# Problem Set 3

Jinze Gu

October 22, 2013

```
# PROBLEM ONE a).
library(rbenchmark)
logf1 <- function(k, n, p, t) {
    if (n != max(k)) {
        lchoose(n, k) + k * log(k) + (n - k) * log(n - k) - n * log(n) + t * (n * log(n) - k * log(k) -
            (n - k) * log(n - k)) + k * t * log(p) + (n - k) * t * log(1 - p)
    } else {
        n * t * log(p)
    }
}
# Comment on why taking log scale: since in some extreme value, R shows 'NaN'in exponential
# scale,but if I take log scale, most of value can be shown.
# This is the function used to exponentiate the 'log' function
f <- function(x, y) {
    exp(logf1(k = x, n = y, p = 0.3, t = 0.5))
}
# This is the function used to get the final answer
g <- function(n) {
    t <- function(x) {
        f(x, y = n)
    }
    ans <- unlist(lapply(c(1:n), t))
    return(sum(ans))
}
g(100)

## [1] 1.419

# b).  Full vectorized fashion with no loops or apply() I may need to delete this function since
# it does not use 'log'
f1 <- function(k, n, p, t) {
    choose(n, k) * (k^k * (n - k)^(n - k)/n^n)^(1 - t) * p^(k * t) * (1 - p)^((n - k) * t)
}
g1 <- function(k, n) {
    return(sum(f1(k, n, 0.3, 0.5)))
}
# This is the comparison between program in a) and b)
benchmark(g(2000), g1(c(1:2000), 2000), columns = c(1:5))

##                  test replications user.self sys.self elapsed
## 1             g(2000)          100     2.778    0.011   2.788
## 2 g1(c(1:2000), 2000)          100     0.104    0.000   0.105
```

```
# c). I tried it in arwen, but it seems I can not speed up my program faster than the time 0.11
# benchmark(g1(a,2000),columns = c(1:5))
test replications user.self sys.self elapsed
1
# g1(a, 2000) 100 0.204 0 0.206
```

```
# PROBLEM TWO a).
set.seed(0)
ranwalk1 <- function(x, p) {
    if (!is.integer(x)) {
        print("The input should be an integer")
    }
    if (x <= 0) {
        print("Negative number and zero are not valid input")
    }
    n <- as.numeric(as.integer(abs(x)) + 1)
    x <- vector("numeric", n)
    y <- vector("numeric", n)
    for (i in c(min(n, 2):max(n, 2))) {
        # In case the input is 0
        x[i] <- x[max((i - 1), 0)]
        y[i] <- y[max((i - 1), 0)]
        ran <- sample(c(1, 2, 3, 4), 1, prob = c(0.25, 0.25, 0.25, 0.25))
        if (ran == 1) {
            x[i] <- x[i] + 1
        } else if (ran == 2) {
            x[i] <- x[i] - 1
        } else if (ran == 3) {
            y[i] <- y[i] + 1
        } else {
            y[i] <- y[i] - 1
        }
    }
    if (p) {
        plot(x, y, type = "l")
    }
    return(c(x[n], y[n]))
}
Rprof("ranwalk.out", memory.profiling = 1, line.profiling = 1)
ranwalk1(10000L, 1)

## [1] 100 180

Rprof(NULL)
summaryRprof("ranwalk.out", memory = "none", lines = "hide")

## $by.self
##              self.time self.pct total.time total.pct
## "ranwalk1"        0.08     50.0       0.16     100.0
## "sample.int"      0.06     37.5       0.06      37.5
## "identical"       0.02     12.5       0.02      12.5
##
```

```
## $by.total
##                       total.time total.pct self.time self.pct
## "ranwalk1"                  0.16     100.0      0.08     50.0
## "block_exec"                0.16     100.0      0.00      0.0
## "call_block"                0.16     100.0      0.00      0.0
## "doTryCatch"                0.16     100.0      0.00      0.0
## "eval"                      0.16     100.0      0.00      0.0
## "evaluate_call"             0.16     100.0      0.00      0.0
## "evaluate"                  0.16     100.0      0.00      0.0
## "handle"                    0.16     100.0      0.00      0.0
## "in_dir"                    0.16     100.0      0.00      0.0
## "knit"                      0.16     100.0      0.00      0.0
## "process_file"              0.16     100.0      0.00      0.0
## "process_group.block"       0.16     100.0      0.00      0.0
## "process_group"             0.16     100.0      0.00      0.0
## "try"                       0.16     100.0      0.00      0.0
## "tryCatch"                  0.16     100.0      0.00      0.0
## "tryCatchList"              0.16     100.0      0.00      0.0
## "tryCatchOne"               0.16     100.0      0.00      0.0
## "withCallingHandlers"       0.16     100.0      0.00      0.0
## "withVisible"               0.16     100.0      0.00      0.0
## "sample.int"                0.06      37.5      0.06     37.5
## "sample"                    0.06      37.5      0.00      0.0
## "identical"                 0.02      12.5      0.02     12.5
## "<Anonymous>"               0.02      12.5      0.00      0.0
## "fun"                       0.02      12.5      0.00      0.0
## "handle_output"             0.02      12.5      0.00      0.0
## "plot_snapshot"             0.02      12.5      0.00      0.0
## "plot.default"              0.02      12.5      0.00      0.0
## "plot.new"                  0.02      12.5      0.00      0.0
## "plot"                      0.02      12.5      0.00      0.0
##
## $sample.interval
## [1] 0.02
##
## $sampling.time
## [1] 0.16
```

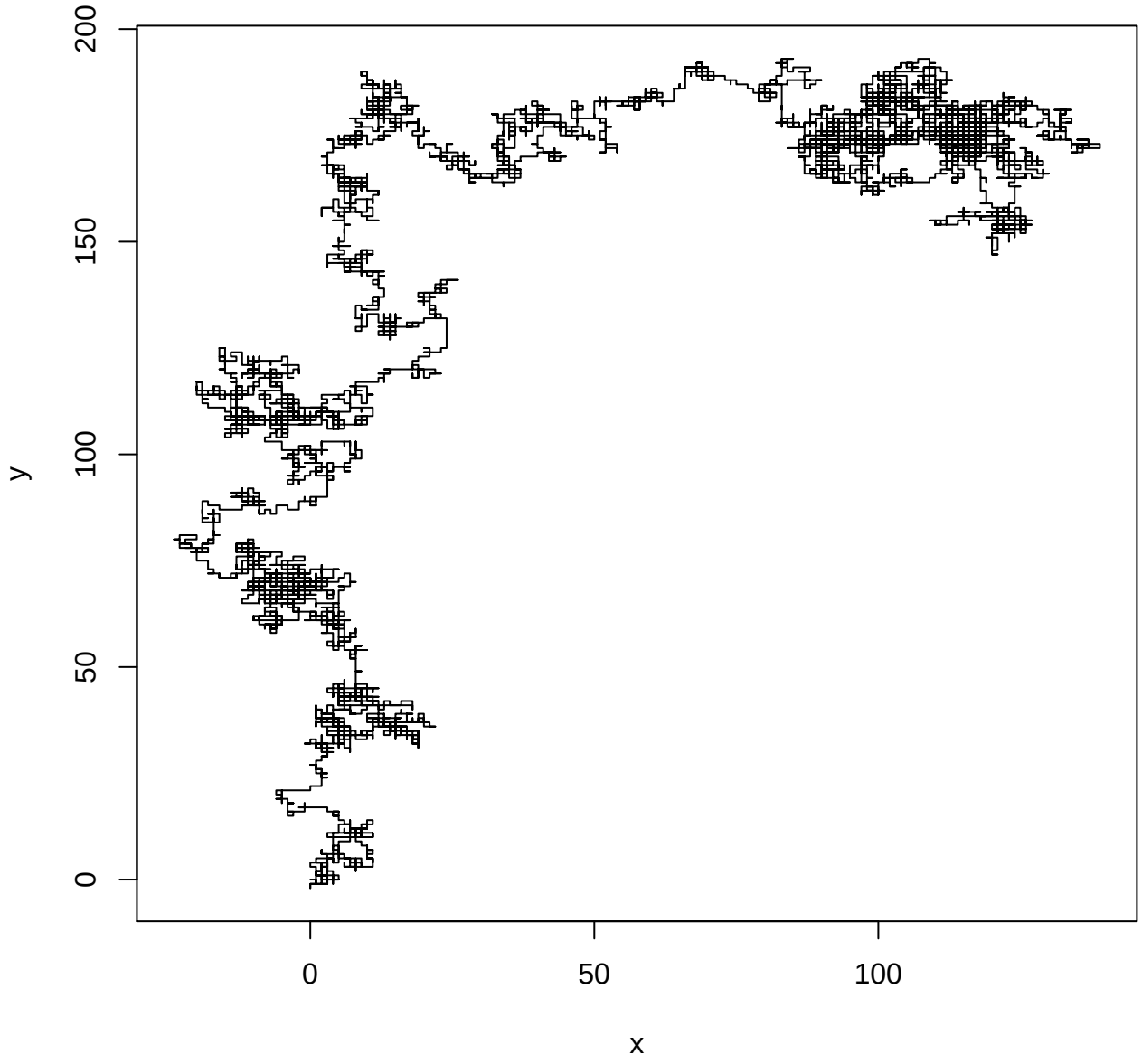
```r
# b). Some improvment based on ranwalk1
ranwalk2 <- function(x, p) {
    if (!is.integer(x) | x <= 0) {
        print("The input should be an integer")
    }
    n <- as.numeric(as.integer(abs(x)) + 1)
    xy <- matrix(0, n, 2)
    for (i in c(min(n, 2):max(n, 2))) {
        # In case the input is 0
        xy[i, 1] <- xy[max((i - 1), 0), 1]
        xy[i, 2] <- xy[max((i - 1), 0), 2]
        ran <- sample(c(1, 2, 3, 4), 1, prob = c(0.25, 0.25, 0.25, 0.25))
        if (ran == 1) {
            xy[i, 1] <- xy[i, 1] + 1
        } else if (ran == 2) {
            xy[i, 1] <- xy[i, 1] - 1
```

```
        } else if (ran == 3) {
            xy[i, 2] <- xy[i, 2] + 1
        } else {
            xy[i, 2] <- xy[i, 2] - 1
        }
    }
    if (p) {
        plot(xy, type = "l")
    }
    return(xy[n, ])
}
benchmark(ranwalk1(1000L, 0), ranwalk2(1000L, 0), columns = c(1:5))
```

```
##                test replications user.self sys.self elapsed
## 1 ranwalk1(1000, 0)          100     1.577    0.004   1.582
## 2 ranwalk2(1000, 0)          100     1.610    0.003   1.613
```

```
# It seems, ranwalk2 is not an improvment, and matrix is no better than vector in storing values
# and speeding up my program
# New way to solve the problem without forloop
ranwalk3 <- function(n, p) {
    right <- c(1, 0)
    up <- c(0, 1)
    left <- c(-1, 0)
    down <- c(0, -1)
    start <- c(0, 0)
    direction <- cbind(left, up, right, down)
    d <- sample(c("left", "up", "right", "down"), as.integer(abs(n)), replace = TRUE)
    x <- cumsum(direction[1, d])
    y <- cumsum(direction[2, d])
    path <- rbind(start, cbind(x, y))
    if (p) {
        plot(path, type = "l")
    }
    return(path)
}
# The comparison is as follows
benchmark(ranwalk1(10000L, 0), ranwalk2(10000L, 0), ranwalk3(10000, 0), columns = c(1:5))
```

```
##                 test replications user.self sys.self elapsed
## 1 ranwalk1(10000, 0)          100    15.010    0.031  15.042
## 2 ranwalk2(10000, 0)          100    15.748    0.034  15.782
## 3 ranwalk3(10000, 0)          100     0.161    0.016   0.178
```




```r
# PROBLEM THREE
set.seed(0)
# This is the random walk function we used
ranwalk.rw <- function(numbersteps) {
    right <- c(1, 0)
    up <- c(0, 1)
    left <- c(-1, 0)
    down <- c(0, -1)
    start <- c(0, 0)
    direction <- cbind(left, up, right, down)
    d <- sample(c("left", "up", "right", "down"), as.integer(abs(numbersteps)), replace = TRUE)
```

```
    x <- cumsum(direction[1, d])
    y <- cumsum(direction[2, d])
    path <- rbind(start, cbind(x, y))
    return(path)
}
# This is my constructor function
rw <- function(numbersteps) {
    s <- ranwalk.rw(numbersteps)
    obj <- list(numberofsteps = numbersteps, finalstep = s[(numbersteps + 1), ], path = s)
    class(obj) <- "rw"
    rm(s)
    return(obj)
}
# This is the definition of 'print' in rw.class
print.rw <- function(object) {
    with(object, cat("This is a simulation of random walk of", numberofsteps, "steps", ".\n", "The final step
        finalstep, "\n"))
    print(object$path)
}
# This is definition of 'plot' in rw.class
plot.rw <- function(object) {
    plot(object$path, type = "b", xlab = "Horizontal Move", ylab = "Vertical Move")
}
# This is definition of '[' in rw.class
`[.rw` <- function(object, i) return(object$path[(i + 1), ])
`start<-` <- function(x, ...) UseMethod("start<-")
`start<-.rw` <- function(object, value) {
    object$path[, 1] <- object$path[, 1] + value[1]
    object$path[, 2] <- object$path[, 2] + value[2]
    object$finalstep <- object$finalstep + value
    print(object)
}
rw(10)
```

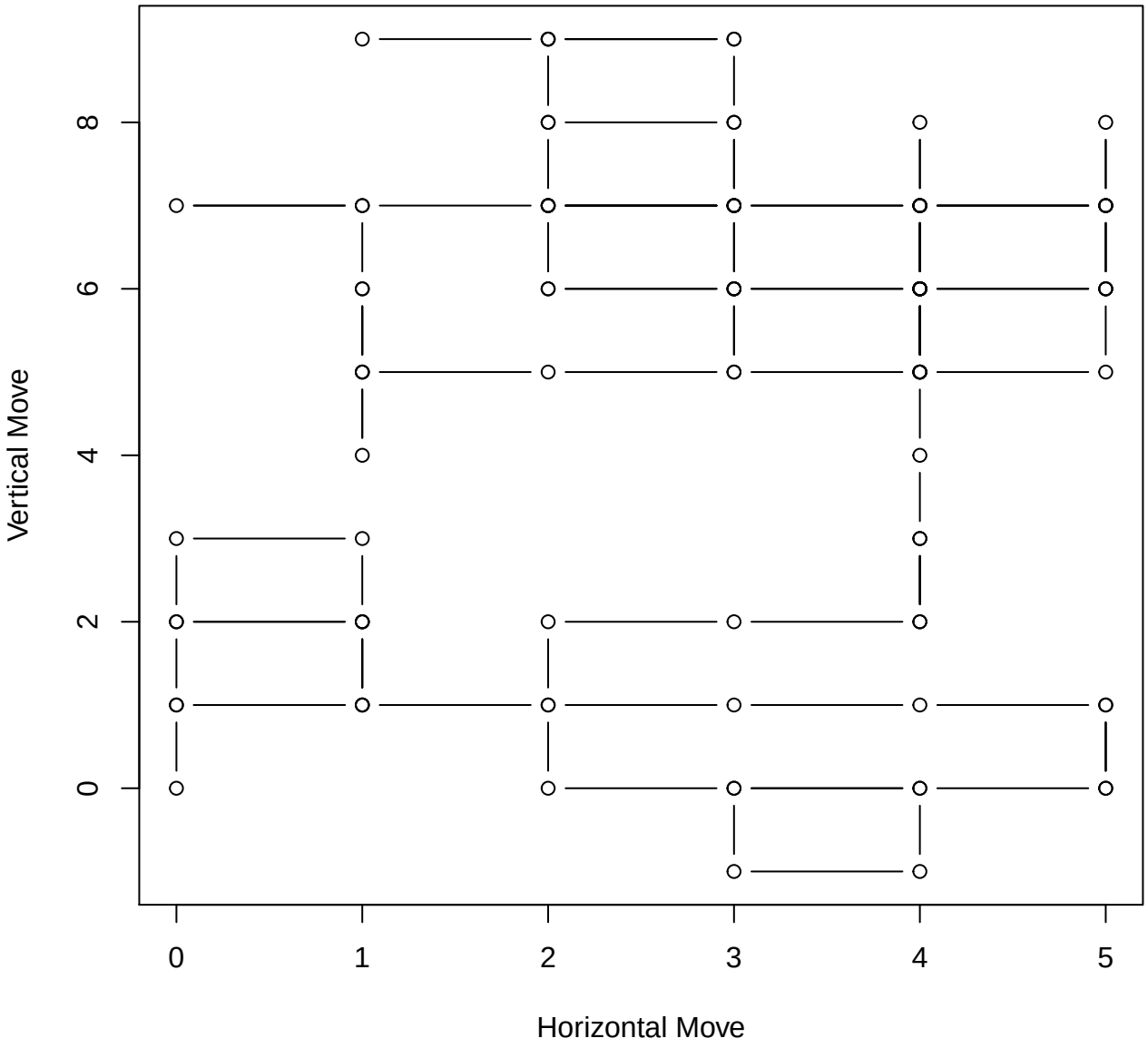
```
## This is a simulation of random walk of 10 steps .
##  The final step is 2 -2
##       x  y
## start 0  0
## down  0 -1
## up    0  0
## up    0  1
## right 1  1
## down  1  0
## left  0  0
## down  0 -1
## down  0 -2
## right 1 -2
## right 2 -2
```

```
rw(10)[5]
```

```
##  x  y
## -2  1
```

```
s <- rw(10)
```

```r
start(s) <- c(5, 7)

## This is a simulation of random walk of 10 steps .
##  The final step is 3 7
##       x y
## start 5 7
## down  5 6
## down  5 5
## left  4 5
## right 5 5
## left  4 5
## up    4 6
## up    4 7
## left  3 7
## up    3 8
## down  3 7

plot(rw(100))
```




```
# PROBLEM FOUR a).
object.size(sample(c(seq(1, 20, by = 1), NA), 1e+07, replace = TRUE))

## 80000040 bytes

fastcount <- function(xvar, yvar) {
    # When we input the value, we have created two 80Mb objects here, so the memory used here is
    # 160Mb
    nalineX <- is.na(xvar)
    # The total memory accumulated is 200Mb here since each boolean takes 4 byte
    nalineY <- is.na(yvar)
```

```
    # The total memory here is 240Mb here for the same reason
    xvar[nalineX | nalineY] <- 0
    yvar[nalineX | nalineY] <- 0
    # The memory used here does not change since the original NA takes same memory as 0
    useline <- !(nalineX | nalineY)
    # The memory accumulated here is 280Mb
    tablex <- numeric(max(xvar) + 1)
    tabley <- numeric(max(xvar) + 1)
    # Tablex and tabley do not take much memory, can be ignored
    stopifnot(length(xvar) == length(yvar))
    res <- .C("fastcount", PACKAGE = "GCcorrect", tablex = as.integer(tablex), tabley = as.integer(tabley),
        as.integer(xvar), as.integer(yvar), as.integer(useline), as.integer(length(xvar)))
    xuse <- which(res$tablex > 0)
    xnames <- xuse - 1
    resb <- rbind(res$tablex[xuse], res$tabley[xuse])
    # Here since resb takes 120Mb memory, the total memory taken here is 400Mb
    colnames(resb) <- xnames
    return(resb)
}
# b).  I rewrite the function as follows, I removed the intermediate vars such as nlinex, nliney
# and useline, each of these takes 40Mb. Thus, I think in optimal case, the new code can release
# 120 Mb memory.
fastcount <- function(xvar, yvar) {
    xvar[is.na(xvar) | is.na(yvar)] <- 0
    yvar[is.na(xvar) | is.na(yvar)] <- 0
    tablex <- numeric(max(xvar) + 1)
    tabley <- numeric(max(xvar) + 1)
    stopifnot(length(xvar) == length(yvar))
    res <- .C("fastcount", PACKAGE = "GCcorrect", tablex = as.integer(tablex), tabley = as.integer(tabley),
        as.integer(xvar), as.integer(yvar), as.integer(!(is.na(xvar) | is.na(yvar))), as.integer(length(xvar)
    xuse <- which(res$tablex > 0)
    xnames <- xuse - 1
    resb <- rbind(res$tablex[xuse], res$tabley[xuse])
    colnames(resb) <- xnames
    return(resb)
}
```

```
# PROBLEM 5 a). The following is my running record using scf.  I tested the time spent on
# multiplying two n*n matrices using system.time then recorded information here.
# !/usr/bin/Rscript\\ system.time(matrix(rnorm(4000^2),4000)%*%matrix(rnorm(4000^2),4000))\\
# This is an information matrix when n = 4000\\
infor <- matrix(0, 8, 3)
colnames(infor) <- c("threads", "expected time", "elapsed time")
infor[, 1] <- c(1:8)
infor[, 3] <- c(32.425, 21.414, 16.242, 14.233, 13.042, 12.811, 11.73, 11.943)
infor[, 2] <- infor[1, 3]/infor[, 1]
infor
```

```
##      threads expected time elapsed time
## [1,]       1        32.425        32.42
## [2,]       2        16.212        21.41
## [3,]       3        10.808        16.24
## [4,]       4         8.106        14.23
```

```
## [5,]       5         6.485        13.04
## [6,]       6         5.404        12.81
## [7,]       7         4.632        11.73
## [8,]       8         4.053        11.94
```

```
plot(infor[, 1], infor[, 2], type = "b", col = "blue", xlab = "Number of threads", ylab = "Time(seconds)",
    main = "4000*4000 matrix multiplication")
lines(infor[, 1], infor[, 3], type = "b", col = "red", pch = 4)
# The plot is in the last 3 pages\\ !/usr/bin/Rscript\\
# system.time(matrix(rnorm(3000^2),3000)%*%matrix(rnorm(3000^2),3000))\\ This is an
# information matrix when n = 3000\\
infor1 <- matrix(0, 8, 3)
colnames(infor1) <- c("threads", "expected time", "elapsed time")
infor1[, 1] <- c(1:8)
infor1[, 3] <- c(14.716, 9.82, 7.882, 7.18, 6.487, 6.107, 6.759, 6.586)
infor1[, 2] <- infor1[1, 3]/infor1[, 1]
infor1
```

```
##      threads expected time elapsed time
## [1,]       1        14.716       14.716
## [2,]       2         7.358        9.820
## [3,]       3         4.905        7.882
## [4,]       4         3.679        7.180
## [5,]       5         2.943        6.487
## [6,]       6         2.453        6.107
## [7,]       7         2.102        6.759
## [8,]       8         1.839        6.586
```

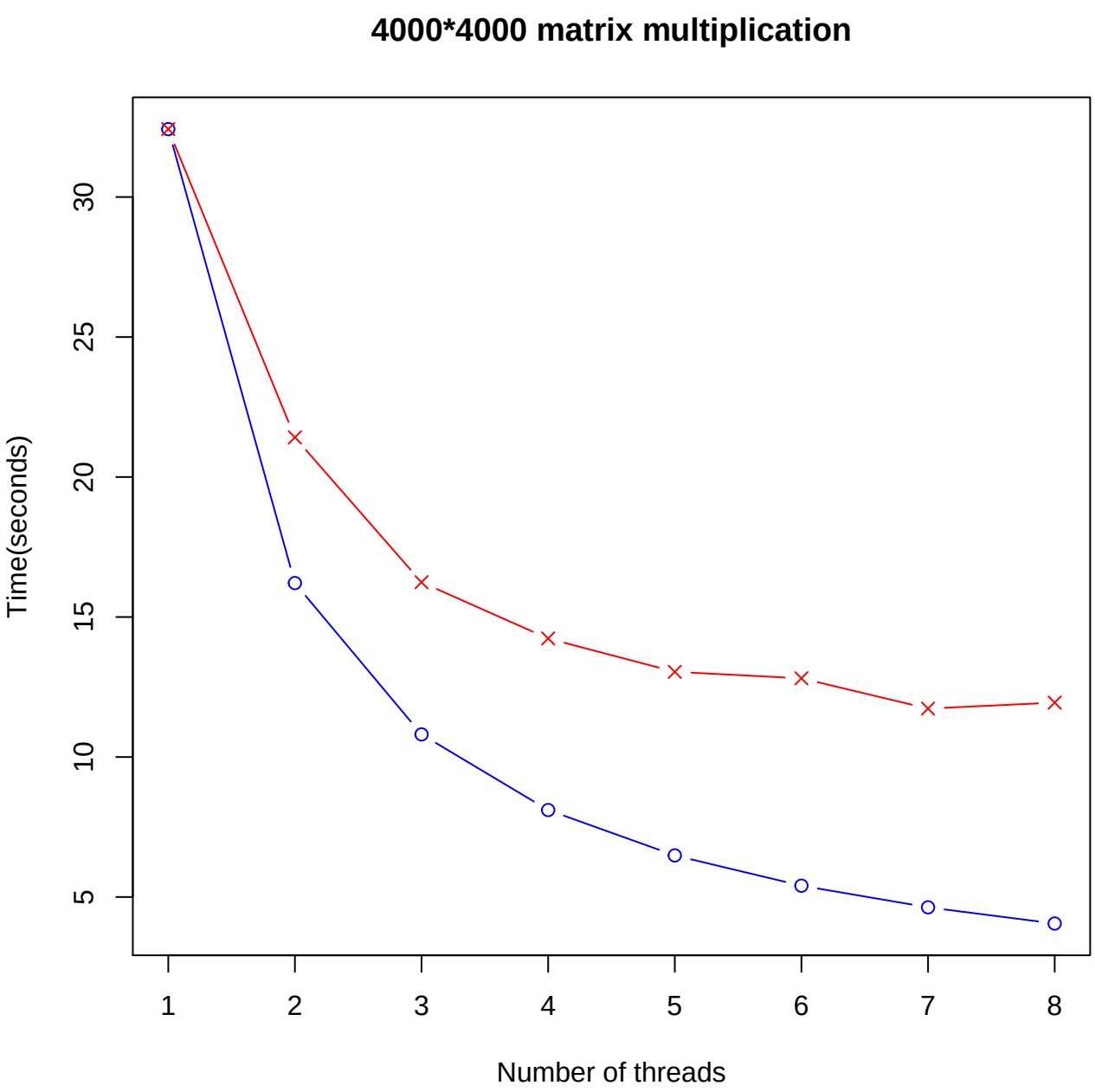
```
plot(infor1[, 1], infor1[, 2], type = "b", col = "blue", xlab = "Number of threads", ylab = "Time(seconds)",
    main = "3000*3000 matrix multiplication")
lines(infor1[, 1], infor1[, 3], type = "b", col = "red", pch = 4)
# The plot is in last 3 pages
# !/usr/bin/Rscript\\ system.time(matrix(rnorm(2000^2),2000)%*%matrix(rnorm(2000^2),2000))\\
# This is an information matrix when n = 2000\\
infor2 <- matrix(0, 8, 3)
colnames(infor2) <- c("threads", "expected time", "elapsed time")
infor2[, 1] <- c(1:8)
infor2[, 3] <- c(5.048, 3.544, 3.027, 2.785, 2.641, 2.529, 2.598, 2.636)
infor2[, 2] <- infor2[1, 3]/infor2[, 1]
infor2
```

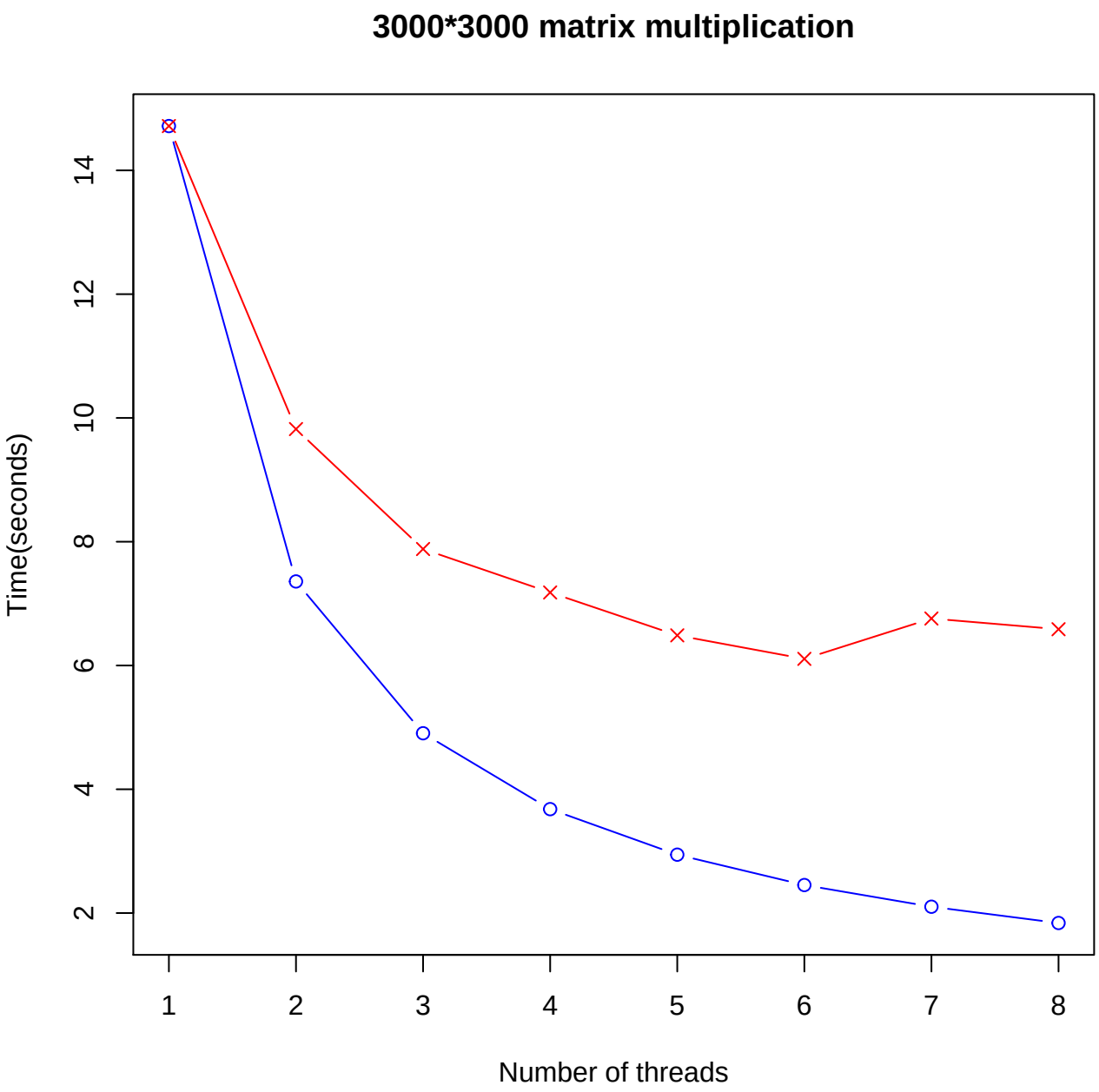
```
##      threads expected time elapsed time
## [1,]       1        5.0480        5.048
## [2,]       2        2.5240        3.544
## [3,]       3        1.6827        3.027
## [4,]       4        1.2620        2.785
## [5,]       5        1.0096        2.641
## [6,]       6        0.8413        2.529
## [7,]       7        0.7211        2.598
## [8,]       8        0.6310        2.636
```

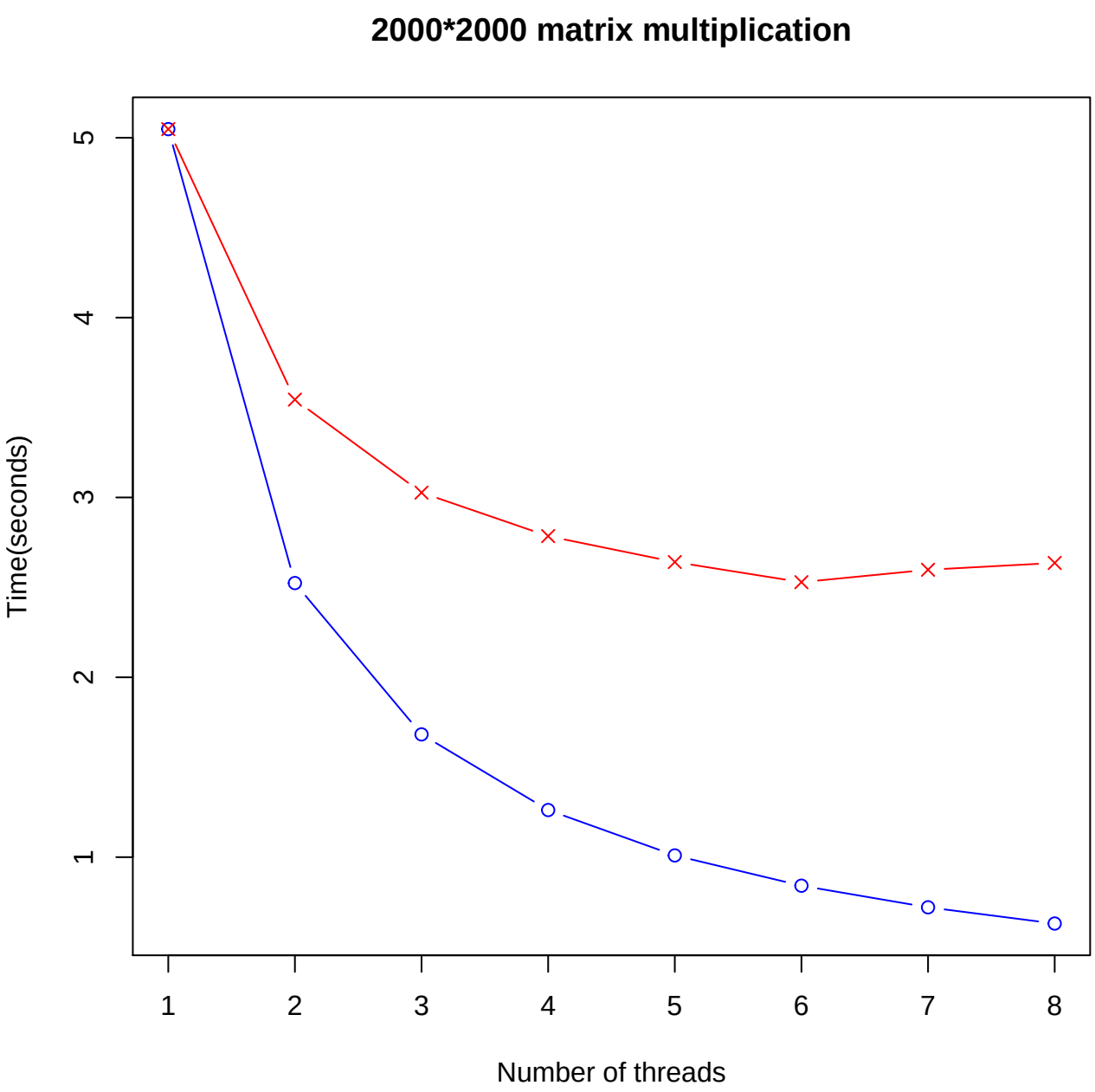
```
plot(infor2[, 1], infor2[, 2], type = "b", col = "blue", xlab = "Number of threads", ylab = "Time(seconds)",
    main = "2000*2000 matrix multiplication")
lines(infor2[, 1], infor2[, 3], type = "b", col = "red", pch = 4)
# The plot is in the last 3 pages
```

```r
# b).
require(parallel)

## Loading required package:  parallel

require(doParallel)

## Loading required package:  doParallel
## Loading required package:  foreach
## Loading required package:  iterators

library(foreach)
library(iterators)
library(rbenchmark)
nCores <- 2
registerDoParallel(nCores)
# I used 2000*2000 matrix to do this problem, and I tried 4000*4000 matrix using the same code
# on arwen.
a <- matrix(rnorm(2000^2), 2000)
b <- matrix(rnorm(2000^2), 2000)
sample <- seq(1, 2000, by = 250)
out <- foreach(i = sample, .combine = cbind) %dopar% {
    t <- a %*% b[, i:(i + 249)]
}
benchmark(foreach(i = sample, .combine = cbind) %dopar% {
    t <- a %*% b[, i:(i + 249)]
}, a %*% b, replications = 10, columns = c(1:5))

##                                                                                  test
## 2                                                                              a %*% b
## 1 foreach(i = sample, .combine = cbind) %dopar% {\n    t <- a %*% b[, i:(i + 249)]\n}
##   replications user.self sys.self elapsed
## 2           10    67.050    0.250   67.80
## 1           10     0.799    0.879   43.59

identical(a %*% b, out)

## [1] TRUE


# nCores = 1
user system elapsed
13.565 0.104 13.675
user system elapsed
16.265
# 0.280 16.548
nCores = 2
user system elapsed
13.509 0.092 13.604
user system
# elapsed
9.153 1.012 9.857
nCores = 4
user system elapsed
13.385 0.080
```

```
# 13.464
user system elapsed
22.801 1.724 5.673
nCores = 6
user system
# elapsed
13.829 0.120 13.950
user system elapsed
24.874 1.808 4.249
nCores =
# 8
user system elapsed
13.121 0.052 13.177
user system elapsed
26.861 1.952
# 3.502
```


4000*4000 matrix multiplication
Time(seconds)
5
10
15
20
25
30
1
2
3
4
5
6
7
8
Number of threads


3000*3000 matrix multiplication
Time(seconds)
2
4
6
8
10
12
14
1
2
3
4
5
6
7
8
Number of threads


2000*2000 matrix multiplication
Time(seconds)
1
2
3
4
5
1
2
3
4
5
6
7
8
Number of threads