Lecture 2 Exercises

Isabel Fulcher

8/6/2018

```
library(dplyr)
library(matrixStats)
```

Part 1: Matrix / Vector Operations

Before starting generate the following in R:

```
set.seed(11)
b <- sample(10,4,replace=TRUE)
A <- matrix(sample(10,16,replace=TRUE),4,4)
```

Now perform the following exercises to get familiar with matrix operations:

1. What is the dimension of **A**?

```
dim(A)
```

```
## [1] 4 4
```

2. Find the transpose of **b**

```
t(b)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    3    1    6    1
```

3. Are vectors in R column or row vectors by default?
4. Calculate the following in R: $\mathbf{A}^T\mathbf{A}$

```
t(A)%*%A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]  111   46  126   62
## [2,]   46  114  154   82
## [3,]  126  154  281  132
## [4,]   62   82  132   70
```

5. Repeat the previous exercise with a built-in R function

```
crossprod(A)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]  111   46  126   62
## [2,]   46  114  154   82
## [3,]  126  154  281  132
## [4,]   62   82  132   70
```

6. Solve $\mathbf{b} = \mathbf{A}\mathbf{x}$ for $\mathbf{x}$

```
solve(A)%*%b
```

```
##             [,1]
## [1,] -0.7329843
## [2,] -0.8403141
## [3,]  0.9738220
## [4,]  0.3115183
```

7. Repeat the previous with a built-in R function.

```r
solve(A,b)
```

```
## [1] -0.7329843 -0.8403141  0.9738220  0.3115183
```

8. Create a $4x4$ matrix with diagonal elements equal to 5 and off-diagonals equal to 0.

```r
5*diag(4)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    5    0    0    0
## [2,]    0    5    0    0
## [3,]    0    0    5    0
## [4,]    0    0    0    5
```

Part II: Flow control

For Loops

1. Using a for loop, generate a vector containing the natural numbers up to 10 and add 2 to each element. How can you do this without a loop?

```r
# Option / Interpretation 1: (thanks Patrick!)
nat_numbers <- c()
for (i in 1:10) {
  nat_numbers <- c(nat_numbers, sample(10, 1) + 2)
}
print(nat_numbers)
```

```
##  [1] 5 9 6 6 3 7 6 3 4 6
```

```r
# Option / Interpretation 2:
nat <- integer(10)
for (i in 1:10){
  nat[i] <- i + 2
}

# Without a loop!
1:10 + 2
```

```
##  [1]  3  4  5  6  7  8  9 10 11 12
```

2. Load in the iris dataset using the R code provided below. Use a for loop to compute the standard deviation of the first four columns measuring sepal length, sepal width, pedal length, and pedal width respectively.

```r
# load dataset
data(iris)
# view the first ten observations
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
# solution
result <- integer(4)
for (i in 1:4){

  result[i] <- sd(iris[,i])

}
result
```

```
## [1] 0.8280661 0.4358663 1.7652982 0.7622377
```

Apply functions

1. For question 2 in the "For Loops" section, use `apply` to perform the same task.

```
# Option 1 (thanks Jemar!)
apply(iris[1:4],2,sd)
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##    0.8280661    0.4358663    1.7652982    0.7622377
```

```
# Option 2: dplyr (probably not efficient here)
iris %>% dplyr::select(Sepal.Length,Sepal.Width,Petal.Length,Petal.Width) %>% summarise_all(funs(sd))
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1    0.8280661   0.4358663     1.765298   0.7622377
```

```
# Option 3: built-in R function (using matrixStats package)
colSds(as.matrix(iris[1:4]))
```

```
## [1] 0.8280661 0.4358663 1.7652982 0.7622377
```

2. Report the 20th and 80th percentiles for each variable in the Iris dataset. Hint: use the `quantile` function.

```
# Option 1: using sapply (thanks Daniel!)
sapply(iris[1:4], quantile, probs = c(.2, .8))
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width
## 20%         5.00         2.7         1.50         0.2
## 80%         6.52         3.4         5.32         1.9
```

```
# Option 2: using apply
apply(iris[1:4],2,quantile,probs = c(.2,.8))
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width
## 20%         5.00         2.7         1.50         0.2
## 80%         6.52         3.4         5.32         1.9
```

3. Compute the variance-covariance matrix for the four variables in the Iris dataset. Confirm your answer by using the `cov` function. Hint: the covariance matrix is given by $(n-1)^{-1}\mathbf{X}^{*T}\mathbf{X}^{*}$ where $\mathbf{X}^{*}$ contains

mean centered values for each of the four variables.

```
# Option 1
#Step 1: created the function
meancent <- function(x) {x - mean(x)}

#Step 2: apply over the function
d <- apply(iris[1:4],2,meancent)

# Option 2: combine both steps into one line
d <- apply(iris[,1:4],2,function(x) {x - mean(x)})
(t(d)%*%d)/(nrow(iris)-1)
```

```
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length    0.6856935  -0.0424340    1.2743154   0.5162707
## Sepal.Width    -0.0424340   0.1899794   -0.3296564  -0.1216394
## Petal.Length    1.2743154  -0.3296564    3.1162779   1.2956094
## Petal.Width     0.5162707  -0.1216394    1.2956094   0.5810063
```

```
# Check answer
cov(iris[,1:4])
```

```
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length    0.6856935  -0.0424340    1.2743154   0.5162707
## Sepal.Width    -0.0424340   0.1899794   -0.3296564  -0.1216394
## Petal.Length    1.2743154  -0.3296564    3.1162779   1.2956094
## Petal.Width     0.5162707  -0.1216394    1.2956094   0.5810063
```

4. In a given library of RNA-seq reads, there are duplicate observations due to PCR amplification. However, you know that there are truly $N = 50,000$ unique reads. Your collaborator wants to know how many reads she should sequence ($K \geq N$) to get a good saturation of her library (i.e. capture as many unique reads as possible). Assume that sampling from the N unique reads occurs with replacement and equal probability. What is the expected number of unique reads for the values of K between 50,000 and 200,000 (use increments of 10,000)?

```
set.seed(11)

# Option 1: with a loop (thanks, Izzy!)
reads <- c(1:50000)
expected <- numeric(15)
for (j in seq(50000,200000,10000)) {
  avg <- 0
  for (i in 1:100) {
    samp <- sample(reads,j,replace=TRUE)
    num_unique <- length(unique(samp))
    avg <- avg + (num_unique)/100
  }
  expected[((j-50000)/10000)] <- avg
}
expected
```

```
##  [1] 34932.77 37672.01 39909.94 41727.10 43221.42 44457.82 45470.95
##  [8] 46285.41 46957.68 47511.76 47961.89 48334.22 48635.26 48875.92
## [15] 49078.65
```

```
# Option 2: with no loop
#Step 1: Enumerate all values for K
```

```r
K = seq(50000,200000,by=10000)

#Step 2: Create a function
estimateNunique <- function(N,k){length(unique(sample(1:N, k, replace = TRUE)))}

#Step 3: sapply!
sapply(K,estimateNunique, N=50000)
```

```
##  [1] 31680 34941 37704 39945 41801 43208 44458 45556 46255 47033 47608
## [12] 47986 48301 48649 48851 49087
```

```r
# Option 3: one line solution
sapply(K,function(N,k){length(unique(sample(1:N, k, replace = TRUE)))}, N = 50000)
```

```
##  [1] 31589 34942 37719 39996 41709 43105 44476 45444 46277 46939 47554
## [12] 48000 48286 48593 48883 49078
```

```r
# Option 4: one line solution using piping
sapply(K,function(N,k){sample(1:N, k, replace = TRUE) %>% unique() %>% length()}, N = 50000)
```

```
##  [1] 31619 34784 37774 39942 41876 43097 44482 45472 46296 46999 47395
## [12] 48012 48353 48609 48893 49053
```

```r
# Caveat: for expected number, you should technically be averaging (like Izzy did in her loop!)
estimateNunique <- function(k, N, nrep=10){
  sapply(1:nrep, function(i){
    sample(1:N, k, replace = TRUE) %>% unique() %>% length()
  }) -> sim_vec
  return(mean(sim_vec))
}

sapply(K, estimateNunique, N = 50000)
```

```
##  [1] 31578.1 34928.1 37676.2 39906.4 41736.9 43222.1 44483.9 45437.2
##  [9] 46281.4 46961.3 47505.3 47940.2 48327.4 48593.4 48888.5 49077.8
```

While Loops and conditional statements

1. Roll a fair six sided die; if the roll is a prime number, print "prime"; if the roll is a composite number, print "composite"; otherwise print "1"

```r
# Hint: the %in% function shows if an element is in a vector
2 %in% c(1,2,3)
```

```
## [1] TRUE
```

```r
# Solution
die <- sample(6,1)
if (die %in% c(2,3,5)){
  print("prime")
  } else if (die %in% c(4,6)){
  print("composite")
  } else {
  print("1")}
```

```
## [1] "composite"
```

2. The initial value of the number is 0. If the number is less than 10, then add a random number between 0 and 1 to it and print the resulting sum. Continue until the total sum is greater than 10.

```r
#Hint: the runif function returns a random number between 0 and 1
runif(n=1,min=0,max=1)
```

```
## [1] 0.5898732
```

```r
# Solution
num <- 0
while(num < 10){
  num = num + runif(n=1,min=0,max=1)
  print(num)}
```

```
## [1] 0.1225798
## [1] 0.2953144
## [1] 0.879078
## [1] 1.658828
## [1] 2.323904
## [1] 2.696939
## [1] 2.735539
## [1] 3.08817
## [1] 3.193703
## [1] 3.823458
## [1] 4.306577
## [1] 4.949586
## [1] 5.664576
## [1] 5.683325
## [1] 5.845616
## [1] 6.083945
## [1] 6.3395
## [1] 6.535332
## [1] 6.688843
## [1] 7.175337
## [1] 7.918058
## [1] 8.623856
## [1] 9.359821
## [1] 9.748814
## [1] 9.843532
## [1] 10.3194
```

3. The Newton-Raphson method (or Newton Method) is a simple iterative technique for finding the zero r of a real-valued function $f(x)$. First, we provide some initial guess x_0 for r. The goal is then to obtain a better estimate x_n of r in n iterations. Using the Newton-Raphson method, find the 5^{th} root of 7.

- The algorithm can be summarized as follows. First, we can rewrite $r = x_0 + h$ for some h which measures how far the estimate initial guess x_0 is from the true zero r. Assuming h is small, we can use linear approximation to conclude that

  $$0 = f(r) = f(x_0 + h) \approx f(x_0) + hf'(x_0)$$

  It then follows that

  $$h \approx -f(x_0)/f'(x_0)$$

  and therefore a better estimate of r is given by

  $$x_1 = x_0 + h \approx x_0 - f(x_0)/f'(x_0)$$

Continuing in this way, if x_n is the current estimate of r then x_{n+1} is given by

$$x_{n+1} \approx x_n - f(x_n)/f'(x_n)$$

- Hint: You may want to specify a small pre-specified tolerance level in place of the approximations. In some cases, it may also be useful to specify a maximum number of iterations.

```
#Solution
max.it = 100 # max number of iterations
tol.level = 1e-7 # desired tolerance level
x_0 = 2 # initial guess
iters = x_0 # will store iterations
i = 1 # counts iterations
diff = tol.level + 1 # arbitrary difference to start,
while(i < max.it & diff > tol.level){
    f.eval = x_0^5 - 7;
    fprime.eval = 5*x_0^4;
    next.guess = x_0 - f.eval/fprime.eval
    diff = abs(x_0 - next.guess)
    iters = c(iters, next.guess)
    x_0 = next.guess
    i = i+1
    }
iters
```

```
## [1] 2.000000 1.687500 1.522645 1.478571 1.475784 1.475773 1.475773
```

```
#Check answer
7^(1/5)
```

```
## [1] 1.475773
```