Exercises Data Science with R

Thomas Kirschstein

Chapter 6

1. Try to rearrange the data objects `a.vec`, `x.vec`, and `y.vec` such that apply can be used to calculate all distances.

```
weig.mh.dist <- function(x) x[1] * (abs(x[2] - x[4]) + abs(x[3] - x[5]) )
n <- 10                              # set a number of points/customers
a.vec <- sample(1:100, size = n)     # sample weights for each point/customer
x.vec <- rnorm(n)                    # sample x coordinates
y.vec <- rnorm(n)                    # sample y coordinates
df <- data.frame(a = a.vec, x = x.vec, y = y.vec, u = 1, v = 1)
apply(df, 1, weig.mh.dist)
##  [1]   8.888207 145.675854 111.592796  17.537813  39.133444 115.587124
##  [7]  33.431153  14.242379  13.543013 119.989210
```

2. Try to fasten the code by initializing all data objects in the necessary size in advance.

```
# old time: 12.58 sec.
n <- 10^6                # number of trials
u <- 0; v <- 0           # current location
res.vec <- numeric(n)    # result vector

system.time(for(i in 1:n){
  a <- sample(1:100, size = 1, replace =T)     # sample weight for current point/customer
  x <- rnorm(1)                        # sample x coordinate
  y <- rnorm(1)                        # sample y coordinate
  res.vec[i] <- a * (abs(x - u) + abs(y - v) )     # save Manhattan distance
})
##    user  system elapsed
##    9.35    0.04    9.38
```

Note that the computation times depend on your PC.

3. Can you simplify and fasten the code by sampling only even numbers?

```
# old time: 11.85 sec.
n <- 10^6                  # number of trials
system.time({
a.vec <- sample(seq(2, 100, by = 2), size = n, replace =T)
x.vec <- rnorm(n)                    # sample x coordinate
y.vec <- rnorm(n)                    # sample y coordinate
```

```r
res.vec <- a * (abs(x) + abs(y) )
})
##    user  system elapsed
##    0.23    0.00    0.24
```

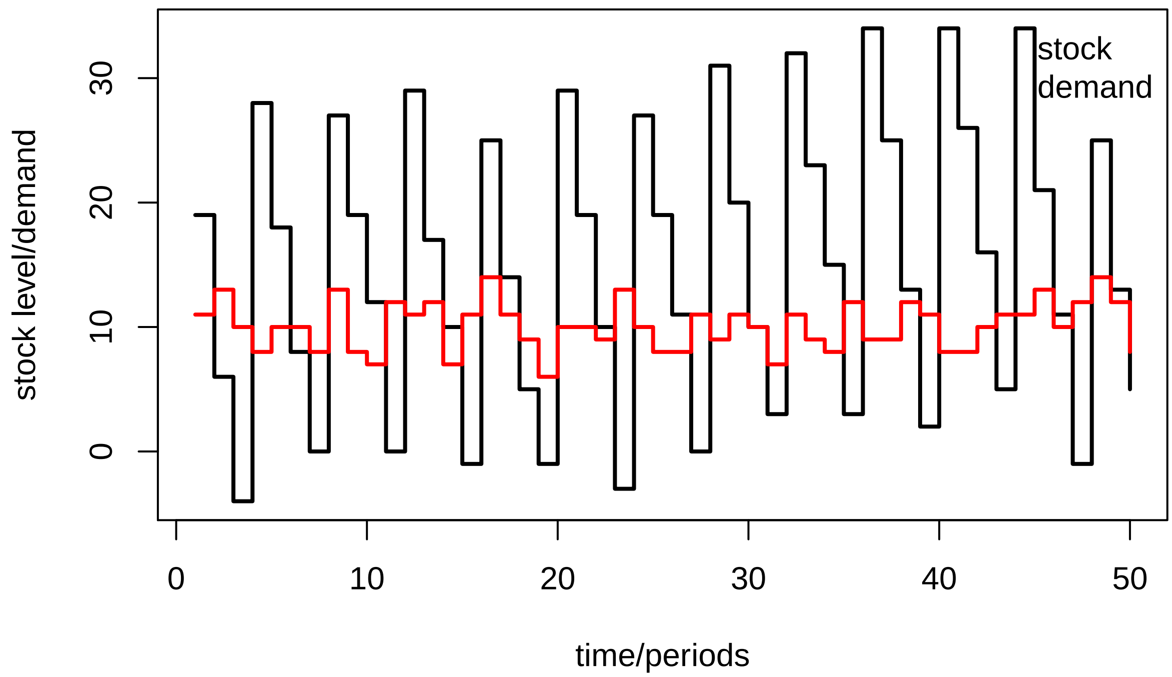
4. Can you reformulate the stopping criteria such that only one break-statement is necessary?

```r
# old time: 12.27 sec.
n <- 10^6          # number of trials
n.excess <- 10^6 * 1.5    # but no more then n.excess loops
u <- 0; v <- 0      # current location
i <- 1              # trial index
j <- 0              # iteration counter
res.vec <- numeric(n)   # result vector

system.time(repeat{
    j <- j + 1
    if(i > n | j > n.excess) break  # joint break statement
    a <- sample(1:100, size = 1, replace =T)    # sample weight
    if(a %% 2 == 1) next            # skip iteration
    x <- rnorm(1)                   # sample x coordinate
    y <- rnorm(1)                   # sample y coordinate
    res.vec[i] <- a * (abs(x - u) + abs(y - v) )
    i <- i+1
  })
##    user  system elapsed
##   11.29    0.01   11.33
```

5. Formulate a loop that calculates the inventory records over n periods based on an initial stock level (say $i_0 = 20$) where every 4 periods 40 units arrive at the inventory. Sample the demand for each period from a normal distribution with $\mathcal{D} \sim N(10, 2)$ and round to integers.

```r
n <- 50
i.vec <- numeric(n)
d.vec <- round(rnorm(n, mean = 10, sd = 2))
i.vec[1] <- 30 - d.vec[1]
for(i in 2:n){
  if(i  %% 4 == 0){
    i.vec[i] <- i.vec[i-1] - d.vec[i] + 40
  }
  else{
    i.vec[i] <- i.vec[i-1] - d.vec[i]
  }
}
plot(1:n, i.vec, xlab="time/periods", ylab="stock level/demand", type="s", lwd=2)
lines(1:n, d.vec, type="s", col="red", lwd=2)
legend("topright", col=c("black","red"), legend = c("stock", "demand"), bty = "n")
```



6. Consider a dynamic lot sizing problem with ordering cost of $c_o = 100$ and a holding cost rate $c_h = 0.1$ \$ per period and unit. The demand over 10 periods is sampled from a Possion distribution with $\lambda = 10$ (use `rpois()`). Calculate the total cost matrix with R.

```
n <- 10
d.vec <- rpois(n, lambda = 10)
c.mat <- matrix(NA, ncol = n, nrow = n)
c.h <- 0.5
c.o <- 100
for(i in 1:n){
  c.mat[i, i:n] <- cumsum(0:(n-i) * d.vec[i:n]) * c.h + c.o
}
```

7. Formulate a function that performs 1st-order exponential smoothing: $p_{t+1} = (1 - \alpha) \cdot p_t + \alpha \cdot x_t$. Is there also an builtin function? If so, compare run times.

```
first.exsm <- function(alpha, d, p.ini){
  n <-length(d)
  p <- numeric(n+1)
  p[1] <- p.ini
  for(i in 1:n){
    p[i+1] <- (1-alpha) * p[i] + alpha * d[i]
  }
  return(p)
}
```

```r
n <- 1e+6
d.vec <- rnorm(n, 10, 2)
system.time(p.vec <- first.exsm(alpha = 0.4, d = d.vec, p.ini = 10))
##    user  system elapsed
##    0.17    0.00    0.17
# Built-in function: HoltWinters() for 1st-3rd order ES
system.time(p.bi <- HoltWinters(d.vec, alpha = 0.4, beta = F, gamma = F, l.start = 10))
##    user  system elapsed
##    0.09    0.02    0.11
# Alt. built-in function: ses() for 1st order ES
library(forecast)
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
system.time(p.bi2 <- ses(d.vec, alpha = 0.4, h=1, initial = "simple"))
##    user  system elapsed
##   14.20    0.00   14.21
```

Note that the forecasts are different as the initialization procedures of `HoltWinters()` and `ses()` are different.