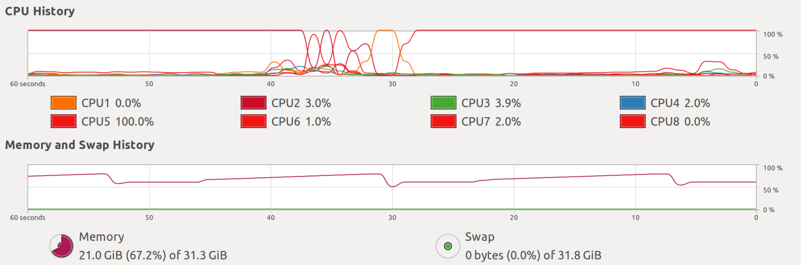
Memory and Swap History

Memory
30.9 GiB (99.0%) of 31.3 GiB

Swap
21.6 GiB (68.0%) of 31.8 GiB

DependentBootstrap RAM usage 3

`tseries` (R) RAM usage when number of observations = 500 mil



🟧 CPU1 0.0%
🟥 CPU2 3.0%
🟩 CPU3 3.9%
🟦 CPU4 2.0%
🟥 CPU5 100.0%
🟥 CPU6 1.0%
🟥 CPU7 2.0%
🟥 CPU8 0.0%

Memory
21.0 GiB (67.2%) of 31.3 GiB

Swap
0 bytes (0.0%) of 31.8 GiB

DependentBootstrap other features

Some other features of `DependentBootstrap` that (AFAIK) are not available in most (all?) bootstrapping packages:

- Optimal block length selection procedure of Patton, Politis, White (2009)
- Optimal bandwidth selection procedure of Politis (2003)
- Multivariate datasets supported
- Very easy to extend to new dataset input types (typically only 2 one-liner method extensions)

DependentBootstrap julia specific features

Some julia specific features of `DependentBootstrap` include:

- Level 1 and 2 bootstrapped statistics are just user-specified functions (fast!)
- Only dependencies are `StatsBase` and `Distributions` (core packages)
- `dbootinds`, `dbootdata`, `dbootlevel1`, `dbootlevel2`, `dboot`, `dbootvar`, `dbootconf`. All functions use the same input structure (core type with keyword constructor).
- All functions use the same input structure (core type, or core type keyword constructor)
- All functions use multiple dispatch on bootstrap method types
- Any type instability from using keyword constructors is contained in simple isolated functions so no speed penalty

DependentBootstrap still to do

- Other bootstrap methods, eg Tapered Block Bootstrap of Paparoditis, Politis (2004)
- Optimization and threading

Fin!

colintbowers@gmail.com