Project 1

Benjamin Draves

Exercise 1

(a)

In this exercise we look to find a matrix M such that $UM = MU = I$ where $U \in \mathbb{R}^{n\times n}$ is an upper triangular matrix. As we discussed in class, we can use backsolve for an efficient solution to finding the solution of $U^{-1}b$ by rewriting the system as $U\vec{x} = \vec{b}$ and solving for $\vec{x}$. Rewriting our problem in a similar way we have

$$UM = I$$
$$\left[U\vec{m}_1 \middle| U\vec{m}_1 \middle| \dots U\vec{m}_n\right] = \left[\vec{e}_1 \middle| \vec{e}_2 \middle| \dots \middle| \vec{e}_n\right]$$

where $\{\vec{e}_i : 1 \leq i \leq n\}$ is the standard coordinate basis. Hence we see that finding the inverse to this matrix matrix is equivalent to using backsolve to find the solution to the system $U\vec{m}_i = \vec{e}_i$ for $1 \leq i \leq n$. In a similar fashion of we with to find the inverse of U^T we look to solve the equations $U^T\vec{m}_i = \vec{e}_i$ for $1 \leq i \leq n$. In either case, n applications of backsolve will complete this task.

When we apply backsolve in R, we can pass an entire matrix in which case backsolve will be applied column wise to the upper triangular matrix and the input matrix. Moreover, we can use the built in transpose argument to solve have the function also be able to find the transpose inverse.

```
inv.upper.tri <- function(U, trans = F) backsolve(U, diag(1, nrow(U)), transpose = trans)
```

(b)

Here we look to return the L_2 norm for a vector $\vec{v}$, $||\vec{v}||_2 = \sqrt{\sum_{i=1}^n v_i^2}$. However, when $\vec{v}_i$ is very large, we see that v_i^2 can lose percision due to overflow errors. Instead consider the following

$$||\vec{v}||_2 = \sqrt{\sum_{i=1}^n v_i^2} = \sqrt{v_{max}^2 \sum_{i=1}^n \left(\frac{v_i}{v_{max}}\right)^2} = |v_{max}|\left|\left|\frac{\vec{v}}{v_{max}}\right|\right|_2$$

Here we see that as $-1 \leq \left|\frac{v_i}{v_{max}}\right| \leq 1$ that we can avoid overflow errors with this operation. Consider the implementation of this procedure below.

```
norm2 <- function(u){
  umax <- max(u) #get max
  unorm <- u/umax #normalize u
  abs(umax)*sqrt(sum(unorm^2)) #return L2 norm
}

u <- 1e200 * rep(1, 100)
norm2(u)
```

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

(c)

This is a simple application of the L_2 norm we constructed in the previous problem. In an attempt to use the vectorized operations in R, we locally define a function *norm.vec* which returns the L_2 normalized vector of the input. We then apply this functions across the columns A.

```
normalize.cols<- function(A){
  norm.vec <- function(u) u/norm2(u) #define vector normalization func.
  apply(A, 2, norm.vec) # vectorize
}
```

(d)

Here we look to calculate $\text{proj}_u(a)$. Here we must calculate the value of $||u||_u^2$ which based on the entries of u could lead to overflow issues. Instead, as we see that this is just a normalization term for the u vectors, we can rewrite the expression as follows

$$\text{proj}_u(a) = \frac{u^T a}{||u||_2^2} u = \left(\frac{u}{||u||_2}\right)^T a \left(\frac{u}{||u||_2}\right)$$

Now, as this is just an innerproduct followed by a dot product we use crossprod then the multiplying operation in R. See below for the implementation.

```
proj <- function(a, u) as.numeric(crossprod(u/norm2(u), a)) * u/norm2(u)

str(proj(1:100, u))
```

```
##  num [1:100] 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 50.5 ...
```

(e)

Here, since we are constructing a full matrix here we will need to use $O(n^2)$ memory and the same order of calculations. But we can localize the operations so that all of our data exists in nearby physical location in memory. In particular, we now every Vandermonde matrix starts with a column of full 1s. From here we can iteratively update the columns using the operation $V_{\cdot j+1} = a * V_{\cdot j}$ where $*$ is the element wise product.

```
vandermonde <- function(a, d){
  V <- matrix(1, nrow = length(a), ncol = d + 1) #allocate memory
  for(i in 1:d) V[,i+1] <- a * V[,i]  #fill memory
  V #return
}
```

Exercise 2

(a)

Here we can find the machine epsilon by iteratively checking the criterion of the machine-ϵ (i.e $1 + \epsilon/2 == 1$). We complete this task using a while loop constantly checking the criterion stated above.

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

```
  eps
}
```

(b)

Here we implement and evaluate the function $f(x) = \log(1 + \exp(x))$

```
f <- function(x) log(1 +exp(x))
c(f(0),f(-80),f(80),f(800))
```

```
## [1]  0.6931472  0.0000000 80.0000000        Inf
```

Here we see that $f(0) = \log(2)$, $f(-80) = 0$ while analytically the function should be strictly positive, $f(80) = 80$ while analytically the function should be slightly larger than 80, and lastly $f(800) = \infty$ while this should be a finite value.

(c)

In the case above $f(-80) = 0$ we see that the function is treating the value $\exp(-80) = 0$ then just returning $\log(1) = 0$. Hence to avoid these calculations we need to find the value for which R can't distinguish the difference between $\exp(x)$ and 0. Well, given our discussion in part (a) we know that the machine-ϵ acts as the smallest floating point number that our memory system can support. Hence we see that $\exp(x) < \epsilon$ will simply return 0. Therefore, if we wish to avoid computations, if $x < \log(\epsilon)$ we should simply return 0. To highlight this point, consider the following.

```
e <- machine_epsilon()
c(f(log(e)-1), f(log(e)), f(log(e)+1))
```

```
## [1] 0.000000e+00 2.220446e-16 6.661338e-16
```

So we see that in this case, $f(\log(\epsilon)) = \epsilon$ as expected and $f(\log(\epsilon) - 1) = 0$, $f(\log(\epsilon) + 1) > 0$. Implementing this procedure we have

```
f <- function(x){
  if(x < log(machine_epsilon())) return(0) #very small values
  log(1+ exp(x)) #standard function
}
```

(d)

In a similar way we defined the machine-ϵ there exists some machine bound B such that $B + 1 = B$. Hence for any x with $x > \log(B)$ then $\exp(x) > B$ and $\exp(x) + 1 = \exp(x)$. In this case, our function should return $\log(1 + \exp(x)) = \log(\exp(x)) = x$ or just the identity. So the final addition to this function would be to test $x > \log(B)$ and simply return x if that were true.

Exercise 3

(a)

Suppose that $\mathbf{A} = [a_1 a_2 \dots a_p] \in \mathbb{R}^{n \times p}$ where $n > p$. The using Gram-Schmidt we can obtain the thin QR decomposition with $\mathbf{Q} = [q_1 q_2 \dots q_p] \in \mathbb{R}^{n \times p}$ where $q_i = u_i / ||u_i||_2$ where

$$u_i = a_i - \sum_{j=1}^{i-1} \text{proj}_{u_j}(a_i)$$

Now we look to show that $\mathbf{Q}^T \mathbf{A}$ is upper triangular for some values of $\mathbf{A}$.

```
gram_schmidt <- function(A){
  U <- matrix(c(A[,1], rep(NA,dim(A)[1]*(dim(A)[2]-1))),
              ncol = ncol(A),
              nrow = nrow(A)) # set first column of U to be a1

  for(i in 2:(ncol(A))){ #fill the remainder of U
    proj_sum <- 0 #get the take away sum of projecting on u_1,..., u_{i-1}
    for(j in 1:(i-1)) proj_sum <- proj_sum + proj(A[,i],U[,j])
    U[,i] <- A[,i] - proj_sum #set u_i
  }
  normalize.cols(U) #returned normalized columns
}

A <- matrix(rnorm(10), ncol = 2)
Q <- gram_schmidt(A)
crossprod(Q,A)
```

```
##           [,1]       [,2]
## [1,] 3.060932 -0.6840035
## [2,] 0.000000  2.0460871
```

```
abs(crossprod(Q,A) - chol(crossprod(A)))
```

```
##      [,1]         [,2]
## [1,]    0 1.110223e-16
## [2,]    0 4.440892e-16
```

(b)

```
vandermonde_orth_factor <- function(x, d){

  U <- matrix(c(rep(1, length(x)), rep(NA, length(x)*d)),
              nrow = length(x), ncol = d+1) #initialize U with first column a_1

  for(i in 2:(d+1)){ #fill the remainder of U
    a <- x^(i-1) #set a
    proj_sum <- 0 #get the take away sum of projecting on u_1,..., u_{i-1}
    for(j in 1:(i-1)) proj_sum <- proj_sum + proj(a,U[,j])
    U[,i] <- a - proj_sum #set u_i
  }
  normalize.cols(U) #returned normalized columns
}

Q <- vandermonde_orth_factor(c(1,2,3),4)
crossprod(Q)
```

```
##               [,1]          [,2]          [,3]       [,4]       [,5]
## [1,]  1.000000e+00 -5.551115e-16  9.992007e-16 -0.0360453 -0.0360453
## [2,] -5.551115e-16  1.000000e+00  3.441691e-15 -0.1030080 -0.1030080
## [3,]  9.992007e-16  3.441691e-15  1.000000e+00 -0.9940272 -0.9940272
## [4,] -3.604530e-02 -1.030080e-01 -9.940272e-01  1.0000000  1.0000000
## [5,] -3.604530e-02 -1.030080e-01 -9.940272e-01  1.0000000  1.0000000
```

(c)

Here we build a function that given η, $\mathbf{x}$, and α finds the corresponding $\mathbf{Q}$ matrix

```
build_Q <- function(eta, alpha, x,d){
  U <- matrix(c(rep(1, length(x)),rep(x - alpha[1], length(x)), rep(NA, length(x) *(d-1))),
              nrow = length(x),
              ncol = d+1) #initialize U1 = 1, U2 = a1 * 1, Ui = NA i > 2

  for(i in 2:d){#fill the rest of U
    for(j in 1:length(x)) U[j,i+1] <- (x[j] - alpha[i])*U[j,i] - eta[i+1]/eta[i]*U[j,i-1]
  }
  normalize.cols(U)#return Q
}
```

(d)

Here we extend the function we created in (b) to calculate α and η.

```
modified_gram_schmidt <- function(x,d){
  #initialize variables
  U <- matrix(c(rep(1, length(x)), rep(NA, length(x)*d)),
              nrow = length(x), ncol = d+1) #initialize U with first column a_1

  alpha <- c(sum(x*U[,1]^2)/sum(U[,1]^2), rep(NA,d)) #set alpha
  eta <- c(1, rep(NA,  d))

  for(i in 2:(d+1)){ #fill the remainder of U
    a <- x^(i-1) #set a
    proj_sum <- 0 #get the take away sum of projecting on u_1,..., u_{i-1}
    for(j in 1:(i-1)) proj_sum <- proj_sum + proj(a,U[,j])
    U[,i] <- a - proj_sum #set u_i

    alpha[i] <- sum(x*U[,i]^2)/(sum(U[,i]^2)) #update alpha
    eta[i] <- sum(U[,i-1]^2) #update eta

  }
  list(Q = normalize.cols(U), alpha = alpha, eta = eta) #returned normalized columns
}
```

Here we look to test the function on the vector $x = c(1, 2, \ldots, 10)$ and $d = 4$.

```
x <- 1:10
d <- 4
obj <- modified_gram_schmidt(x,d)
c(obj$alpha[1],obj$eta[2], obj$eta[3])
```

```
## [1]  5.5 10.0 82.5
```

```
c(mean(x),length(x), (length(x)-1)*var(x))
```

```
## [1]  5.5 10.0 82.5
```

```
#check to see if (c) and (d) agree
Q_new <- build_Q(obj$eta, obj$alpha, x, d)
Q_new - obj$Q
```

```
##       [,1] [,2]          [,3]          [,4]          [,5]
##  [1,]    0    0  5.551115e-16  1.021405e-14  5.485612e-13
```

```
##  [2,]    0    0  6.383782e-16  2.192690e-15 -2.229883e-13
##  [3,]    0    0  4.579670e-16 -4.329870e-15 -4.842793e-13
##  [4,]    0    0  1.665335e-16 -6.161738e-15 -4.132389e-13
##  [5,]    0    0  0.000000e+00 -8.021361e-15 -1.211808e-13
##  [6,]    0    0  0.000000e+00 -8.354428e-15  2.284839e-13
##  [7,]    0    0  0.000000e+00 -7.882583e-15  5.069070e-13
##  [8,]    0    0 -6.245005e-16  2.442491e-15  4.964917e-13
##  [9,]    0    0 -6.106227e-16  5.218048e-15  1.754152e-13
## [10,]    0    0 -6.661338e-16  1.587619e-14 -7.182033e-13
```

Exercise 4

(a)

Suppose that $\mathbf{X} = \mathbf{QR}$ has a thin QR decomposition with $\mathbf{Q} \in \mathbb{R}^{n\times p}$ and $\mathbf{R} \in \mathbb{R}^{p\times p}$ is an upper triangular matrix. Moreover, due to the fact that $\mathbf{X}$ is full rank we see that $\mathbf{X}^T\mathbf{X}$ is also full rank p. Now recall that $\mathbf{R}$ is the Cholesky factor of $\mathbf{R}$ so $\mathbf{X}^T\mathbf{X} = \mathbf{R}^T\mathbf{R}$. Thus, $\text{rank}(\mathbf{R}^T\mathbf{R}) = \text{rank}(\mathbf{X}^T\mathbf{X})$. Hence, by the Sylvester Rank inequality $\mathbf{R}$ is full rank which implies that the $\det(\mathbf{R}) \neq 0$. Therefore the diagonal entries of $\mathbf{R}$ cannot be zero $\mathbf{R}_{ii} \neq 0$. Now consider the following hypothesis

$$H_0 : \beta_j = \beta_{j+1} = \cdots = \beta_p$$

Now consider the structure of $R\beta$, $(R\beta)_j = \sum_{i=j}^{p} R_{ji}\beta_p$ Using this, we could rewrite the hypothesis $H_0 : \mathbf{R}\beta = 0$ as follows

$$\begin{aligned}
\mathbf{R}_{pp}\beta_p &= 0 \\
\mathbf{R}_{p-1,p-1}\beta_{p-1} + \mathbf{R}_{p-1p}\beta_p &= 0 \\
&\vdots \\
\mathbf{R}_{jj}\beta_j + \cdots + \mathbf{R}_{j,p-1}\beta_{p-1} + \mathbf{R}_{jp}\beta_p &= 0
\end{aligned}$$

Clearly, since $\mathbf{R}_{pp} \neq 0$, the first hypothesis is equivalent to testing $\beta_p = 0$. Then if we were to simultaneous test this with the second hypothesis, under the null we assume that $\beta_p = 0$ so our second hypothesis reduces to $\mathbf{R}_{p-1,p-1}\beta_{p-1} = 0$. As $\mathbf{R}_{p-1,p-1} \neq 0$ we see that this is equivalent to testing $\beta_{p-1,p-1} = 0$. Iterating, we see that these hypothesis are in fact equivalent.

Now notice that that if we parameterized the model such that $\mathbb{E}(\mathbf{y}|\mathbf{X}) = \mathbf{QR}\beta = \mathbf{Q}\gamma$ then by assuming that $\mathbf{y}$ we see that $\mathbf{Q}$ "acts as" our new $\mathbf{X}$. Therefore, under the same OLS estimation procedure we see that $\mathbf{y} \sim N(\mathbf{Q}\gamma, \sigma^2(\mathbf{Q}^T\mathbf{Q})^{-1})$ thus

$$\mathbf{y} \sim N(\mathbf{QR}\beta, \sigma^2(\mathbf{Q}^T\mathbf{Q})^{-1}) \overset{D}{=} N(\mathbf{Q}\gamma, \sigma^2\mathbf{I}_n)$$

Hence we see that we can instead reparameterize the model and instead estimate γ.

(b)

Recall that that the maximum likelihood estimate of β was given by

$$\widehat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} = (\mathbf{R}^T\mathbf{R})^{-1}\mathbf{R}^T\mathbf{Q}^T\mathbf{Y}$$

Moreover, as we showed that R was full rank we may invert them individually to get

$$\widehat{\beta} = \mathbf{R}^{-1}\mathbf{R}^{-T}\mathbf{R}^T\mathbf{Q}^T\mathbf{Y} = \mathbf{R}^{-1}\mathbf{Q}^T\mathbf{Y}$$

Then due to invariance of MLE we see that

$$\widehat{\gamma}_{MLE} = R\widehat{\beta}_{MLE} = \mathbf{R}\mathbf{R}^{-1}\mathbf{Q}^T\mathbf{Y} = \mathbf{Q}^T\mathbf{Y}$$

Moreover, as $\widehat{\gamma}_{MLE}$ is a linear combination of $\widehat{\beta}$, it is normally distributed. Hence to show that the entries of $\widehat{\gamma}_{MLE}$ are independent if suffices to show that the covariance matrix is diagonal.

$$Var(\mathbf{Q}^T\mathbf{Y}) = \mathbf{Q}^T Var(\mathbf{Y})\mathbf{Q} = \sigma^2\mathbf{Q}^T\mathbf{Q} = \sigma^2\mathbf{I}$$

(c)

Recall that $\hat{\gamma}_{MLE} = \mathbf{R}\widehat{\beta}_{MLE}$ and $\mathbf{R}$ is an upper triangular matrix. Therefore, we can backsolve to find the MLE estimate of $\hat{\beta}$.

```
#beta_hat <- backsolve(R, gamm_hat)
```

Suppose that $\hat{\beta}$ has covariance matrix Σ. Then $\text{Corr}(\hat{\beta}) = \text{diag}(\Sigma)^{-1/2}\Sigma\text{diag}(\Sigma)^{-1/2}$. Under this model, we see that $\Sigma = \sigma^2(\mathbf{X}^T\mathbf{X})^{-1} = \sigma^2(\mathbf{R}^T\mathbf{R})^{-1}$. Notice that this is just the diagonol matrix with the L_2-norm $\mathbf{R}$ of the columns of $\mathbf{R}$. Continuing with these calculations, let $\mathbf{D} = \text{diag}[(\mathbf{R}^T\mathbf{R})^{-1}]$.

$$\text{Corr}(\hat{\beta}) = \frac{1}{\sigma}\mathbf{D}^{-1/2}\sigma^2(\mathbf{R}^T\mathbf{R})^{-1}\frac{1}{\sigma}\mathbf{D}^{-1/2} = (\mathbf{R}^{-T}\mathbf{D}^{-1/2})^T(\mathbf{R}^{-T}\mathbf{D}^{-1/2})$$

For a general matrix $\mathbf{A}$ we can get the column L_2-normalized version of $\mathbf{A}$ by considering $\mathbf{A}[\text{diag}(\mathbf{A}^T\mathbf{A})]^{-1/2}$. Notice here, however, that for $\mathbf{A} = \mathbf{R}^{-T}$ so the normalized version is given by

$$\mathbf{R}^{-T}[\text{diag}(\mathbf{R}^{-1}\mathbf{R}^{-T})]^{-1/2} = \mathbf{R}^{-T}\{\text{diag}[(\mathbf{R}^T\mathbf{R})^{-1}]\}^{-1/2} = \mathbf{R}^{-T}\mathbf{D}^{-1.2}$$

Therefore, $\mathbf{R}^{-T}\mathbf{D}^{-1/2}$ is just the column normalized version of $\mathbf{R}^{-T}$. Moreover, as $\mathbf{R}$ is upper triangular, we can use the *inv.upper.tri* function to find this inverse. Therefore, together, we can calculate this correlation in R using the following command

```
#correlation_matrix <- crossprod{normalize.cols(inv.upper.tri(R, trans = T))}
```

(d)

Here load the cars data and compare estimates for $\hat{\gamma}$ as well as $\hat{\beta}$ for the functions we build here to the built in estimates form the *lm* function.

```
data(cars)
str(cars)
```

```
## 'data.frame':    50 obs. of  2 variables:
##  $ speed: num  4 4 7 7 8 9 10 10 10 11 ...
##  $ dist : num  2 10 4 22 16 10 18 26 34 17 ...
```

(i)

```r
Q <- vandermonde_orth_factor(cars$speed, 3)
gamma_hat <- crossprod(Q, cars$dist)

rbind(t(gamma_hat),coef(lm(dist~Q-1, data = cars)))
```

```
##            Q1       Q2       Q3       Q4
## [1,] 303.9145 145.5523 22.99576 13.79688
## [2,] 303.9145 145.5523 22.99576 13.79688
```

(ii)

```r
R <- crossprod(Q, vandermonde(cars$speed, 3))
beta_hat <- backsolve(R, gamma_hat)

rbind(t(beta_hat),coef(lm(dist ~ vandermonde(speed, 3)-1, data = cars)))
```

```
##      vandermonde(speed, 3)1 vandermonde(speed, 3)2 vandermonde(speed, 3)3
## [1,]              -19.50505               6.801106             -0.3496578
## [2,]              -19.50505               6.801106             -0.3496578
##      vandermonde(speed, 3)4
## [1,]             0.01025205
## [2,]             0.01025205
```