Applied Statistical Programming

Code profiling

T. Ryan Johnson

April 18, 2022

Washington University in St. Louis

Today's lecture

You will learn

- what code profiling is,
- what call stacks are,
- how to read flame graphs,
- the basics of *`data.table()`*

Part I

Code profiling

Code profiling

Code profiling is a way to measure storage and time complexity of a program with respect to its instruction set.

Answers questions like

1. How frequently is a method called?
2. How many times is a method called?
3. How much memory is allocated?
4. When does garbage collection happen?

Code profiling

How do I make my R code faster?

$\longrightarrow$ How long does it take to run?

Code profiling

```
# 400k rows, 150 columns
data <- as.data.frame(
    matrix(rnorm(4e5*150, mean=5),
    ncol=150))

normCols <- function(d){
    # Get column means
    means <- apply(d,2,mean)

    # De-mean each column
    for (i in seq_along(means)){
        d[,i] <- d[,i] - means[i]
    }
}
data_demeaned <- normCols(data)
```

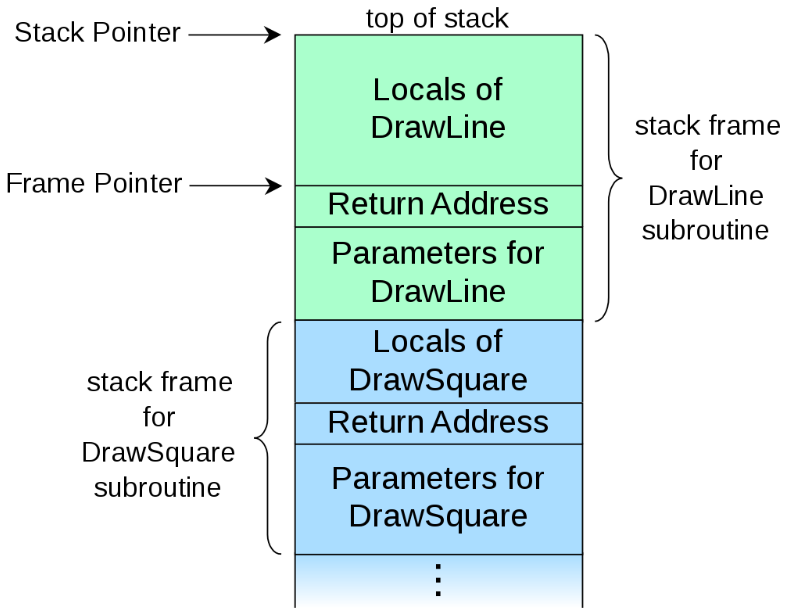
```
# system.time isn't very informative
system.time({

normCols <- function(d){
    # Get column means
    means <- apply(d,2,mean)

    # De-mean each column
    for (i in seq_along(means)){
        d[,i] <- d[,i] - means[i]
    }
}
data_demeaned <- normCols(data)

})
```

Code profiling

How do I make my R code faster?

$\longrightarrow$ What parts of the code slow things down?

The parts of the code: Call Stacks



```r
install.packages("profvis")
library(profvis)

profvis({
    data(diamonds, package = "ggplot2")

    plot(price ~ carat, data = diamonds)
    m <- lm(price ~ carat, data = diamonds)
    abline(m, col = "red")
})
```

Our running example

```r
profvis({

normCols <- function(d){
    # Get column means
    means <- apply(d,2,mean)
    # De-mean each column
    for (i in seq_along(means)){
        d[,i] <- d[,i] - means[i]
    }
}
normCols(data)

})
```

```r
# Four ways to get column means
profvis({
    means <- apply(data, 2, mean)
    means <- colMeans(data)
    means <- lapply(data, mean)
    means <- vapply(data, mean, numeric(1))
})
```

```r
# Profile using vapply instead
profvis({
    means <- vapply(data, mean, numeric(1))

    for (i in seq_along(means)){
        data[,i] <- data[,i] - means[i]
    }
})
```

```
gramschmidt <- function(x) {
  x <- as.matrix(x)
  n <- ncol(x)
  m <- nrow(x)
  q <- matrix(0, m, n)
  r <- matrix(0, n, n)
  for (j in 1:n) {
    v = x[,j]
    if (j > 1) {
      for (i in 1:(j-1)) {
        r[i,j] <- t(q[,i]) %*% x[,j]
        v <- v - r[i,j] * q[,i]
      }}
    r[j,j] <- sqrt(sum(v^2))
    q[,j] <- v / r[j,j]
  }
  return(list('Q'=q, 'R'=r))    }
```

`gramschmidt()` PROFILE

```
profvis({
    set.seed(1234)
    n <- 1000
    M <- matrix(rnorm(n*n, mean=5), ncol=n)
    QR <- gramschmidt(M)
})
```

Detailed `gramschmidt()` profile

```
profvis({
    n <- 1000
    M <- matrix(rnorm(n**2, mean=5), ncol=n)
    m <- nrow(M)
    q <- matrix(0, m, n)
    r <- matrix(0, n, n)
    for (j in 1:n) {
        v = M[,j]
        if (j > 1) {
            for (i in 1:(j-1)) {
            r[i,j] <- t(q[,i]) %*% M[,j]
            v <- v - r[i,j] * q[,i]
        }}
        r[j,j] <- sqrt(sum(v^2))
        q[,j] <- v / r[j,j]
    }   })
```

Improving `gramschmidt()` performance

The algorithm for *`gramschmidt()`* is not numerically stable.
Instead use:

- Householder transforms (dense matrices)
- Givens rotations (sparse matrices)

Code profiling doesn't help with choosing the better implementation.

Part II

data.table

`data.table` PACKAGE

The benefits of *`data.table`*:

- subset rows,
- select and compute on columns, and
- perform aggregations by group

```
install.packages("data.table")
library(data.table)
input <- if (file.exists("flights14.csv")) {
    "flights14.csv"
} else {
    "https://raw.githubusercontent.com
    /Rdatatable/data.table/master/vignettes/
    flights14.csv"
}
flights <- fread(input)
flights
dim(flights)
```

```
# Suppose DT is your data table that you fread in
DT[i, j, by]

#  R:                 i                  j        by
#SQL:  where | order by   select | update  group by
```

Example: subsetting

```
# Get all the flights with “JFK”
# as the origin airport
# in the month of June.
ans1 <- flights[origin == "JFK" & month == 6L]
head(ans1)

# Get the first two rows from flights.
ans2 <- flights[1:2]
ans2
```

```
# Sort flights first by column origin
# in ascending order,
# and then by dest in descending order:

ans3 <- flights[order(origin, -dest)]
head(ans3)

# Advanced computations
# How many trips have had total delay < 0?

ans4 <- flights[,sum((arr_delay + dep_delay) < 0)]
ans4
```

```
# Calculate the average arrival and
# departure delay for all flights with “JFK”
# as the origin airport in the month of June.

ans5 <- flights[origin == "JFK" & month == 6L,
           .(m_arr = mean(arr_delay),
           m_dep = mean(dep_delay))]
ans5
```

Example: Selecting

```
# Get a vector
dat1 = flights[ , origin]

# Get a data table
dat2 = flights[ , .(origin)]

# Get multiple variables
dat3 = flights[, .(origin, year, month, hour)]
```

Activity

Run the R code that implements your EM algorithm from last in class activity, and continue working on the `Rcpp` implementation.

Include the resulting `.profvis` profile of your code with your in EM algorithm in class activity submission on Wednesday, April 20.

Links

Click this link to go to the references.

Graphics

Washington University in St. Louis

References

Shaw, R. S. (2007). Example layout of a call stack showing stack frames and frame pointer. Retrieved April 17, 2022 from *https://commons.wikimedia.org/wiki/File:Call_stack_layout.svg*.