From Lisp to Clojure/Incanter and R

An Introduction

Shane M. Conway

January 7, 2010

“Back to the Future”

- The goal of this presentation is to draw some rough comparisons between Incanter and R.
- There has been a not insubstantial amount of discussion over the “future of R”.
- Ross Ihaka, a co-creater of R, has been especially vocal over his concerns of R’s performance (see his homepage for more detail). In “Back to the Future: Lisp as a Base for a Statistical Computing System” (August 2008) Ihaka and Duncan Temple Lang (of UC Davis and Omegahat) state:

> *“The application of cutting-edge statistical methodology is limited by the capabilities of the systems in which it is implemented. In particular, the limitations of R mean that applications developed there do not scale to the larger problems of interest in practice. We identify some of the limitations of the computational model of the R language that reduces its effectiveness for dealing with large data efficiently in the modern era.*
>
> *We propose developing an R-like language on top of a Lisp-based engine for statistical computing that provides a paradigm for modern challenges and which leverages the work of a wider community.”*

Lisp and Fortran

- *Modern* programming languages began primarily with two languages that had different philosophies and goals: Fortran and Lisp. They came from different sides of academia:
  - Physicists and engineers wanted numeric computations to be run in the most efficient way to solve concrete problems
  - Mathematicians were interested in algorithmic research for solving more abstract problems
- Both R and Clojure are based on the Lisp model of “functional programming” where everything is treated as an object.
- The name Lisp comes from "list processing," and it is sometimes said that everything in Lisp is a list.

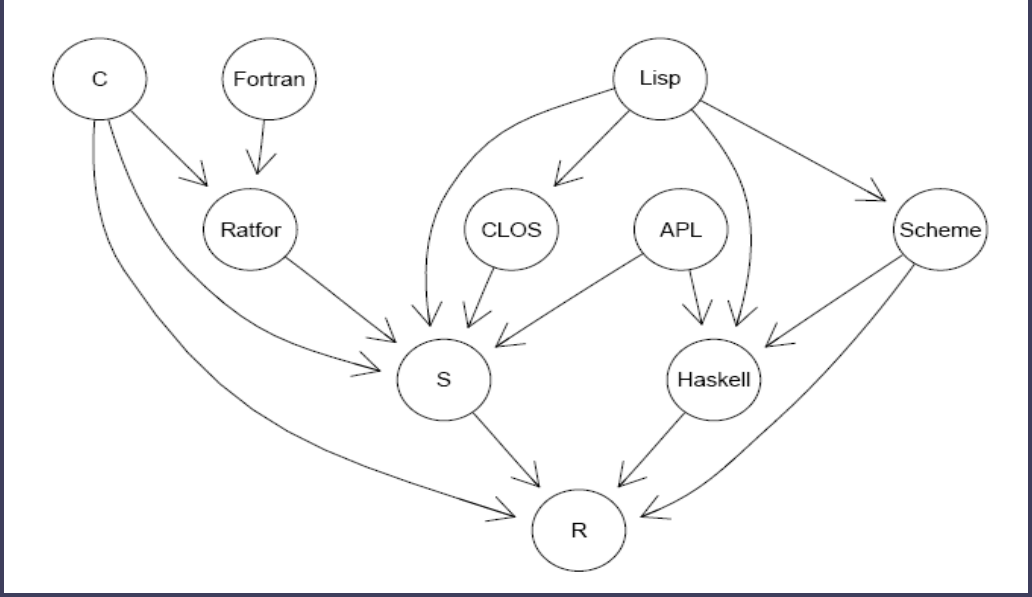
Timeline

- Looking at the history of programming languages is complex, as new languages tend to be informed by all prior developments.

- 1950/60s: **Fortran** (54), **Lisp** (58), Cobol (59), APL (62), Basic (64)
- 1970s: Pascal (70), **C** (72), **S** (75), SQL (78)
- 1980s: C++ (83), Erlang (86), Perl (87)
- 1990s: Haskell (90), **Python** (91), **Java** (91), **R** (93), Ruby (93), Common Lisp (94), PHP (95)
- 2000s: C# (00), Scala (03), Groovy (04), F# (05), **Clojure** (07), Go (09)

R

- S began as a project at Bell Laboratories in 1975, involving John Chambers, Rick Becker, Doug Dunn, Paul Tukey, and Graham Wilkinson.
- R is a “Scheme-like” language. R is written primarily in C and Fortran, although it is being extended through other languages (e.g. Java).



JVM

- The Java Virtual Machine (JVM) is very similar in theory to the Common Language Runtime (CLR) for the .Net framework: it provides a virtual machine for the execution of programs.
- Offers memory and other resource management (garbage collection), JIT, a type system.
- JVM was designed for Java, but it operates on Java bytecode so it can be used by other languages such as Jython, JRuby, Groovy, Scala, and Clojure.

Clojure

- Clojure is a Lisp language that runs on the JVM. It was released in 1997 by Rich Hickey, who continues to be the primary contributor.
  - “Clojure (pronounced like closure) is a modern dialect of the Lisp programming language. It is a general-purpose language supporting interactive development that encourages a functional programming style, and simplifies multithreaded programming. Clojure runs on the Java Virtual Machine and the Common Language Runtime. Clojure honors the code-as-data philosophy and has a sophisticated Lisp macro system.”
- Clojure can be used interactively (REPL) or compiled and deployed as an executable. REPL stands for “read-eval-print loop”.

Incanter

- Incanter is a Clojure-based, R-like platform for statistical computing and graphics, created by David Edgar Liebke.
  - Incanter "leverages both the power of Clojure, a dynamically-typed, functional programming language, and the rich set of libraries available on the JVM for accessing, processing, and visualizing data. At its core are the Parallel Colt numerics library, a multithreaded version of Colt, the JFreeChart charting library, the Processing visualization library, as well as several other Java and Clojure libraries."
- http://www.jstatsoft.org/v13 "Lisp-Stat, Past, Present and Future" in *Journal of Statistical Software* Vol. 13, Dec. 2004
- Why Incanter? The primary reason is easy access to Java.

Comparison

Similarities:

- They can both be used interactively (for Clojure: REPL)
- They are both functional, based on Scheme
- Both languages have type inference
- “Code as data”

Differences:

- R requires more effort to integrate with Java
- R influenced more by C and Fortran
- Clojure can be compiled
- Clojure is not OO, while R has S3, S4, and r.oo
- Clojure has many more data types
- R is more of a DSL

Tradeoffs

Advantages:

- Clojure runs on the JVM, so it can reference any Java library, and can be called by other languages on the JVM
- Clojure natively deals with concurrency
- Vectors/Lists/etc. in Clojure allow you to add/remove

Disadvantages

- Incanter is very immature in comparison; there is no equivalent to CRAN
- Clojure has 339 questions on stackoverflow compared to 562 for R
- Clojure/Incanter are each primarily developed by 1 person; no Core team

Using Clojure/Incanter

- Clojure is a set of jars, so it can be used from the command line by calling java.
- To use Incanter, just load the desired library into a Clojure session:
  - *(use '(incanter core stats charts))*
- Many IDE options:
  - I use Eclipse for all my development (R: StatET, Python: Pydev, C/C++: CDT: http://code.google.com/p/counterclockwise/ and http://www.ibm.com/developerworks/opensource/library/os-eclipse-clojure/index.html
  - Using Emacs: http://incanter-blog.org/2009/12/20/getting-started/

Hello World

- R takes syntax from both Lisp and C.

```
// Java
public void hello(String name) {
        System.out.println("Hello, " + name);
}

; Clojure
(defn hello [name]
        (println "Hello," name))

# R
hello <- function(name) {
        print(paste("Hello,“, name))
}
```

Basic Syntax

		
Statements in R use more of a C-like syntax	(+ 1 2) ; => 3 (range 3) ; => (1 2 3)	`+`(1,2) # => 3 seq(1,3) # => (1 2 3)
Getting help	(doc functionname)	help(functionname)
Checking an object type	(type objectname)	class(objectname)
Timing performance	(time functioncall)	System.time(functioncall)
Browsing the workspace	(ns-publics 'user)	ls()
Nagivating the workspace	(all-ns)	search()

Collections

		
Lists	[def stooges ["Moe" "Larry" "Curly" "Shemp"]]	stooges <- c(“Moe”, “Larry”, “Curly”, “Shemp”)
Vectors	(def stooges ["Moe" "Larry" "Curly" "Shemp"])	stooges <- c(“Moe”, “Larry”, “Curly”, “Shemp”)
Maps	(def popsicle-map {:red :cherry, :green :apple, :purple :grape}) def popsicle-map (sorted-map :red :cherry, :green :apple, :purple :grape))	popsicle.map <- list(“red”=“cherry”, “green”=“apply”, “purple”=“grape”)
Matrix (does not exist as part of Clojure)	(def A (matrix [[1 2 3] [4 5 6] [7 8 9]])) (def A2 (matrix [1 2 3 4 5 6 7 8 9] 3))	A <- matrix(1:9, nrow=3)
Count	(count stooges)	length(stooges)
Filtering	(filter #(> (count %) 3) stooges) (some #(= % "Moe") stooges)	stooges[nchar(stooges)==3] stooges[stooges==“Moe”]

Matrices

		
Matrix (does not exist as part of Clojure)	(def A (matrix [[1 2 3] [4 5 6] [7 8 9]])) (def A2 (matrix [1 2 3 4 5 6 7 8 9] 3))	A <- matrix(1:9, nrow=3)
Dimensions	(dim A) (ncol A) (nrow A)	dim(a) ncol(a) nrow(a)
Filtering	(use 'incanter.datasets) (def iris (to-matrix (get-dataset :iris))) (sel iris 0 0) (sel iris :rows 0 :cols 0) (sel iris :except-cols 1)	iris[1,1] Iris[,-1]

Statistics

		
Quantile	(quantile (range 10))	quantile(1:10)
Sampling	(sample (range 100) :size 10)	sample(1:100, 10)
Mean	(mean (range 10))	mean(1:10)
Skewness	(skewness (range 10))	moments::skewness(rnorm(100))
Regression	(linear-model y x)	lm(y ~ x)
Correlation	(correlation x y) (correlation matrix)	cor(x, y) cor(x)

Loops

- Several different ways to loop in Clojure:

```
;; Version 1
(loop [i 1]
  (when (< i 20)
    (println i)
    (recur (+ 2 i))))

;; Version 2
(dorun (for [i (range 1 20 2)]
        (println i)))

;; Version 3
(doseq [i (range 1 20 2)]
  (println i))
```

- Some examples of the same sequence in R:

```
for(i in seq(1, 20, 2)) print(i)
```

- R also makes heavy usage of the apply family of functions (Clojure also has an apply function):

```
sapply(seq(1, 20, 2), print)
```

- R also has a *while()* function.

Java and Clojure

- Clojure interacts with Java seamlessly. A trivial example:

```
(. javax.swing.JOptionPane (showMessageDialog nil "Hello World"))
```

- Or a slightly more advanced example:

```
(defn fetch-xml [uri]
  (xml-zip
   (parse
    (org.xml.sax.InputSource.
     (java.io.StringReader.
      (slurp* (java.net.URI. (re-gsub #"\s+" "+" (str uri)))))))))
```

R and Java: RJava

- Calling Java code from R (and vice versa) can be done with the RJava package and JRI.

```
# R
helloJavaWorld <- function(){
  hjw <- .jnew("HelloJavaWorld") # create instance of class
  out <- .jcall(hjw, "S", "sayHello") # invoke sayHello method
  return(out)
}

// Java
public class HelloJavaWorld {
        public String sayHello() {
                String result = new String("Hello Java World!");
                return result;
        }
        public static void main(String[] args) {
        }
}
```

Java and R: JRI

- JRI allows you to pass R commands to an R console and get results back:

```
import org.rosuda.JRI.Rengine;
...

        Rengine re=new Rengine(args, false, new Rexecutor());
        REXP x;
        re.eval("data(iris)",false);
        System.out.println(x=re.eval("iris"));
        RVector v = x.asVector();
        if (v.getNames()!=null) {
                        System.out.println("has names:");
                        for (Enumeration e = v.getNames().elements() ; e.hasMoreElements() ;) {
                                        System.out.println(e.nextElement());
                        }
        }
```

Performance

- Problem Number 1 from Project Euler: “Find the sum of all the multiples of 3 or 5 below 1000.” I just changed this to be a count instead:

```
user=> (defn divisible-by-3-or-5? [num]
         (or (== (mod num 3) 0)(== (mod num
         5) 0)))

user=> (time (println (count (filter
         divisible-by-3-or-5? (range
         10000000)))))
4666667
"Elapsed time: 29321.981146 msecs"
nil
```

```
divby3or5 <- function(n) {
      n[(n %% 3) == 0 | (n %% 5) == 0]
}

system.time(print(length(divby3or5(1:10
      000000))))
[1] 4666667
   user  system elapsed
  12.21    0.22   12.70
```

- This is a trivial example, but R significantly outperformed on this simple operation.
- A more thorough benchmarking of Incantor is necessary.

Final Thoughts

- Clojure/Incanter is a very promising programming language based on Lisp. It provides functional programming with a seamless Java integration and native concurrency.
- R has a remarkable user community of dedicated scientists and mathematicians which is continuing to grow. Performance issues can be mitigated by using parallelization (e.g. MPI), and there are efforts to create compilers that promise 10x speed improvements.
- Incanter can be used in the place of R for projects that use relatively basic statistics, and that have a reliance on Java (especially for something that is web-based).

Resources

Some useful resources for Clojure/Incanter

- http://clojure.org/
- http://incanter.org/
- http://java.ociweb.com/mark/clojure/article.html
- http://en.wikibooks.org/wiki/Clojure_Programming/

For R:

- http://r-project.org and http://cran.r-project.org
- http://www.stats.uwo.ca/faculty/murdoch/2864/Flourish.pdf

Lastly, http://rosettacode.org/ has good examples for both languages.