Statistics with Julia

Douglas Bates
dmbates

Follow

Block or report user

University of Wisconsin

Madison, WI, U.S.A.

Joined on Aug 20, 2010

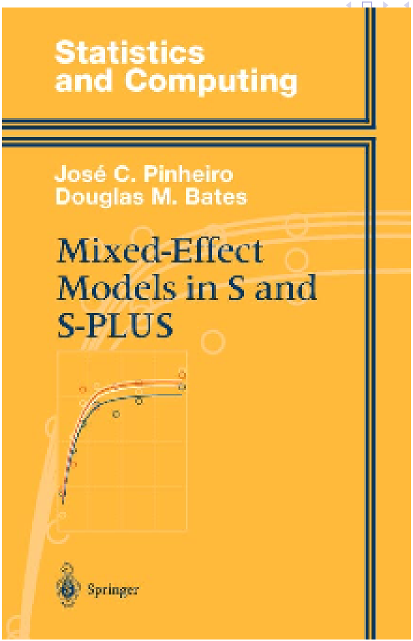
Overview | Repositories 29 | Stars 8 | Fol

Popular repositories

MixedModels.jl
A Julia package for fitting (statistical) mixed-effects models
★ 65 Julia

RePsychLing
Data sets from subject/item type studies in Psychology and Linguistics
★ 13 HTML

ParallelGLM.jl
Parallel fitting of GLMs using SharedArrays


Statistics and Computing
José C. Pinheiro
Douglas M. Bates
Mixed-Effect Models in S and S-PLUS
Springer

Statistics with Julia

R is great, but ...

- The language encourages operating on the *whole object* (i.e. vectorized code). However, some tasks (e.g. MCMC) are not easily vectorized.
- Unvectorized R code (*for* and *while* loops) is slow.
- Techniques for large data sets – parallelization, memory mapping, database access, map/reduce – can be used but not easily. *R* is single threaded and most likely will stay that way.
- *R* functions should obey *functional semantics* (not modify arguments). Okay until you have very large objects on which small changes are made during parameter estimation.
- Sort-of object oriented using generic functions but implementation is casual. Does garbage collection but not based on reference counting.
- The real work is done in underlying C code and it is not easy to trace your way through it.

Statistics with Julia

Fast development vs. fast execution - Can we have both?

- The great advantage of *R*, an interactive language with dynamic types, is ease of development. High level language constructs, ease of testing small pieces of code, a read-eval-print loop (REPL) versus an edit-compile-run loop.
- Compilation to machine code requires static types. *C++* allows templates instead of dynamic types, and recent libraries like *STL*, *Boost*, *Rcpp*, *Armadillo*, *Eigen* use template metaprogramming for flexibility. But those who value their sanity leave template metaprogramming to others.
- *Julia* has a wide range of types, including user-defined types and type hierarchies, and uses multiple dispatch on generic functions with sophisticated type inference to emit code for the LLVM JIT.
- In my opinion *Julia* provides the best of both worlds and is the technical programming language of the future.

Statistics with Julia

Julia version using the Distributions package

```
using Distributions
function jgibbs(N::Integer, thin::Integer)
    mat = Array(Float64,(N,2))
    x = y = 0.
    for i in 1:N
        for j in 1:thin
            x = rand(Gamma(3.,1./(y*y+4.))) #shape/scale
            y = rand(Normal(1./(x+1.),1./sqrt(2.(x+1.))))
        end
        mat[i,1] = x; mat[i,2] = y
    end
    mat
end
```

- In *Julia* `0` is an integer and `0.` is floating point. *R* has the peculiar convention that `0` is floating point and `0L` is an integer.