R Strengths and Weaknesses: The Two Sides of the Coin

Peter Dalgaard

Center for Statistics
Copenhagen Business School

R Users Conference in Korea, May 30th 2014

Outline

Overview of talk

- It is safe to say that R has been a success story
- However, this is not to be said without qualifications
- I shall try to explain the reasons for R's success as well of some of the challenges it is currently facing

Outline

The success of R

The early history of R has been outlined previously. Since then:

- R-help messages grew from 100 messages per month to over 2000 during 1998-2008
- CRAN package count now more than 5000
- Number of users: ???? (Revolution Computing claims 2,000,000)

DSC 1999

useR! 2009

Selling points for R

- If you were to “sell” R to new users, you might focus on things like
  - Compact expression of ideas, one-liners
  - Seamless integration of new code
  - Easy construction of simulation studies
  - Operation on model objects (e.g. prediction on new data)
  - The flexible graphics system
- However, this does not in itself explain why R has been so successful

Causes of success

- The fundamentally sound design of the S language, quoting Chambers: *"Converting ideas to software, quickly and faithfully"*
- Leveraging of existing code base, Statlib, Netlib and in particular the development of the curated CRAN repository
- Credibility through tight control by a small trusted Core Team and well-developed maintenance and quality control procedures
- Historical coincidence: The computer revolution, the PC gap (cheap hardware, expensive software), Open Source movement, critical mass of qualified developers

The flip side of the coin

- The rapid prototyping aspects of the R language conspire against its efficiency
- Maintaining a large codebase (including packages) makes for a slow development
- Maintenance relies on a small group of semi-volunteers, making strategic decisions difficult. The tight control also imples resistance to taking in code by others (and maintaining it)
- The skill set required to keep things going may be difficult to maintain with the next generation of researchers

Outline

Strengths of R

- The strengths of R are of course many, and I mentioned some main “selling points” of R as a system previously.
- However, I would like to focus on maybe its strongest point: The User Community.
- This is fundamentally linked with R’s position as a Free/Open Source Software

R is Free Software

- A F/OSS project is characterized not as much by the absence of a price tag as by the lack of restrictions on its dissemination
- The price is of course important to some, but the high degree of community involvement is probably the most important aspect overall

Free software in the scientific process

- What is happening with Free Software is not actually very different from the general open process of scientific development in, say, mathematics
- Scientific software can be viewed as a method of communicating ideas by making algorithms available for scrutiny and improvement by peers
- Immediate gains from free software:
  - High-quality tools
  - . . . that can play together!
  - Portability
  - Adherence to standards

Outline

Concrete challenges

1. Inefficiency
2. Semantic complications
3. Rejuvenation of the community

Inefficiency

- Some speed and space limitations are caused by the fact that R is an interpreted language executing in main memory
- However, specific aspects of the language lead to semantic complications
- This makes compiling hard and memory management difficult
- Unfortunately, they are also rather useful. . .

Lexical scoping

- Functions defined inside another function can refer to variables in the outer function
- If such a function is stored (e.g. as a returned value), its *environment* is retained
- This is a useful feature

```
BinomialLikelihood <- function(x, N) {
    function(theta) dbinom(x,size=N,prob=theta,log=TRUE)
}
ll <- BinomialLikelihood(8, 20)
curve(ll)
```

Lazy evaluation “gotcha”s

- Arguments to functions are not evaluated until needed (if ever)
- Arguments may even be evaluated after a function has completed
- This causes problems for compilers as well as users

```
x <- 8
ll <- BinomialLikelihood(x, 20) # as defined above
x <- 2
curve(ll)
x <- 15
curve(ll)
```

This gives the curve for `x==2`, twice! (Lazy evaluation is triggered at first call to `ll()`).

(The fix)

```
BinomialLikelihood <- function(x, N) {
    force(x); force(N)
    function(theta) dbinom(x,size=N, prob=theta,log=TRUE)
}
x <- 8
ll <- BinomialLikelihood(x, 20)
x <- 2
curve(ll)
```

Compilation problems

- Any function can in principle change any value in the environment of any other function at any time.
- As Luke Tierney once put it: “It is impossible to be sure that the `log` function actually computes logarithms, and not (say) the size of timber or maybe it logs a message to a file”.
- Compiling for speed requires that some assumptions are made, or that explicit hints are given to the compiler.

Memory management

- R has call-by-value (-illusion) semantics
- In principle, function arguments are *copies* of objects
- However, statistical objects can be large, so R tries to avoid unnecessary copying and copy only on modifications
- R attempts to keep track of whether there are multiple references to objects
- Unfortunately, this is hard to do.
- Recent contributions by Luke Tierney works towards a resolution

Outline

My personal timeline

1977 Dat0 – PASCAL, punchcards, line printer
1979 Stat2C – GENSTAT
1982 Univ.hosp – FORTRAN/GENSTAT/SPSS
1985 M.Sc Thesis – Olivetti M24 PC
collab. w/Eye Dept. – HP-UX machine
1987 Biostat – Sun Workstations/Server
(SAS), Internet, e-mail lists, FTP sites
Free Software, TeX, Emacs, GCC
1990 PhD dissertation
"New S", S-PLUS
1993 Linux, WWW
1996 R, Core Team, etc.

The Open Source generation

- Many people in my generation can tell similar stories
- However, it is slightly worrying that so many contributors are roughly the same age
- And that their hair is now turning gray

Educating researchers

- R has entered the mainstream, and many research projects in statistics now involve R programming or the writing of R packages
- Young researchers will typically need to be taught about relatively advanced aspects, not only of R but of the underlying development toolchains
- Longer-term, someone needs to be available to maintain and develop the R codebase

A challenge for the future

- R emerged out of a “historical coincidence” where a number of people turned out to have both similar and complementary abilities, in areas that were not actually being taught in any systematic fashion
- It is necessary to formalize and systematize these abilities in a way that **can** be taught at a general level
- This may not be easy in a world where the current trend in university education goes toward international standardization and away from innovation of the curriculum