MA589 Project 1

Megha Pandit

September 26, 2018

1. (WaRming up) Write (R) functions that return:

(a) The inverse or the transpose inverse of an upper triangular matrix. Call this function inv.upper.tri and provide a transpose argument to specify if the transpose is requested. Hint: use backsolve.

```
inv.upper.tri<-function(A = matrix, v = matrix, transpose){
  backsolve(r = A,x = v,transpose = transpose)
}
```

(b)Quick check: if u <- 1e200 * rep(1, 100), what is norm2(u)?

If we directly do a crossprod() on u, we get Inf as the answer, since 1e200 is a large value. Therefore, we can multiply and divide the each element of the vector u by the maximum value in the vector and then find the norm2 of u.

```
u <- 1e200*rep(1,100)
norm2 <- function(v){
  max_v <- max(abs(v))
  sqrt(crossprod(v/max(v), v/max(v)))*max(v)
}
norm2(u)
```

```
##         [,1]
## [1,] 1e+201
```

(c) The column-normalization U of matrix A, $U_{ij} = A_{ij}/||A_j||$ (call this function normalize.cols, and feel free to use norm2 above).

```
A <- matrix(1:20, 5, 4)
normalize.cols <- function(A){
  U <- matrix(0, nrow = dim(A)[1], ncol = dim(A)[2])
  for (i in 1:dim(A)[1]){
    for (j in 1:dim(A)[2]) {
      U[i,j] <- A[i,j]/norm2(A[,j]) #using the norm2 function from part (b)
    }
  }
  U
}
normalize.cols(A)
```

```
##           [,1]      [,2]      [,3]      [,4]
## [1,] 0.1348400 0.3302891 0.3761921 0.3963019
## [2,] 0.2696799 0.3853373 0.4103913 0.4210708
## [3,] 0.4045199 0.4403855 0.4445906 0.4458397
## [4,] 0.5393599 0.4954337 0.4787899 0.4706085
## [5,] 0.6741999 0.5504819 0.5129892 0.4953774
```

(d) Quick check: what is proj(1:100, u), u as in (b) above?

For very large values $proj_u(a) = \frac{u^T a}{\|u\|^2} u$ ***tends to zero as*** $\|u\|^2$ ***tends to infinity. Therefore, we can split the expression as*** $proj_u(a) = \frac{u^T a}{\|u\|} \frac{u}{\|u\|}$.

```
a <- 1:100
u <- 1e200 * rep(1, 100)
proj_1 <- function(a, u) {
  (drop(crossprod(u, a)) / drop(norm2(u))) * (u / drop(norm2(u)))
}
proj_1(a = a, u = u)
```

```
##   [1] 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5
##  [15] 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5
##  [29] 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5
##  [43] 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5
##  [57] 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5
##  [71] 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5
##  [85] 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5
##  [99] 50.5 50.5
```

(e) The Vandermonde matrix of vector $a = [a_i]_{i=1,\ldots,n}$ and degree d

```
vander_monde <- function(a,d){
  V <- matrix(0, nrow = length(a), ncol = d+1)
  for (j in 1:(d+1)){
    for (i in 1:length(a)){
      V[i,j] <- a[i]^(j-1)
    }
  }
  V
}
c <- rep(1:4)
vander_monde(c,4)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    1    1    1    1
## [2,]    1    2    4    8   16
## [3,]    1    3    9   27   81
## [4,]    1    4   16   64  256
```

2. The machine epsilon, ϵ, can be defined as the smallest floating point (with base 2) such that $1 + \epsilon > 1$, that is, $1 + \epsilon/2 == 1$ in machine precision.

(a) Write a function that returns this value by starting at eps = 1 and iteratively dividing by 2 until the definition is satisfied.

```
machine_ep <- function(eps){
  while (1 + (eps/2) !=1) {
    eps <- eps/2
  }
  eps
}
machine_ep(1)
```

```
## [1] 2.220446e-16
```

(b) Write a function that computes f(x) = log(1 + exp(x)) and then evaluate: f(0), f(???80), f(80), and f(800).

```
fun_x <- function(x){
  log(1 + exp(x))
```

```
}
fun_x(0)
```

```
## [1] 0.6931472
```

```
fun_x(-80)
```

```
## [1] 0
```

```
fun_x(80)
```

```
## [1] 80
```

```
fun_x(800)
```

```
## [1] Inf
```

Since exp(-80) is a very small number almost equal to 0, log(1+0) = log(1) = 0, Hence, fun_x(-80) is zero. Similarly, since exp(800) is a very large number tending to infinity, fun_x(800) yields Inf as the answer.

(c) How would you specify your function to avoid computations if $x << 0$ (x < 0 and |x| is large)? (Hint: ϵ.)

Since machine epsilon is defined as the smallest floating point, for any x < epsilon/2, we can have f(x)=0

```
fun_small <- function(x1){
  if(exp(x1) > machine_ep(x1)/2){
    log(1 + exp(x1))
  }
  else{
    log(1)
  }
}
fun_small(-80)
```

```
## [1] 0
```

(d) How would you implement your function to not overflow if $x >> 0$?

To avoid computations when x >>0, we consider the smallest value of x for which log(1 + exp(x)) = log(exp(x)). We define s be such that log(1 + exp(s)) = log(exp(s)). If x > s, then we just return x without calculating f(x).

```
fun_large <- function(x){
  s<- 1
  while(exp(s) != exp(s)+1){
    s = s+1
  }
  s
  if(x < s){
    fun_x(x)
  }else{
    x
  }
}
fun_large(800)
```

```
## [1] 800
```

3.(a)Show that $C = Q^T A$ is upper triangular and that C is the Cholesky factor of $A^T A$.

If A is a positive definite matrix, then we can find an upper triangular matrix C such that $A = C^T C$. ***This process is called Cholesky Decomposition. Applying QR Decomposition on A, we have*** $A = QR$ ***where Q is an orthogonal matrix and R is an upper triangular matrix. Given*** $C = Q^T A$, ***Q is an orthogonal matrix and hence*** $Q^{-1} = Q^T$.

$$C = Q^T A = Q^{-1} A$$

$$C = Q^{-1} QR = R$$

Since R is an upper triangular matrix and C = R, C is an upper triangular matrix. To show that C is the Cholesky factor of $A^T A$, ***we need to show that*** $A^T A = C^T C$ ***We have*** $C = Q^T A$.

$$C^T C = (Q^T A)^T (Q^T A)$$

$$C^T C = A^T Q Q^T A = A^T Q Q^{-1} A = A^T I A = A^T A$$

Therefore, $A^T A = C^T C$ ***and C is the Cholesky factor of*** $A^T A$.

3.(b) Write a R function that computes the Q orthogonal factor of a Vandermonde matrix with base vector x and degree d without computing the Vandermonde matrix explicitly, that is, as your function iterates to compute u_i, compute and use the columns of the Vandermonde matrix on the fly.

```
q_fun <- function(a,d){
  U <- matrix(nrow = length(a), ncol = d+1)
  e <- rep(1,d+1)
  x <- rep(0,d+1)
  U[,1] <- rep(1, length(a))
  e[2] <- crossprod(U[,1],U[,1])
  for (i in 2:(d+1)){
        sum_proj <- 0
    U[,i] <- a^(i-1)
    for (j in 1:(i-1)){
      proj_u <- proj_1(U[,i],U[,j])
      sum_proj <- sum_proj + proj_u
    }
    U[,i] <- U[,i] - sum_proj
    e[i+1] <- drop(crossprod(U[,i],U[,i]))    #Calculating eta for part (c)
    x[i] <- t(U[,i])%*% diag(a) %*%U[,i]/(drop(crossprod(U[,i],U[,i])))   # Calculating alpha for part(
  }
  Q <- normalize.cols(U)
  return (list(Q,e,x))
}
a <- c(1,2,3)
d <- 3
Q_ortho <- q_fun(a,d)
Q_ortho
```

```
## [[1]]
##           [,1]          [,2]       [,3]       [,4]
## [1,] 0.5773503 -7.071068e-01  0.4082483 -0.3165797
## [2,] 0.5773503 -3.140185e-16 -0.8164966  0.8020019
## [3,] 0.5773503  7.071068e-01  0.4082483 -0.5065275
##
## [[2]]
## [1] 1.000000e+00 3.000000e+00 2.000000e+00 6.666667e-01 7.083971e-27
##
```

```
## [[3]]
## [1] 0.000000 2.000000 2.000000 2.156347
```

```
#For part(c) of the question
q <- Q_ortho[[1]]
e <- Q_ortho[[2]]
x <- Q_ortho[[3]]
```

3.(c) Write a R function that, given ?? and ??, computes Q.

```
x
```

```
## [1] 0.000000 2.000000 2.000000 2.156347
```

```
q_comp <- function ( e = vector , x = vector ,  a = vector ){
  Q <-  matrix(0, nrow = length(a), ncol = d+1 )
  Q[,1] <- 1
  Q[,2] <- a - x[1]*rep(1,length(a))
  for ( i in 2:d){
    for ( j in 1:length(a)){
      Q[j,i+1] <- ((a[j] - x[i]) * Q[j,i])- (e[i+1]/e[i] * Q[j,i-1]) #algorithm from the question
    }
  }
  return(Q)
}

Q <- q_comp(e,x,a)
Q_new <- normalize.cols(Q)
Q_new
```

```
##           [,1]      [,2]       [,3]       [,4]
## [1,] 0.5773503 0.2672612 -0.5661385  0.6666667
## [2,] 0.5773503 0.5345225 -0.2264554 -0.3333333
## [3,] 0.5773503 0.8017837  0.7925939  0.6666667
```

3.(d)

```
#Alpha1 is the mean of vector a
x[1] == mean(a)
```

```
## [1] FALSE
```

```
#Eta2 is the number of vectors in vector a
e[2] == length(a)
```

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

```
#Eta3 gives the value of (n-1)*Var(a)
e[3] == (length(a) -1 )*var(a)
```

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

4. (a) To prove that $H_0 : \beta_j = \beta_{j+1} = ... = \beta_p = 0$ is equivalent to testing $\gamma_j = ... = \gamma_p = 0$, where $\gamma = R\beta$

Since X has thin QR Decomposition, $X = QR$,

$$\beta = (X^T X)^{-1} X^T y = [(QR)^T QR]^{-1} (QR)^T y$$

$$\beta = [R^T Q^T QR]^{-1} R^T Q^T y$$

Since Q is an orthogonal matrix, $Q^TQ = I$

$$\beta = (R^TR)^{-1}R^TQ^Ty = R^{-1}R^{-T}R^TQ^Ty = R^{-1}Q^Ty$$

Therefore,

$$R\beta = Q^Ty = (Q^TQ)^{-1}Q^Ty$$

Since $\gamma = R\beta$,

$$\gamma = Q^Ty$$

, H_0 ***is equivalent to testing*** $\gamma_j = ... = \gamma_p = 0$ ***Hence,*** $y = Q\gamma$ ***and y can be regressed on Q instead of X. And,***

$$Var(\gamma) = Var(Q^Ty) = E((Q^Ty)(Q^Ty)^T) = Q^TE(Y^TY)Q = \sigma^2 I_n Q^TQ = \sigma^2 I_n$$

4.(b) Show that the ML estimator for γ is $\hat{\gamma} = Q^Ty$ and the components of $\hat{\gamma}$ are independent.

Since y $N(Q\gamma, \sigma^2 I_n)$, $y = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-1}{2\sigma^2}(y-Q\gamma)^2}$ ***So,*** $y = (constant)e^{-(y-Q\gamma)^2} = (constant)e^{-(y-Q\gamma)^T(y-Q\gamma)}$ ***To maximize the expectation, we need to minimize*** $f = (y-Q\gamma)^T(y-Q\gamma)$ ***Equating*** $\frac{\partial f}{\partial \gamma}$ ***to zero, we get***

$$\frac{\partial(y^Ty - 2Q^Ty\gamma + (Q\gamma)^T(Q\gamma))}{\partial\gamma} = 0, \quad i.e., \quad -2Q^Ty + 2\hat{\gamma} = 0$$

Therefore,

$$\hat{\gamma} = Q^Ty$$

In order to prove that the components of $\hat{\gamma}$ ***are independent, we can show that the covariance or the non-diagonal terms of the*** $Cov[\hat{\gamma}]$ ***are zero.***

$$cov[\hat{\gamma}] = cov[Q^Ty] = E((Q^Ty)(Q^Ty)^T) = E[Q^Tyy^TQ]$$

$$cov[\hat{\gamma}] = Q^TE(yy^T)Q = Q^T\sigma^2 I_n Q = \sigma^2 I_n$$

$\sigma^2 I_n$ ***matrix has its non-diagonal elements equal to zero, and hence, the components of*** $\hat{\gamma}$ ***are independent***

4.(c)Using R, explain how you compute: (i) the ML estimate ??̂ as a function of ??̂, and (ii) the correlation matrix of ??̂ using only crossprod, normalize.cols, and inv.upper.tri.

Since $\gamma = R\beta$,

$$\hat{\beta} = R^{-1}\hat{\gamma}$$

In R, we can use the inv.upper.tri function that we defined in question 1. #b = beta hat and g = gamma vector and R is the upper triangular matrix b <- inv.upper.tri(R, g, transpose = FALSE)

In R,

$$cor(\hat{\beta}) = crossprod(normalized.cols(\hat{\beta}), normalized.cols(\hat{\beta}))$$

$$cor(\hat{\beta}) = crossprod(normalized.cols(inv.upper.tri(R, \hat{\gamma}), normalized.cols(inv.upper.tri(R, \hat{\gamma}))$$

4.(d)(i) Compute Q using the routine from 3.b, obtain $\hat{\gamma} = Q^Ty$ and compare it to the estimate from coef(lm(dist ~ Q - 1)).

```
data(cars)
y <- as.vector(cars$dist)
Q_cars <- q_fun(as.vector(cars$speed), 3)[[1]]
gamma <- crossprod(Q_cars, y) #estimate of gamma as crossprod of Q and y
gamma
```

```
##           [,1]
## [1,] 303.91449
## [2,] 145.55226
## [3,]  22.99576
## [4,]  13.79688
```

```
gamma1 <- coef(lm(cars$dist ~ Q_cars -1))
gamma1
```

```
##   Q_cars1   Q_cars2   Q_cars3   Q_cars4
## 303.91449 145.55226  22.99576  13.79688
```

4.(d)(ii) Compute $\hat{\beta}$ according to (c) and compare it to the estimate from coef(lm(dist ~ vandermonde(speed, 3) - 1))

```
data(cars)
V <-(vander_monde(cars$speed, 3))
colnames(V) <- c("cars1","cars2","cars3","cars4")

Q <- q_fun(cars$speed, 3)
q_cars <- Q[[1]]
G <- crossprod(q_cars,cars$dist)
G
```

```
##           [,1]
## [1,] 303.91449
## [2,] 145.55226
## [3,]  22.99576
## [4,]  13.79688
```

```
coef(lm(cars$dist~ q_cars))
```

```
## (Intercept)     q_cars1     q_cars2     q_cars3     q_cars4
##    42.98000          NA   145.55226    22.99576    13.79688
```

```
beta1 <- coef(lm(cars$dist ~ V - 1))
beta1
```

```
##       Vcars1       Vcars2       Vcars3       Vcars4
## -19.50504910   6.80110597  -0.34965781   0.01025205
```

```
qr <- qr(vander_monde(cars$speed,3))
R <- qr.R(qr)
R
```

```
##           [,1]       [,2]       [,3]       [,4]
## [1,] -7.071068 -108.89444 -1870.7217 -34660.960
## [2,]  0.000000   37.01351  1117.9377  27771.535
## [3,]  0.000000    0.00000  -230.0513 -10089.202
## [4,]  0.000000    0.00000     0.0000  -1345.769
```

```
In <- diag( rep(1,ncol(R)) )
crossprod(inv.upper.tri( R, In , TRUE), G)
```

```
##               [,1]
## [1,] -101.61314068
## [2,]    1.06371154
## [3,]    0.34965781
## [4,]   -0.01025205
```