Common Lisp Statistics

Using History to design better data analysis environments

Anthony (Tony) Rossini

Group Head, Modeling and Simulation Statistics
Novartis Pharma AG, Switzerland

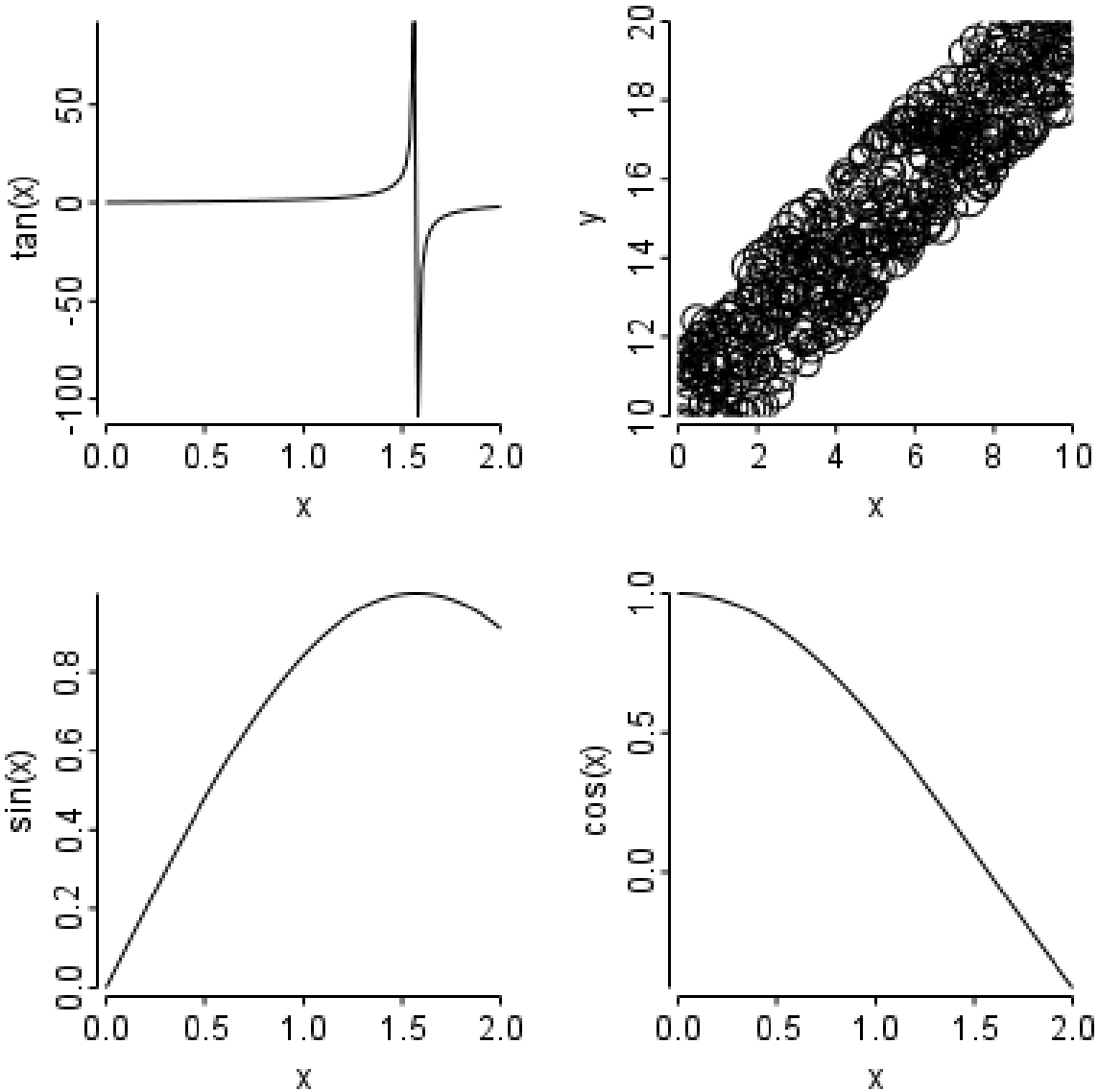
Affiliate Assoc Prof, Biomedical and Health Informatics
University of Washington, USA

DSC 2009, Copenhagen

Is it Vaporware? Not quite

The follow is possible with the help of the open source Common Lisp community, who provided most of the packages, tools, and glue. (Tamas Papp, Raymond Toy, Mark Hoemmomem, and many, many others). Most of the underlying code was written by others, and “composed” by me.

Silly Visualization Example



How?

```
(defparameter *frame2*
   (as-frame (create-xlib-image-context 200 200)
        :background-color +white+))
(bind ((#2A((f1 f2) (f3 f4))
        (split-frame *frame2*
                     (percent 50)
                     (percent 50))))
  (defparameter *f1* f1) ; lower left
  (defparameter *f2* f2) ; lower right   f3   f4
  (defparameter *f3* f3) ; top left      f1   f2
  (defparameter *f4* f4)); top right
```

Functions to Plot

```
(plot-function *f1* #'sin
  (interval-of 0 2)
  :x-title "x" :y-title "sin(x)")
(plot-function *f2* #'cos (interval-of 0 2)
  :x-title "x" :y-title "cos(x)")
(plot-function *f3* #'tan (interval-of 0 2)
  :x-title "x" :y-title "tan(x)")
```

Things to Plot

```
(let* ((n 500)
       (xs (num-sequence
             :from 0 :to 10 :length n))
       (ys (map 'vector
             #'(lambda (x) (+ x 8 (random 4.0)))
             xs))
       (weights
          (replicate #'(lambda () (1+ (random 10)))
                     n 'fixnum))
       (da (plot-simple *f4*
             (interval-of 0 10)
             (interval-of 10 20)
             :x-title "x" :y-title "y")))
  (draw-symbols da xs ys :weights weights))
```

Copying existing graphics

And we generated the figure on the first page by:

```
(xlib-image-context-to-png
   (context *f1*)
   "/home/tony/test1.png")
```

Linear Regression

```
;; Worse than LispStat, wrapping LAPACK's dgelsy:
(defparameter *result1*
   (lm (list->vector-like iron)
       (list->vector-like absorbtion)))
*result*1 =>
((#<LA-SIMPLE-VECTOR-DOUBLE (2 x 1)
 -11.504913191235342
 0.23525771181009483>
  2)

 #<LA-SIMPLE-MATRIX-DOUBLE  2 x 2
 9.730392177126686e-6 -0.0015137871142069 32
 -0.0015137871142069 32 0.3035785121570625 5>

 13 2)
```

DataFrames

```
(defparameter *my-df-1*
  (make-instance 'dataframe-array
 :storage #2A((1 2 3 4 5) (10 20 30 40 50))
 :doc "This is a boring dataframe-array"
 :case-labels (list "x" "y")
 :var-labels (list "a" "b" "c" "d" "e")))

(xref *my-df-1* 0 0) ; API change in progress

(setf (xref *my-df-1* 0 0) -1d0)
```

Numerical Matrices

```
(defparameter *mat-1*
  (make-matrix 3 3
     :initial-contents #2A((2d0 3d0 -4d0)
                           (3d0 2d0 -4d0)
                           (4d0 4d0 -5d0))))

(xref *mat-1* 2 0) ; => 4d0  ; API change
(setf (xref *mat-1* 2 0) -4d0)

(defparameter *xv*
 (make-vector 4 :type :row
   :initial-contents '((1d0 3d0 2d0 4d0))))
```

Macros make the above tolerable

```
(defparameter *xv*
 (make-vector 4 :type :row
   :initial-contents '((1d0 3d0 2d0 4d0))))

; can use defmacro for the following syntax =>

(make-row-vector *xv* '((1d0 3d0 2d0 4d0)))

; or reader macros for the following:
#mrv(*xv* '((1d0 3d0 2d0 4d0)))
```

Outline

Why CLS?

- a component-based structure for statistical computing
- Common Lisp provides a simple, *primitive*, syntax
- Common Lisp provides an amazing number of advanced features that keep getting reinvented in other languages.
- Common Lisp has linkages to many amazing features developed in other languages.
- ability to leverage non-statisticians interested in computing technologies (compilers, protocols, interfaces, libraries, functionality which can be reused for statistical purposes)
- This is a “customization” through packages to support statistical computing, not a independent language. “Ala Carte”, not “Menu”.

Current Functionality

- basic dataframes (similar to R); indexing/slicing API under development.
- Basic regression (similar to XLispStat)
- matrix storage both in foreign and lisp-centric areas.
- LAPACK (small percentage, increasing), working with both matrix storage types
- static graphics (X11) including preliminary grid functionality based on CAIRO. Generation of PNG files from graphics windows.
- CSV file support
- Common Lisp!

Computational Environment Supported

- Should work on Linux, with recent SBCL versions
- Definitely works on bleeding edge Debian (unstable).
- Has worked for weak definitions of “work” on 4 different people’s environments (not quite, but sort of requires a `/home/tony/` !)
- Threaded support on threaded lisps (SBCL/CCL, soon CLISP). But not yet integrated.

Goals

Short Term

- Better integration of data structures with statistical routines (auto-handling with dataframes, rather than manual parsing).

Medium/Long Term

- Support for CLISP (byte-compiled interpreted lisp) and Clozure Common Lisp (formerly OpenMCL)
- high-level Front-end API to a number of matrix and numerical packages and numerical structures (
- constraint system for interactive GUIs and graphics
- full LispStat compatibility (object system partially works; GUI support coming).
- Integrated threading via Bordeaux threads (portable CL API package).

Common Lisp

- Parens provide clear delineation of a **Complete Thought** (functional programming with side effects).
- Lisp-2 (symbols can denote both a separate function and a value)
- ANSI standard (built by committee, but the committee was reasonably smart)
- Many implementations
- Most implementations are interactive **compiled** languages (few are interpreted, and those are usually byte-compiled).
- The Original *Programming with Data* Language (*Programs are Data* and *Data are Executable* also apply).
- advanced, powerful, first-class macros (macros functionally re-write code, allowing for structural clarity and complete destruction of syntax, should that be reasonable)

Common Lisp Packages

(They are packages and called packages, not libraries. Some people can rejoice!)

- infrastructure enhancement: infix-notation, data structures, control and flow structures
- numerics, graphics, GUIs,
- primitive R to CL compiler (which could also be considered an object-code compiler for R); 3 interfaces which embed R within CL.
- Web 2.0 support and reporting facilities (similar to TeX) for PDF.

See `http://www.common-lisp.net/` and `http://www.cliki.org/`. CLS sources can be found on `http://github.com/blindglobe/`

What does NOT work?

Primarily, the reason that we doing this:
Computable and Executable Statistics
(which is the subject of another talk, slides in the backup). But consider XML:

```
<car brand="honda" engine="4cyl">accord</car>
```

becomes

```
; data follows keywords...
(car :brand 'honda :engine "4cyl" accord)
```

Conclusion

Active but slow development, spanning the range of needs:

- Numerics: Linear algebra basics done – full development
- Static graphics: progress being made, have a partial grid-solution, need interactive graphics
- LispStat emulation needs to be finished
- Model specification and unification

Related numerical/statistical projects:

- Incanter : R/LispStat/Omegahat-like system for Clojure (Lisp on the JVM)
- FEMLisp : system/workshop for finite-element analysis modeling using Lisp
- matlisp/LispLab : LAPACK-based numerical linear algebra packages
- GSLL : GNU Scientific Library, Lisp interface.

Followup

I'd be happy to talk with anyone on the following topics:

- Introduction to Common Lisp
- support for new statistical programming environment modalities (subject for another talk).
- computable and executable statistics (code that explains itself and can be parsed to generate knowledge about its claims; "XML's promise")

and if you are interested in getting involved, or trying it out.