Back To The Future: Lisp as a Base for a Statistical Computing System

Ross Ihaka
University of Auckland

Duncan Temple Lang
University of California

Talk Outline

This talk will seek to answer a number of questions about statistical software systems:

- An apology
- Goals
- Implementation strategy
- Performance results
- The future

An Apology

- This talk addresses a perceived need for an improved interactive statistical computing environment.
- The basic requirements of such a system are:
  - it should be possible to easily express a wide variety of computations (i.e. it should be a "language"),
  - it should be possible to define new functionality easily (at run time).
  - good performance.
- Examples of such systems are S, R, etc.

Present Systems are Not Enough

- Present systems are not satisfying all the present demands placed on them.
- They do not handle large problems well:
  - they place too much demand on machine resources,
  - they run too slowly for many purposes.
- In some cases, they do not provide the flexibility we need.
- It is highly likely that they will fail to meet more and more of the demands made of them in the future.

Particular Problems with the S Family

- The whole-object model limits the size of problem which can be handled. This is compounded by the use of call-by-value semantics which can produce multiple copies of data sets in memory.
- It also leads to dependence on algorithms which require the presence of entire data sets.
- The emphasis is very much on vector operations. In fact the family does not include the notion of scalar quantities, meaning the overhead of array access is unavoidable. This makes it difficult to implement many computations efficiently.

Features for a New System

- Speed
- Smaller (relative) footprint
- Foreign function interface
- Parallelism (multicores)
- Support for reasoning about code
- Ability to play nicely with other software

Candidate Systems

- We don't have the resources to build a new system entirely from scratch. We need some giant shoulders to stand on.
- Two candidate systems are:
  - Python
  - Lisp
- Benchmarking shows that (compiled) Lisp is better for some array-based computations (by a factor of 4).
- We also think that Lisp has features which will ultimately make it a better data analysis language.

Lisp as the Base for New System

On the plus side:

- Lisp is a well-established, widely-used system
- There are a multiplicity of high-quality implementations
- There are very good resources explaining Lisp at both the high and low levels.

On the minus side:

- Lisp has an image problem – it is perceived as a "dead" language.
- Because Lisp is an amalgam of features there are some inelegances to deal with.

Particular Lisp Features of Interest

- Compilation to machine code
- Optional type-declaration
- Pass-by-reference semantics
- A flexible generic-function based object system
- Macros
- Support for computing on the language.

A System Vision

- We envision Lisp lying at the heart of a rich computational environment.
- On top there is an interactive language which is really just a thin syntax layer over the Lisp system.
- Desirable Lisp features such as the FFI, and existing interfaces can be exploited directly.
- Commercial tree-shaking technology can be used to produce small stand alone applications.

Implementation Issues

- Syntax
- Lisp idiosyncrasies
- The object model
- Access to standard technologies

A Simple Example

```
defun runif(n &optional a = 0, b = 1)
{
    local x = make_array(n, element_type = 'double)
    local offset = a
    local scale = b − a
    for (i = 0; i < n; ++i; x)
        x[i] = offset + scale * random(1.0)
}

x = runif(10000)
```

A Simple Benchmark

```
defun sum(x)
{
    local sum = 0.0
    local n = length(x)
    for (i = 0; i < n; ++i; sum)
        sum = sum + x[i]
}


time(for(i = 0; i < 10000; ++i) sum(x))
```

A Simple Benchmark

```
defun sum(x)
{
    local sum = 0.0
    local n = length(x)
    for (i = 0; i < n; ++i; sum)
        sum = sum + x[i]
}
```

```
time(for(i = 0; i < 10000; ++i) sum(x))
```

Run time: 6.833 sec

A Simple Benchmark

```
defun sum(x)
{
    local sum = 0.0
    local n = length(x)
    for (i = 0; i < n; ++i; sum)
        sum = sum + x[i]
}
```

```
time(for(i = 0; i < 10000; ++i) sum(x))
```

Run time: 6.833 sec
Run time for R: 133.9 sec (20 times slower than lisp)

The Effect of Type Declarations

```
defun sum(double[*] x)
{
   local double sum = 0.0
   local fixnum n = length(x)
   for (fixnum i = 0; i < n; ++i; sum)
     sum = sum + x[i]
}


time(for(i = 0; i < 10000; ++i) sum(x))
```

The Effect of Type Declarations

```
defun sum(double[*] x)
{
   local double sum = 0.0
   local fixnum n = length(x)
   for (fixnum i = 0; i < n; ++i; sum)
     sum = sum + x[i]
}
```

```
time(for(i = 0; i < 10000; ++i) sum(x))
```

Run time: 0.207 sec (33 times faster than without declarations and 650 times faster than R)

Other Performance Results

- We examined the performance of a lisp based system using a number of examples.
  - Artificial examples (like the one shown here).
  - "Real" examples; generally simulations for random walks, Markov chains etc.
- Some of the real examples showed up to 300-fold performance gains over the best we could get with R.
- Performance in Lisp for the artificial examples was significantly better than we could achieve in competing environments such as Python.

Development Time-Frame

- At present we are very much exploring the feasibility of building a new system.
- We feel that it is prudent to invest a good deal of time in understanding the technology we are proposing to adopt.
- Building a basic computing language can be relatively quick, but there are other tasks which may take longer.

Summary

- This talk represents a progress report on an investigation of how we might go about producing a new statistical computing environment.
- Preliminary experiments have indicated that building a system on top of Common Lisp provides a productive direction to proceed in.
- We believe that systems based on Lisp will be both more sophisticated and better performing than present systems.