How To Make Your Code Faster

Yixuan Qiu @ GSO Student Seminar

February 11, 2016

Content

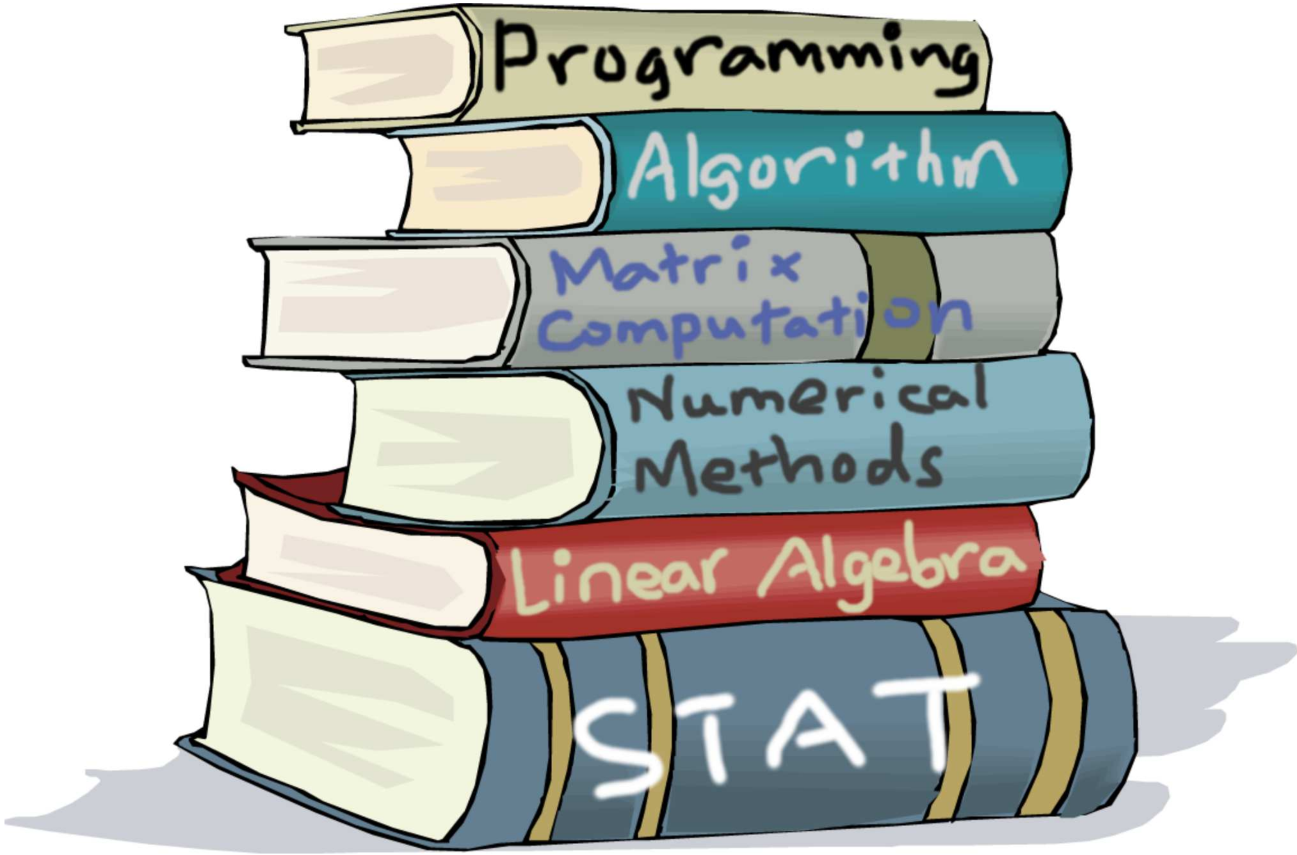
- Why Is My Code So Slow?
- Approaches To Accelerating
- Remarks

How You Do Research

- You design a model
- You write the code
- You run the code
- You wait
- You wait
- You wait...

Why Is My Code So Slow?

There are many factors that may play a role…



Possible Reasons

- Is the algorithm efficient?
- Are the software/libraries good enough?
- Is the programming language used appropriate for the problem?

Approaches To Accelerating

- Better designed algorithm
- High performance software and libraries
- More efficient languages

Why Algorithm Matters

Example: Regression

$$\hat{y} = X(X'X)^{-1}X'y$$

- Algorithm 1: Direct translation

```
reg1 = function(x, y)
{
    x %*% solve(t(x) %*% x) %*% t(x) %*% y
}
```

- Algorithm 2: From right to left

```
reg2 = function(x, y)
{
    x %*% (solve(t(x) %*% x) %*% (t(x) %*% y))
}
```

Regression Example cont.

- Algorithm 3: No explicit matrix inverse, special matrix structure

```
reg3 = function(x, y)
{
    x %*% solve(crossprod(x), crossprod(x, y))
}
```

Benchmark

```r
library(rbenchmark)
set.seed(123)
n = 2000
p = 500
x = matrix(rnorm(n * p), n)
y = rnorm(n)
benchmark(reg1(x, y), reg2(x, y), reg3(x, y),
          columns = c("test", "replications", "elapsed", "relative"),
          replications = 10)
```

```
        test replications elapsed relative
1 reg1(x, y)           10   18.33    6.891
2 reg2(x, y)           10    3.93    1.477
3 reg3(x, y)           10    2.66    1.000
```

Golden Rules

- Low dimension first in matrix multiplication
- Use `crossprod(x, y)` for $X'y$
- Use `crossprod(x)` for $X'X$
- Use `solve(A, b)` for $A^{-1}b$

Example: PCA

- Calculating the first k PC's of a data matrix X
- Algorithm 1:
  - Form covariance matrix $V = Cov(X)$
  - Calculate eigen decomposition $V = \Gamma\Lambda\Gamma'$
  - Extract the first k columns of Γ (the loadings)
  - Compute PC scores $S = X\Gamma_k$
- Implemented in R function `princomp()`

PCA Example cont.

- Algorithm 2:
  - Subtract column means from X to get centered matrix X_c
  - **Partial** SVD on X_c: $X_c \rightarrow U_k D_k V_k'$
  - Compute PC scores $S = X_c V_k$

```
library(rARPACK)  ## For svds()
pca2 = function(x, k)
{
    xc = scale(x, center = TRUE, scale = FALSE)
    decomp = svds(xc, k, nu = 0, nv = k)
    xc %*% decomp$v
}
```

Benchmark

```
benchmark(princomp(x),
          pca2(x, 10),
          columns = c("test", "replications", "elapsed", "relative"),
          replications = 10)
```

```
         test replications elapsed relative
2 pca2(x, 10)           10    2.94    1.000
1 princomp(x)           10    8.90    3.027
```

Golden Rules

- Do not compute what is unnecessary
- Sometimes this can be hard

Software/Libraries

Libraries To Use

- Even for the same algorithm, different software/libraries may provide different performance
- Switching to highly optimized libraries is usually easy
- Sometimes you can even keep the code unchanged and gain performance improvements for free!

Linear Algebra: BLAS vs OpenBLAS

- R uses a library called **BLAS** to do many linear algebra computation
- For example matrix-matrix product
- OpenBLAS is a highly optimized version of BLAS
- You can simply replace the BLAS contained in R by OpenBLAS

Benchmark

```r
set.seed(123)
x = matrix(rnorm(2000^2), 2000)
system.time(crossprod(x))
```

- Using original BLAS

```
user   system elapsed
3.95    0.00    3.95
```

- Using OpenBLAS with 4 threads

```
user   system elapsed
0.76    0.00    0.20
```

Reading Data: readr

- The `readr` package provides alternatives to `read.table()` and `read.csv()`
- Functions `read_delim()` and `read_csv()`
- Usually 5x to 10x faster

Benchmark

- Kaggle competition data (https://www.kaggle.com/c/homesite-quote-conversion/data)
- Training set: 197 MB, 260753 observations, 299 variables

```
library(readr)
system.time(read.csv("train.csv"))  ## Base R
```

```
   user  system elapsed
  27.24    0.81   28.07
```

```
system.time(read_csv("train.csv"))  ## readr
```

```
   user  system elapsed
   4.77    0.27    5.03
```

Working With Data Frames: dplyr

- One of the most useful tools in R for data pre-processing and summarization
- Very efficient, no for loops
- Hundreds of documents on web
- An introduction

Summarizing Matrices: matrixStats

- Functions operating on rows and columns of matrices
- Sums and means (in base R), variances, medians, ranks, order statistics, etc.
- Usually 5x to 10x faster than `apply()`

Benchmark

```
library(matrixStats)
benchmark(apply(x, 2, sd),
          colSds(x),
          columns = c("test", "replications", "elapsed", "relative"))
```

```
             test replications elapsed relative
1 apply(x, 2, sd)          100    2.94    8.647
2       colSds(x)          100    0.34    1.000
```

Fitting Least Squares Regression: RcppEigen

- Using the high performance C++ library Eigen for linear algebra
- Efficient algorithm (Cholesky decomposition)

Benchmark

```r
library(RcppEigen)
benchmark(lm.fit(x, y),
          reg3(x, y),
          fastLmPure(x, y, method = 2),
          columns = c("test", "replications", "elapsed", "relative"),
          replications = 10)
```

```
                          test replications elapsed relative
3 fastLmPure(x, y, method = 2)           10    0.68    1.000
1                 lm.fit(x, y)           10    3.81    5.603
2                   reg3(x, y)           10    2.71    3.985
```

Languages

When R Is Still Slow…

- Monte Carlo methods, iterative methods, nested loops, etc.
- You always have the freedom to switch to other languages
- May be a bit hard to learn in the beginning, but deserves
- My recommendations: C++ and Julia

C++

- Extremely hard to **master**
- But **NOT** hard to **get started**
- Nice and convenient interface to R (Rcpp)
- Learning a small subset of C++ can make a big difference

Julia

- **Extremely** easy to learn
- Syntax similar to R and Matlab
- Fast, really fast
- You will love it

Example: Gibbs Sampling

- Example from Darren Wilkinson's Blog

$$f(x,y) \propto x^2 \exp\{-xy^2 - y^2 + 2y - 4x\}$$

- $X|Y \sim \frac{1}{y^2+4} Gamma(3)$
- $Y|X \sim N\left(\frac{1}{1+x}, \frac{1}{2(1+x)}\right)$

R Code

```
gibbs = function(N, thin)
{
    iter = 1:N
    xvec = numeric(N)
    yvec = numeric(N)
    x = 0
    y = 0
    for(i in 1:N)
    {
        for(j in 1:thin)
        {
            x = rgamma(1, shape = 3, scale = 1 / (y * y + 4))
            y = rnorm(1, 1 / (x + 1), 1 / sqrt(2 * x + 2))
        }
        xvec[i] = x
        yvec[i] = y
    }
    data.frame(Iter = iter, x = xvec, y = yvec)
}
```

C++ Code With Rcpp

```
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
DataFrame gibbs_rcpp(int N, int thin) {
    IntegerVector iter(N);
    for(int i = 0; i < N; i++)
        iter[i] = i + 1;

    NumericVector xvec(N);
    NumericVector yvec(N);

    double x = 0.0, y = 0.0;
    for(int i = 0; i < N; i++) {
        for(int j = 0; j < thin; j++) {
            x = R::rgamma(3.0, 1.0 / (y * y + 4.0));
            y = R::rnorm(1.0 / (x + 1.0), 1.0 / std::sqrt(2.0 * x + 2.0));
        }
        xvec[i] = x;
        yvec[i] = y;
    }

    return DataFrame::create(Named("Iter") = iter, Named("x") = xvec, Named("y") = yvec);
}
```

Julia Code

```
using Distributions
using DataFrames

function gibbs_jl(N, thin)
    iter = 1:N
    xvec = zeros(N)
    yvec = zeros(N)
    x = 0.0
    y = 0.0
    for i = 1:N
        for j = 1:N
            x = rand(Gamma(3.0, 1.0 / (y * y + 4.0)))
            y = rand(Normal(1.0 / (x + 1.0), 1.0 / sqrt(2.0 * x + 2.0)))
        end
        xvec[i] = x
        yvec[i] = y
    end
    DataFrame(Iter = iter, x = xvec, y = yvec)
end
```

Benchmark - R and C++

```r
set.seed(123)
system.time(res_r <- gibbs(1000, 1000))
```

```
   user  system elapsed 
   5.17    0.00    5.17 
```

```r
set.seed(123)
system.time(res_rcpp <- gibbs_rcpp(1000, 1000))
```

```
   user  system elapsed 
   0.20    0.00    0.21 
```

```r
identical(res_r, res_rcpp)
```

```
[1] TRUE
```

Benchmark - Julia

```
srand(123)
@elapsed res_jl = gibbs_jl(1000, 1000)
```

```
0.098107808
```

Plots - R and Julia

- R

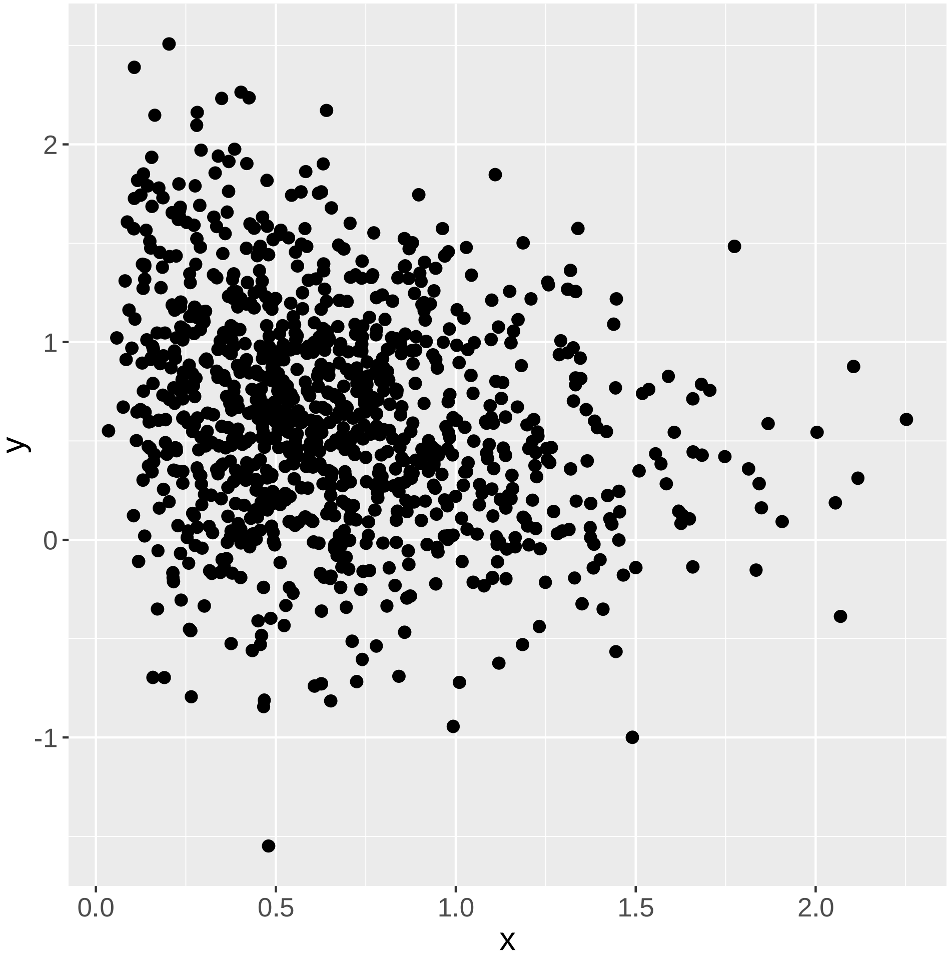
```
library(ggplot2)
ggplot(res, aes(x = x, y = y)) + geom_point()
```

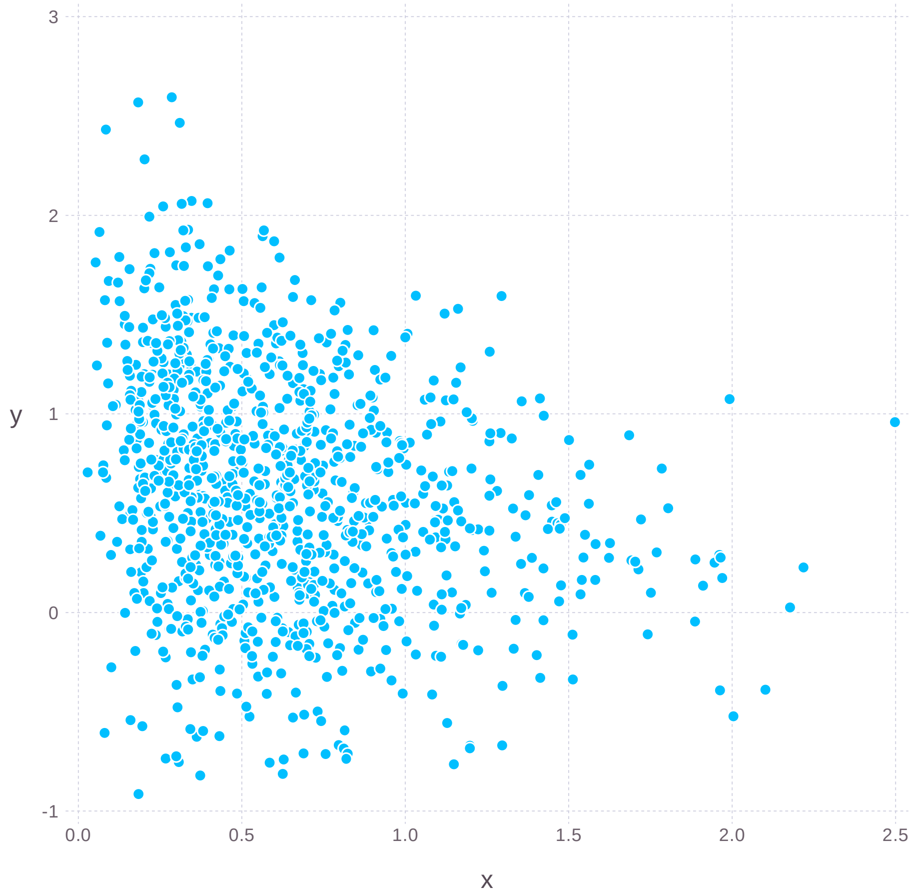
- Julia

```
using Gadfly

draw(SVG("gibbs_jl.svg", 6inch, 6inch),
     plot(res_jl, x = "x", y = "y", Geom.point))
```

Plots - R and Julia





Final Remarks

Some Thoughts

- When we compute for a model…
- First think of a good algorithm
- Then try to use well-developed software and libraries
- If still insufficient, try to learn and use other languages
    - It is not as hard as you thought

Q & A

Thanks!