Pairs Trading

Our Financial Data

```r
readData <-
   #
   # A function to read the data and convert the Date column to
   # an object of class Date.
   # The date values are expected to be in a column named Date.
   # We may want to relax this and allow the caller specify the
   # column - by name or index.
   function(fileName, dateFormat = c("%Y-%m-%d", "%Y/%m/%d"), ...)
   {
      data <- read.csv(fileName, header = T,
                       stringsAsFactors = F, ...)
      for(fmt in dateFormat) {
         tmp <- as.Date(data$Date, fmt)
         if(all(!is.na(tmp))) {
            data$Date <- tmp
            break
         }
      }

      data[ ordered(data$Date), ]
   }
```

```r
getSymbols("T", src = "yahoo", from = "1985-01-01", to = "2015-12-31")
```

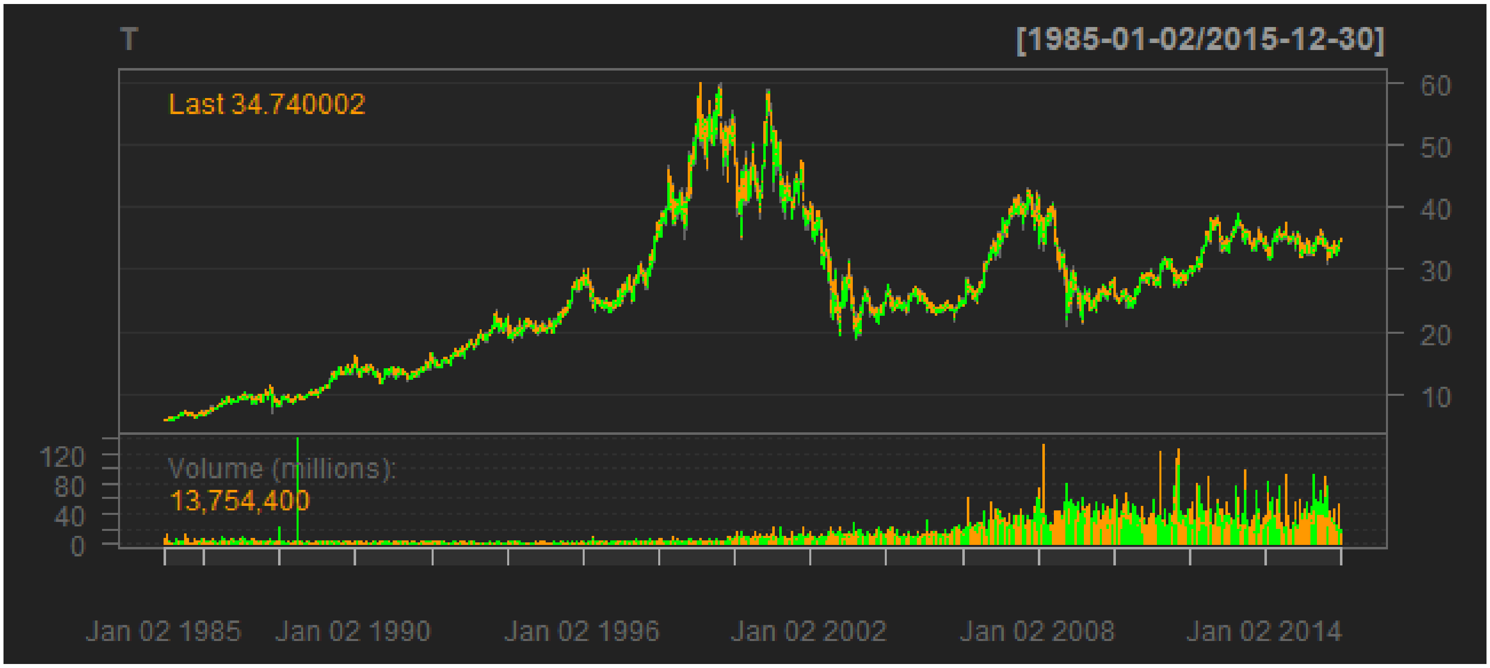
```
'getSymbols' currently uses auto.assign=TRUE by default, but will
use auto.assign=FALSE in 0.5-0. You will still be able to use
'loadSymbols' to automatically load data. getOption("getSymbols.env")
and getOption("getSymbols.auto.assign") will still be checked for
alternate defaults.

This message is shown once per session and may be disabled by setting
options("getSymbols.warning4.0"=FALSE). See ?getSymbols for details.
```

```
[1] "T"
```

```r
getSymbols("VZ", src = "yahoo", from = "1985-01-01", to = "2015-12-31")
```

```
[1] "VZ"
```

```r
chartSeries(T)
```




```r
chartSeries(VZ)
```

```
ATT_adj = T$T.Adjusted; VER_adj <- VZ$VZ.Adjusted
```

```
combineStocks <-
   function(lseries, rseries,
            stockNames = c(deparse(substitute((a)),
                           deparse(substitute(b)))),
            add_ratio = T) {
   l_adj <- lseries[, 6]; r_adj <- rseries[, 6]
   combined <- merge(l_adj, r_adj)

   df_result <- structure(data.table(Date = index(combined), combined),
             names = c("Date", stockNames))

   if(add_ratio) {

      Ratio <- combined[, 1] / combined[, 2]
      df_result <- cbind(df_result, Ratio)
      colnames(df_result)[4] <- "Ratio"
   }

   df_result
}
```

```
overlap <- combineStocks(T, VZ, c("ATT", "VER"))
```

```
Warning in if (add_ratio) {: the condition has length > 1 and only the first
element will be used
```

```
names(overlap)
```

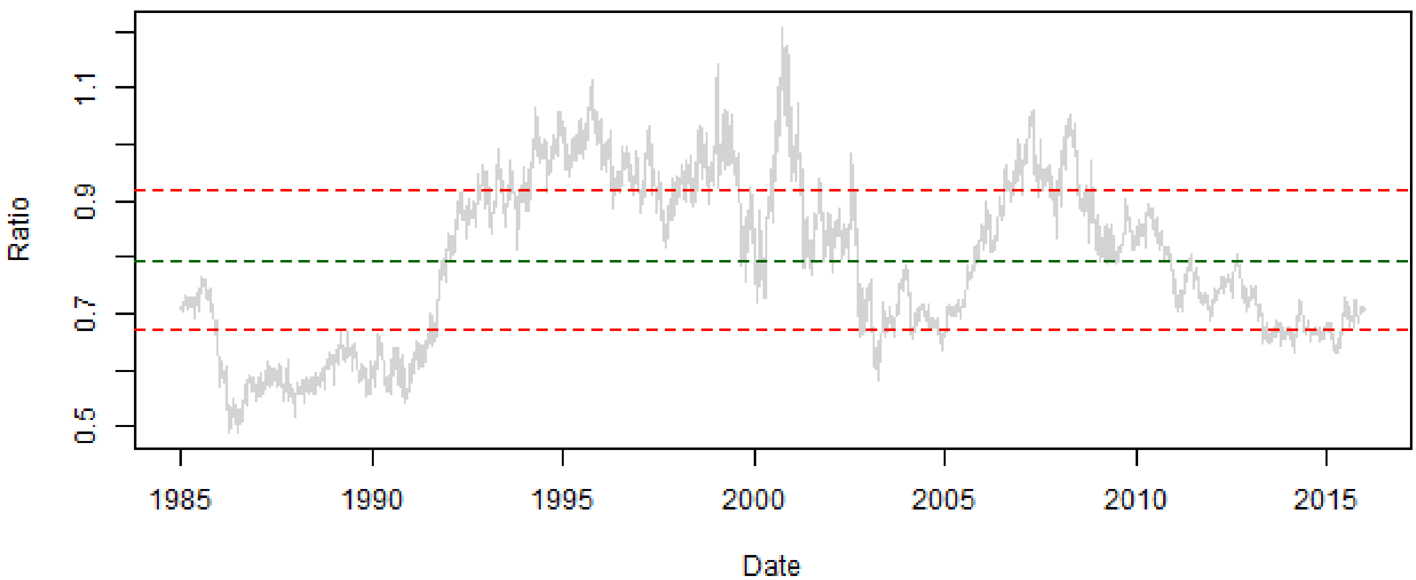
```
[1] "Date"  "ATT"   "VER"   "Ratio"
```

```
range(overlap$Date)
```

```
[1] "1985-01-02" "2015-12-30"
```

```
plotRatio <-
   function(r, k = 1, date = seq(along = r), ...)
   {
      plot(date, r, type = "l", ...)

      mu <- mean(r); kval <- k * sd(r)
      upper <- mu + kval; lower <- mu - kval

      abline(h = c(mu,
                   upper,
                   lower),
```

```
            col = c("darkgreen", rep("red", 2 * length(k))),
            lty = "dashed")
      text(1, upper, upper)
   }


plotRatio(overlap$Ratio, k = .85,
          overlap$Date, col = "lightgray",
          xlab = "Date", ylab = "Ratio")
```




```
findNextPosition <-
   # e.g., findNextPosition(r)
   #      findNextPosition(r, 1774)
   # Check they are increasing and correctly offset
   function(ratio, startDay = 1, k = 1,
            m = mean(ratio), s = sd(ratio))
   {
      up = m + k * s
      down = m - k * s

      if(startDay > 1)
           ratio = ratio[ - (1:(startDay-1)) ]

     isExtreme = ratio >= up | ratio <= down
```

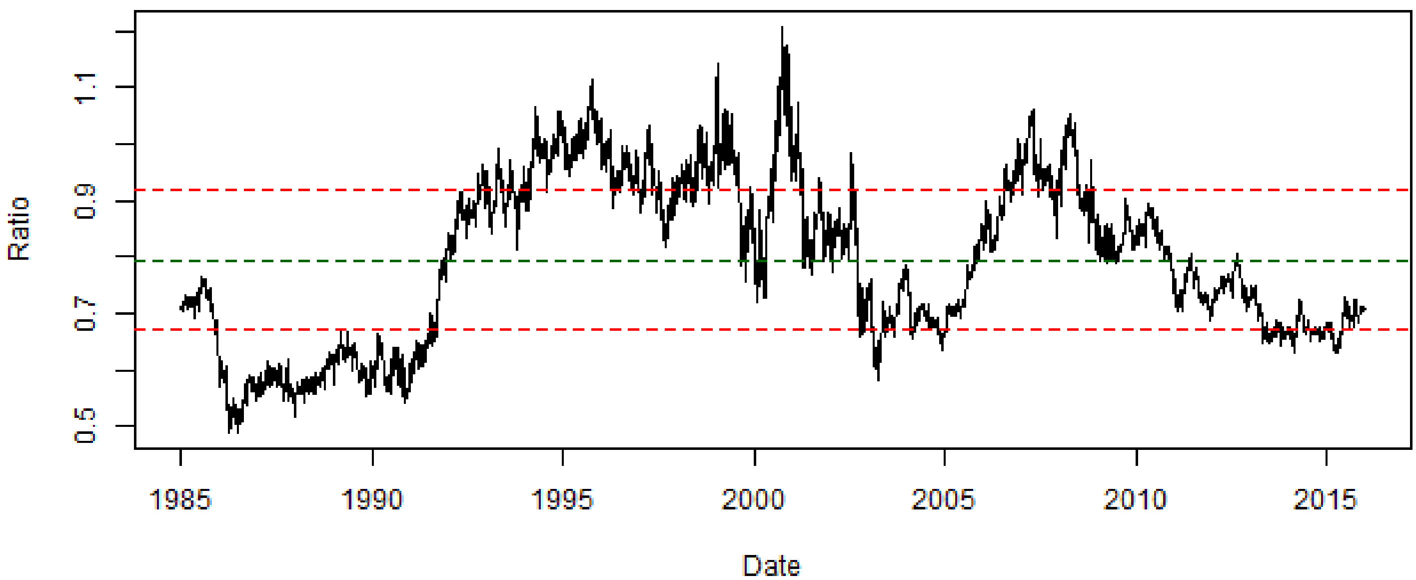
```
    if( is.na(isExtreme) || !any(isExtreme))
        return(integer())

     start = which(isExtreme)[1]
     backToNormal <- if(ratio[start] > up)
        ratio[ - (1:start) ] <= m
     else
        ratio[ - (1:start) ] >= m

     # return either the end of the position or the index
     # of the end of the vector
     # could return NA for not ended, i.e,, which(backToNormal)[1]
     # for both cases. But then the caller has to interpret that.

     end <- if(any(backToNormal))
              which(backToNormal)[1] + start
            else
               length(ratio)

     c(start, end) + startDay + 1
  }


r <- overlap$Ratio; k <- .85

a <- findNextPosition(r, k = k)
b <- findNextPosition(r, a[2], k = k)
c <- findNextPosition(r, b[2], k = k)
```

Displaying the Positions

```
showPosition <-
   function(days, ratios, radius = 100) {
      symbols(days, ratios, circles = rep(radius, 2),
              fg = c("darkgreen", "red"), add = T, inches = F)
   }

showPosition <-
   function(pos, col = c("darkgreen", "red"), ...)
   {
      if(!is.list(pos))
         return(invisible(lapply(pos, showPosition, col = col, ...)))
```

```
      abline(v = pos, col = col, ...)
   }
```

```
plotRatio(r, k, overlap$Date, xlab = "Date", ylab = "Ratio")
```




```
#showPosition(overlap$Date[a], r[a])
#showPosition(overlap$Date[b], r[b])
#showPosition(overlap$Date[c], r[c])
```

Finding all Positions

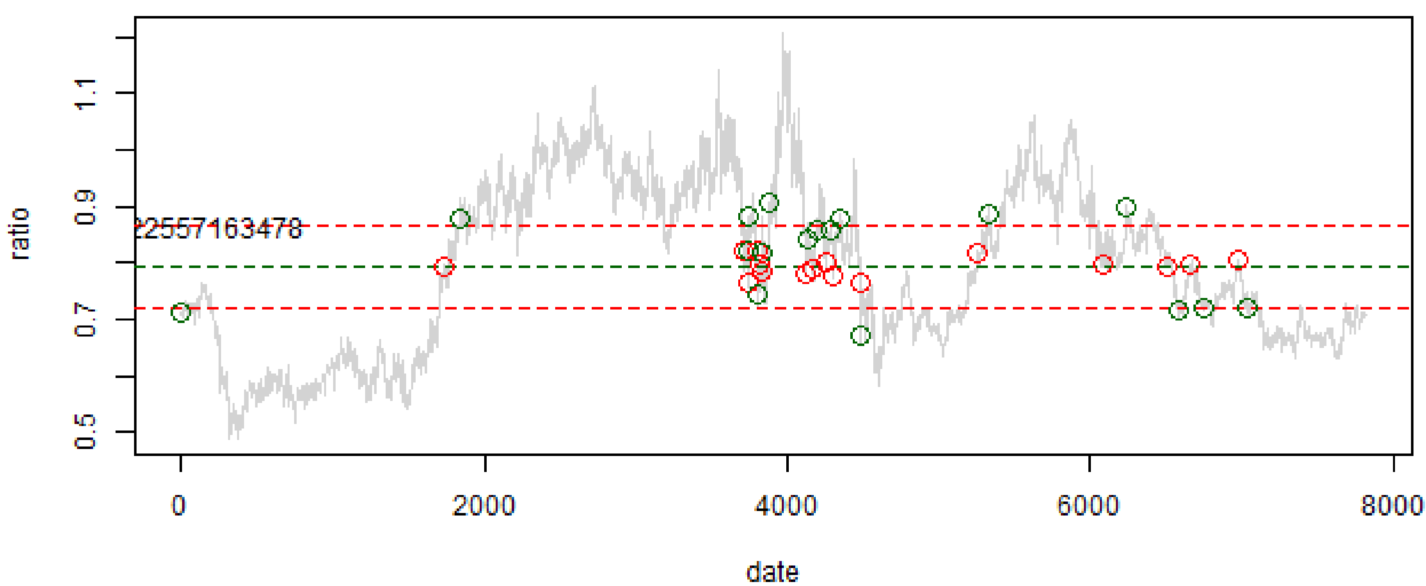
```
getPositions <-
   function(ratio, k = 1, m = mean(ratio), s = sd(ratio))
   {
      when = list()
      cur = 1

      while(cur < length(ratio)) {
         tmp <- findNextPosition(ratio, cur, k, m, s)
         if(length(tmp) == 0)
            break

         when[[length(when) + 1]] <- tmp
         if(is.na(tmp[2] || temp[2] == length(ratio)))
```

```
            break
         cur = tmp[2]
      }

      when
   }
```

```
showPosition <-
   function(days, ratio, radius = 70)
   {
      if(is.list(days))
         days <- unlist(days)

      symbols(days, ratio[days],
              circles = rep(radius, length(days)),
              fg = c("darkgreen", "red"),
              add = T, inches = F)
   }
```

```
k <- .5
pos <- getPositions(r, k)
plotRatio(r, k, col = "lightgray", ylab = "ratio")
showPosition(pos, r)
```

```
Warning in if (!add) {: the condition has length > 1 and only the first element
will be used
```



Computing Profit

```
positionProfit <-
   # r = overlap$att / overlap$verizon
   # k = 1.7
   # pos = getPositions(r, k)
   # positionProfile(pos[[1]], overlap$att, overlap$verizon)
   function(pos, stockPriceA, stockPriceB,
            ratioMean = mean(stockPriceA / stockPriceB ),
            p = 0.001, byStock = F)
   {
      if(is.list(pos)) {
         ans = sapply(pos, positionProfit,
                      stockPriceA, stockPriceB, ratioMean, p, byStock)

         if(byStock)
            rownames(ans) <- c("A", "B", "commission")

         return(ans)
      }

      # prices at the start and end of the positions
      priceA <- stockPriceA[pos]
      priceB <- stockPriceB[pos]

      # how many units can we by of A and B with $1?

      unitsOfA <- 1/priceA[1]
      unitsOfB <- 1/priceB[1]

      # The dollar amount of how many units we would buy of A and B
      # at the cost at the end of the position of each
      amt <- c(unitsOfA * priceA[2], unitsOfB * priceB[2])

      if(is.na(priceA[1]) | is.na(priceB[1]))
         return(0)

      # which are we selling
      sellWhat <- if(priceA[1] / priceB[1] > ratioMean) "A" else "B"

      profit <- if(sellWhat == "A")
                  c( ( 1 - amt[1]), (amt[2] - 1), - p * sum(amt))
               else
                  c( (1 - amt[2]), (amt[1] - 1), - p * sum(amt))
```

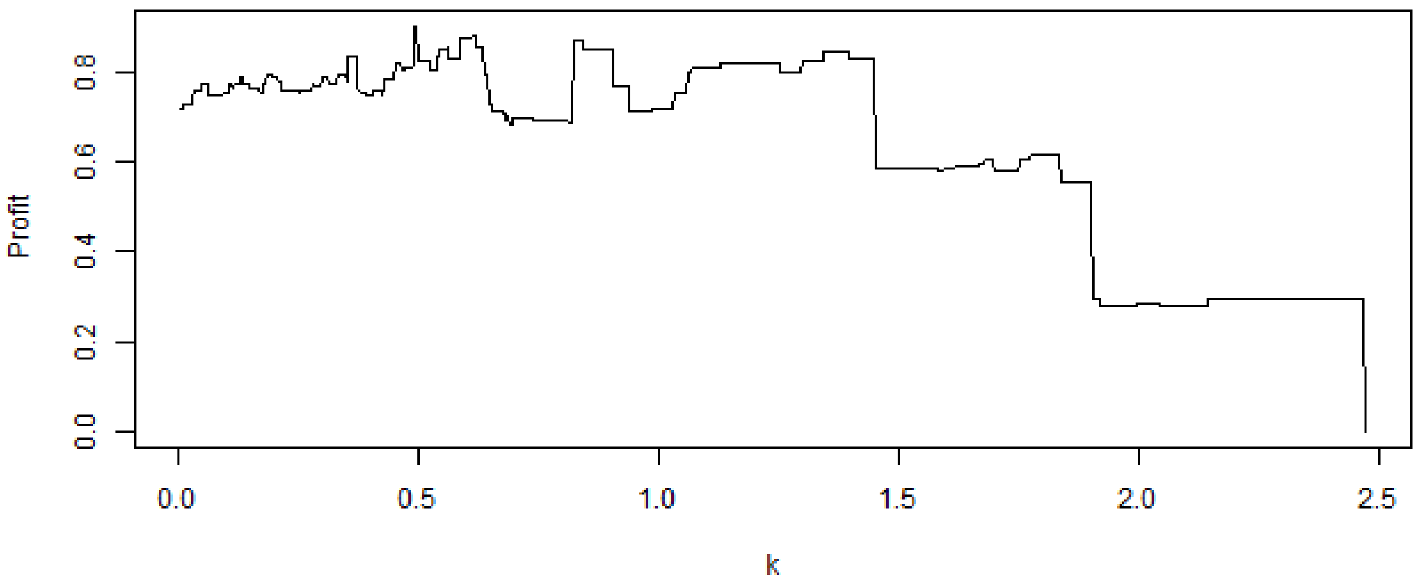
```
      if( byStock )
         profit
      else
         sum(profit)
   }
```

```
pf <- positionProfit(c(1, 2), c(3838.48, 8712.87),
                              c(459.11, 1100.65), p = 0)

prof <- positionProfit(pos, overlap$ATT, overlap$VER, mean(r))
```

Finding Optimal K

```
i <- 1:floor(nrow(overlap)/2)
train <- overlap[i, ]
test <- overlap[ -i, ]

r.train <- train$Ratio
r.test <- test$Ratio

period <- seq(min(overlap$Date), by = "5 years", length = 2)
period.train <- paste(period[1], period[2], sep="/")

att.train <- T[period.train]$T.Adjusted
verizon.train <- VZ[period.train]$VZ.Adjusted
r.train <- att.train/verizon.train

period.test <- paste(period[2], max(overlap$Date), sep="/")
att.test <- T[period.test]$T.Adjusted
verizon.test <- VZ[period.test]$VZ.Adjusted
r.test <- att.test/verizon.test

k.max <- max((r.train - mean(r.train)) / sd(r.train))
k.min <- min((abs(r.train - mean(r.train)) / sd(r.train)))

ks <- seq(k.min, k.max, length = 1000)
m <- mean(r.train)

profits <-
   sapply(ks,
          function(k) {
             pos <- getPositions(r.train, k)
             sum(positionProfit(pos, train$ATT, train$VER,
                                mean(r.train)))
```

```
        })

plot(ks, profits, type = "l", xlab = "k", ylab = "Profit")
```




```
max_profits <- which.max(profits)
optimal <- ks[ max_profits ]
```

```
tmp.k <- ks[ profits == max(profits) ]
pos <- getPositions(r.train, tmp.k[1])
all(sapply(tmp.k[-1],
           function(k)
              identical(pos, getPositions(r.train, k))))
```

```
[1] TRUE
```

```
k.star <- mean( ks[ profits == max(profits)])

pos <- getPositions(r.test, k.star, mean(r.train), sd(r.train))
testProfit <- sum(positionProfit(pos, test$ATT, test$VER), na.rm=T)

testProfit * 100
```

```
[1] 58.62188
```

Simulation

Vector Auto-regression

$$X_t^{(1)} = \rho X_{t-1}^{(1)} + \psi(1-\rho)X_{t-1}^{(2)} + \epsilon_t^{(1)}$$

$$X_t^{(2)} = \rho X_{t-1}^{(2)} + \psi(1-\rho)X_{t-1}^{(1)} + \epsilon_t^{(2)}$$

$$\epsilon_t^{(1)} \sim N(0, \sigma_i^2)$$

…

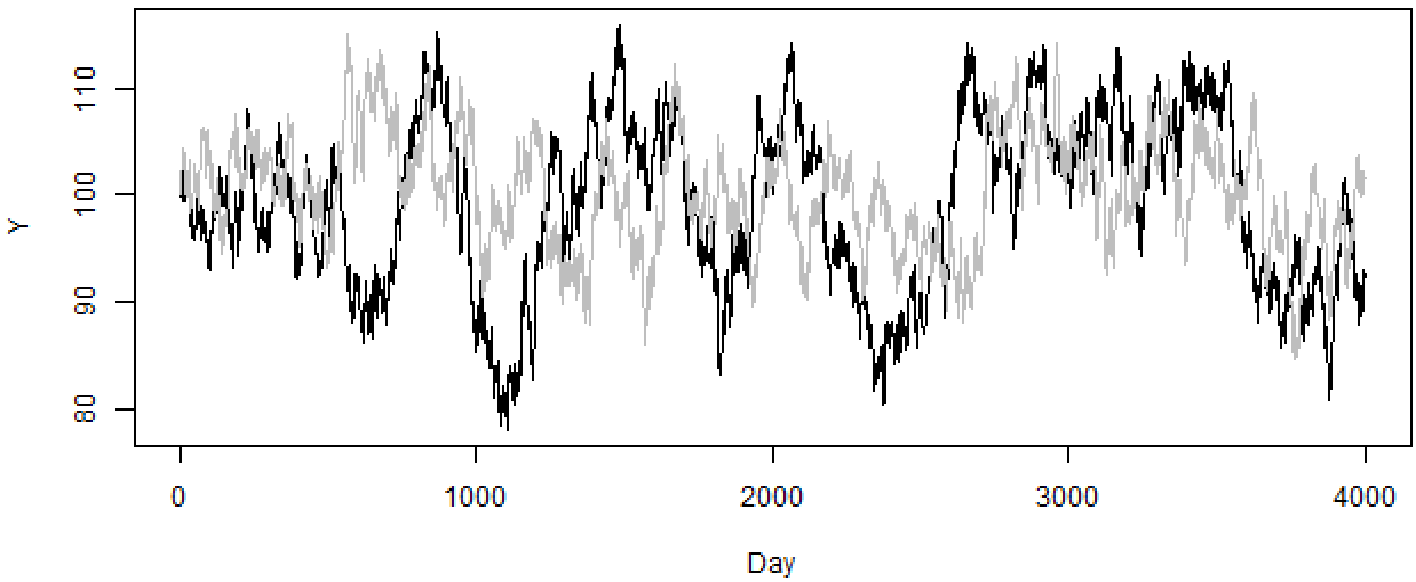
$$Y_t^{(1)} = \beta_0^{(1)} + \beta_1^{(1)} t + X_t^1$$

$$Y_t^{(2)} = \beta_0^{(2)} + \beta_1^{(2)} t + X_t^2$$

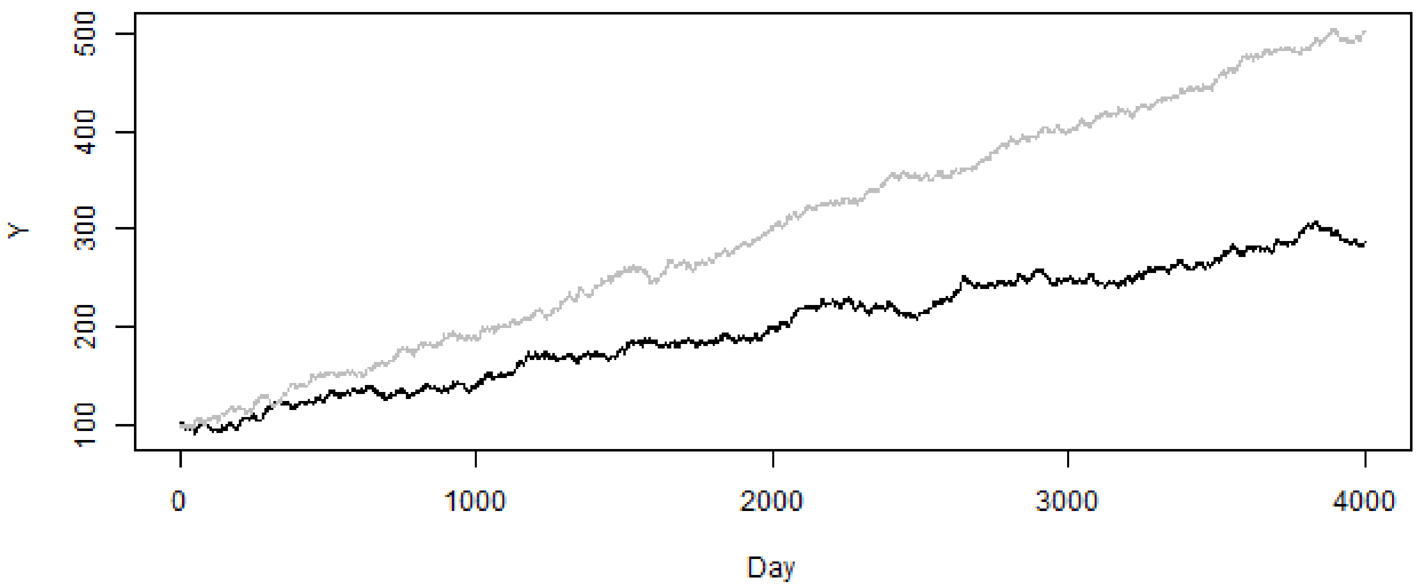
Simulating the Stock Price Series

```
stockSim <-
   function(n = 4000, rho = 0.99, psi = 0, sigma = rep(1, 2),
            beta0 = rep(100, 2), beta1 = rep(0, 2),
            epsilon = matrix(rnorm(2*n, sd = sigma),
                             nrow = n, byrow = T))
   {
      X <- matrix(0, nrow = n, ncol = 2)
      X[1, ] <- epsilon[1, ]

      A <- matrix(c(rho, psi * (1-rho), psi*(1-rho), rho), nrow = 2)
      for(i in 2:n)
         X[i, ] = A %*% X[i-1, ] + epsilon[i, ]

      X[, 1] <- beta0[1] + beta1[1] * (1:n) + X[, 1]
      X[, 2] <- beta0[2] + beta1[2] * (1:n) + X[, 2]

      X
   }
```

```
a <- stockSim(rho = .99, psi = 0)

matplot(1:nrow(a), a, type = "l", xlab = "Day", ylab = "Y",
        col = c("black", "grey"), lty = "solid")
```




```r
beta1 <- c(.05, .1)

a <- stockSim(beta1 = c(.05, .1))

matplot(1:nrow(a), a, type = "l", xlab = "Day", ylab = "Y",
        col = c("black", "grey"), lty = "solid")
```



Simulation Utilities

```
runSim <-
function(rho = .99, psi = .9, beta0 = c(100, 100), beta1 = c(0, 0),
         sigma = c(1, 1), n = 4000)
{
    X = stockSim(n, rho, psi, sigma, beta = beta0, beta1 = beta1)
    train = X[ 1:floor(n/2), ]
    test = X[ (floor(n/2)+1):n, ]
    m = mean(train[, 1]/train[, 2])
    s = sd(train[, 1]/train[, 2])
    k.star = getBestK(train[, 1], train[, 2], m = m, s = s)
    getProfit.K(k.star, test[, 1], test[, 2], m, s)
}

getProfit.K =
function(k, x, y, m = mean(x/y), s = sd(x/y))
{
    pos = getPositions(x/y, k, m = m, s = s)
    if(length(pos) == 0)
       0
    else
       sum(positionProfit(pos, x, y, m), na.rm=T)
}
```

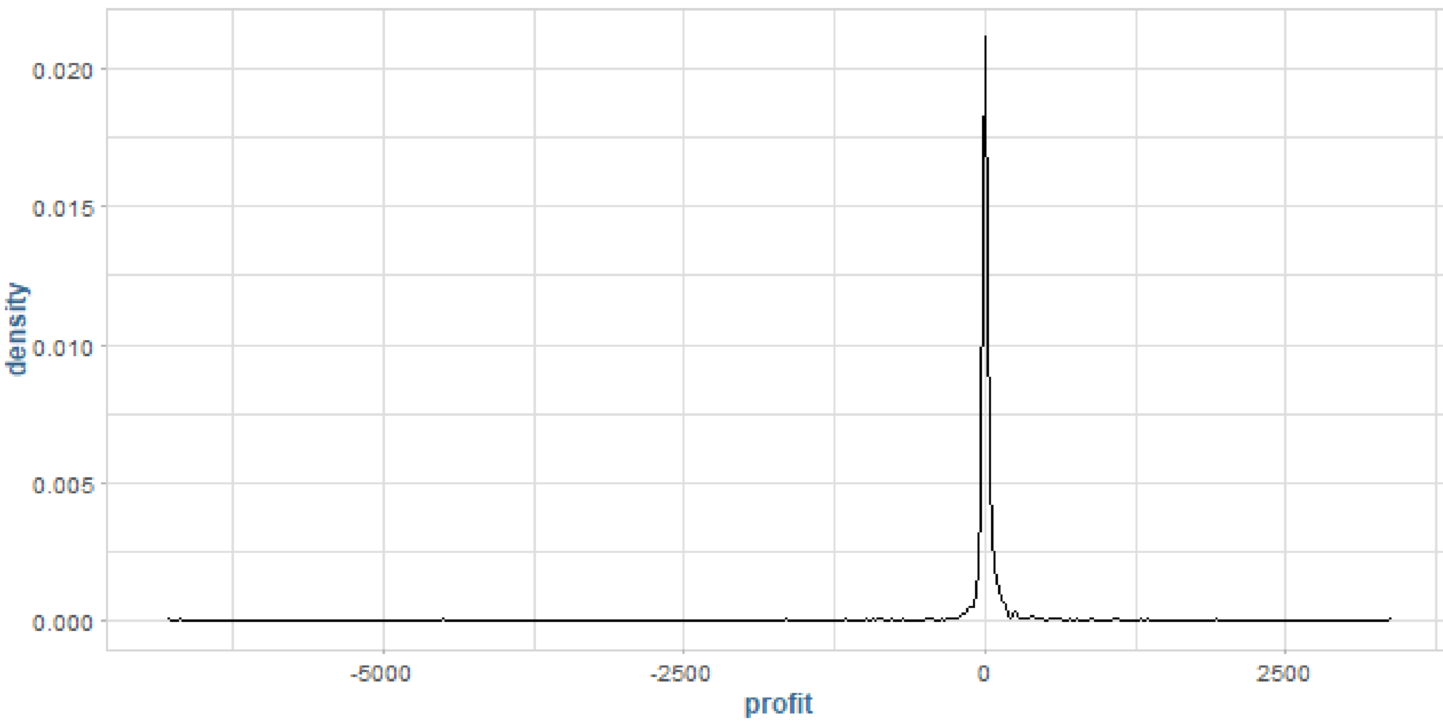
```r
getBestK =
function(x, y, ks = seq(0.1, max.k, length = N), N = 100,
         max.k = NA, m = mean(x/y), s = sd(x/y))
{
    if(is.na(max.k)) {
       r = x/y
       max.k = max(r/sd(r))
    }

    pr.k = sapply(ks, getProfit.K, x, y, m = m, s = s)
    median(ks[ pr.k == max(pr.k) ])
}

simProfitDist =
function(..., B = 999)
      sapply(1:B,  function(i, ...) runSim(...), ...)
```

Run the Simulation

```r
system.time({ x = simProfitDist( .99, .9, c(0, 0)) })
```

```
   user  system elapsed
  44.25    0.11   44.42
```

```r
ggplot(data.table(profit = x), aes(profit)) +
   geom_density()
```




```r
g <- expand.grid(psi = seq(.8, .99, length.out = 20),
                 beta1 = seq(-0.01, .01, length.out = 20),
                 beta2 = seq(-0.1, 0.01, length.out = 20))

dim(g)
```

```
[1] 8000    3
```

```r
Rprof("sim.prof")

system.time({x = simProfitDist( .99, .9, c(0, 0))})
```

```
   user  system elapsed
  44.08    0.04   44.16
```

```r
Rprof(NULL)

head(summaryRprof("sim.prof")$by.self)
```

```
                    self.time self.pct total.time total.pct
"getPositions"          19.08    66.81      21.34     74.72
"runSim"                 2.42     8.47      28.50     99.79
"findNextPosition"       2.26     7.91       2.26      7.91
"lapply"                 1.66     5.81      28.50     99.79
"FUN"                    1.32     4.62      28.50     99.79
"sapply"                 0.56     1.96      28.50     99.79
```

```
counter <- 0L
trace(findNextPosition, quote( counter <<- counter + 1L),
      print = FALSE)
```

```
[1] "findNextPosition"
```

```
system.time({x = simProfitDist( .99, .9, c(0, 0))})
```

```
   user  system elapsed
  45.95    0.08   46.03
```

```
counter
```

```
[1] 620789
```

```
untrace(findNextPosition)
```

```
library(compiler)
stockSim.cmp <- cmpfun(stockSim)
```

```
tm.orig <- system.time({replicate(80, stockSim())})
tm.compiled <- system.time({replicate(80, stockSim.cmp())})

tm.orig/tm.compiled
```

```
   user  system elapsed
1.02439     NaN 1.02439
```

```
c.lib <- paste(here(), "Case Studies", "06_stockSim.dll", sep="/")

dyn.load(c.lib)
```

```
stockSim.c <-
   function(n = 4000, rho = 0.99, psi = 0, sigma = rep(1, 2),
            beta0 = rep(100, 2), beta1 = rep(0, 2),
            epsilon = matrix(rnorm(2*n, sd = sigma), nrow = n))
   {
      X <- matrix(0, nrow = n, ncol = 2)
      X[1, ] <- epsilon[1, ]

      X <- .C("stockSim", X, as.integer(n), rho, psi, epsilon)[[1]]

      X[, 1] <- beta0[1] + beta1[1] + (1:n) + X[, 1]
      X[, 2] <- beta0[2] + beta1[2] + (1:n) + X[, 2]

      X
   }
```

```r
e <- matrix(rnorm(2*4000, sd = c(1, 1)), , 2)
tmp1 <- stockSim.c(epsilon = e)
tmp2 <- stockSim(epsilon = e)
```

```r
stockSim <- stockSim.c

Rprof("sim.prof")
system.time({x = simProfitDist(.99, .9, c(0, 0))})
```

```
   user  system elapsed
  17.00    0.06   17.13
```

```r
Rprof(NULL)
head(summaryRprof("sim.prof")$by.self)
```

```
                   self.time self.pct total.time total.pct
"getPositions"          7.10    63.96       8.26     74.41
"findNextPosition"      1.16    10.45       1.16     10.45
"FUN"                   0.66     5.95      11.06     99.64
"lapply"                0.62     5.59      11.06     99.64
"sapply"                0.40     3.60      11.06     99.64
"simplify2array"        0.40     3.60       0.56      5.05
```