Jennifer Starling
SDS 385
23 Aug 2016

Exercises 1

Part 1. Linear Regression

PART A)

The WLS objective function, rewritten in matrix form, is

$$\hat{\beta} = \text{argmin}_{\beta \in \mathcal{R}^P} \frac{1}{2}(Y - X'\beta)'W(Y - X'\beta) = \frac{1}{2}(Y' - \beta'X')W(Y - X'\beta)$$

To satisfy the 'argmin' part of the expression, take the derivative of $\hat{\beta}$ with respect to β, set equal to zero, and solve as follows.

$$\frac{\delta}{\delta\beta}\left[\frac{1}{2}(Y' - \beta'X')W(Y - X'\beta)\right] = \left(\frac{1}{2}\right)\frac{\delta}{\delta\beta}[Y'WY - 2Y'WX\beta + \beta'XwX'\beta] = 0$$

The derivatives of each term are as follows.

1. $\frac{\delta}{\delta\beta}[Y'WY] = 0$ since is a constant wrt β.
2. $\frac{\delta}{\delta\beta}[-2Y'WX\beta] = -2Y'WX$, since derivative has form $\frac{\delta}{\delta\beta}c\beta = c$ where $c = -2Y'WX$. Then since W diagonal and X and Y vectors, $-2Y'WX = -2X'WY$.
    a. Derivation: $f(\beta) = c'\beta = c_1\beta_1 + \cdots + c_k\beta_k \rightarrow \frac{\delta f(\beta)}{\delta\beta} = \begin{bmatrix} \frac{\delta(c_1\beta_1 + \cdots + c_k\beta_k)}{\delta\beta_1} \\ \cdots \\ \frac{\delta(c_1\beta_1 + \cdots + c_k\beta_k)}{\delta\beta_k} \end{bmatrix} = \begin{bmatrix} c_1 \\ \cdots \\ c_k \end{bmatrix} = c$
3. $\frac{\delta}{\delta\beta}[\beta'XwX'\beta] = 2XWX'\beta$, since derivative has quadratic form $\frac{\delta}{\delta\beta}(\beta'V\beta) = (V + V')\beta = 2V\beta$, where $V = XWX'$. The last equality (=$2V\beta$) applies only when V symmetric, which holds here.

Then the derivative, subbing in $\hat{\beta}$ for β, is $\left(\frac{1}{2}\right)\left[-2X'WY + 2XWX'\hat{\beta}\right] = 0 \rightarrow \ XWX'\hat{\beta} = X'WY \rightarrow \hat{\beta} = (XWX')^{-1}X'WY$

Therefore $\hat{\beta} = (XWX')^{-1}X'WY$.

To show that $\hat{\beta} = (XWX')^{-1}X'WY$ is the solution to the linear system: $(X'WX)\hat{\beta} = (X'WX)(XWX')^{-1}X'WY = IX'WY = X'WY$

PART B)

Numerically speaking, I do not believe that inversion is the fastest and most stable way to solve the linear system. There are several matrix factorization methods which provide more stability and are computationally efficient compared to inversion. Inverting a matrix directly is computationally intensive, especially as N and P become large.

Some of the methods I discovered for solving linear equations of form Ax=B without inverting the A matrix directly are: LU Decomposition, Gaussian elimination, Cholesky decomposition, QR decomposition, RRQR factorization, and the conjugate gradient method. There was not a strict consensus as to which method is universally superior; the key to know the characteristics of the matrix A, so that you can choose an optimal method. Difference characteristics lend themselves to different methods.

- Cholesky performs well for Hermitian matrices (symmetric positive definite).
- LU performs well when A is sparse, and A is only required to be square.
- Conjugate gradient requires A to be symmetric positive definite, but is a good iterative algorithm for scenarios where A is sparse and too large to be inverted directly or for Cholesky.

My method will be the Cholesky decomposition. The pseudo-code is as follows.

Goal: Solve Ax=b where A= X'WX, b=X'WY, and x= the vector of β estimates.

Function inputs:

- X, an NxP matrix
- W, a diagonal matrix of weights
- Y, a Nx1 vector of responses

Function outputs:

- beta_hat, a vector of beta estimates.

(1) Set A = X'WX
(2) Set b = X'WY
(3) Set R = Cholesky decomposition of A. (R gives R (upper) instead of L (lower) by default, see note below.)

Now we have R'R = A.

(4) Solve R'z=b for z by finding z = inverse(R')b.
(5) Solve Rx=z by finding z=inverse(R)z
(6) Return beta_hat estimate as beta_hat = x

I also included the LU decomposition, which is solved by similar steps.

PART C)

```
#-------------------------------------------------
#Inversion Method function:
#   Inputs: X = vector of x values, Y = vector of y values, W = diag matrix of weights.
#   Outputs: B = beta-hat vector; the WLS solution for estimating the beta vector of coefficients.

#   Matrix requirements:
#   1. Length X = Length Y = Dim(W)
#   2. t(X) %*% W %*% X must be invertible

inv_method <- function(X,W,y){

 B_hat <- solve(t(X) %*% W %*% X) %*% t(X) %*% W %*% y
 return(B_hat)
}

#Test data with five beta coeffs
set.seed(1)
X <- matrix(rnorm(25),nrow=5,ncol=5)
set.seed(1)
y <- rnorm(5)
W <- diag(1,nrow=5)

B_hat <- inv_method(X,W,y)
B_hat

#-------------------------------------------------

#Cholesky decomposition function:
#   Inputs: X = vector of x values, Y = vector of y values, W = diag matrix of weights.
#   Outputs: B = beta-hat vector; the WLS solution for estimating the beta vector of coefficients.

cholesky_method  <- function(X,W,y){

     #Solves linear system Ax=b.
     #Since we have (X'WX)B=X'Wy, B (beta) acts as x, with A and b as follows.

     #Finding B (beta_hat) in equation
     A = (t(X) * diag(W)) %*% X  #Efficient way of A = t(X) %*% W %*% X bc W diag matrix.
                                 #Avoids mult by 0's.
     b = t(X) %*% W %*% y  #b = X'Wy

     R <- chol(A)   #Find right/upper cholesky decomposition of A.

     #Now we have R'R=A.

     #1. Solve R'z=b for z.  This is z = inv(R')b.
     z = solve(t(R)) %*% b

     #2. Solve Rx=z for x.  This is x = inv(R)z. (x = beta_hat)
     B_hat_chol <- solve(R) %*% z

     return(B_hat_chol)
}
#-------------------------------------------------

# LU Decomposition to solve linear system Ax=b.
#   Inputs: X = vector of x values, Y = vector of y values, W = diag matrix of weights.
#   Outputs: B_hat_LU, an estimate of the 'x' in Ax=b.
```

```r
lu_method <- function(X,W,y){

     #Solves linear system Ax=b.
     #Since we have (X'WX)B=X'Wy, B (beta) acts as x, with A and b as follows.

     #Finding B (beta_hat) in equation
     A = (t(X) * diag(W)) %*% X  #Efficient way of A = t(X) %*% W %*% X bc W diag matrix.
                                 #Avoids mult by 0's.
     b = t(X) %*% W %*% y   #b = X'Wy

     #Obtain LU matrix decomposition of A.
     decomp <- lu(A)           #Calculates matrix decomposition object, containing each part.
     L <- expand(decomp)$L     #Upper triangular matrix
     U <- expand(decomp)$U     #Lower triangular lower triangular matrix
          #Note: Uses partial pivoting.  $P shows pivot matrix.

     #Now we replace Ax=b with LUx=b.
     #Introduce Ld=b, giving us two linear equation systems: Ld=b and Ux=d.
     #So we will solve in two steps.

     #1. Solve Ld=b for d.  This is d=inv(L)b
     d <- solve(L) %*% b

     #2. Substitutde d into Ux=d to solve for x.  This is x = inv(U)d. (Recall x=beta_hat)
     B_hat_LU <- solve(U) %*% d

     return(B_hat_LU)          #Returns function output.
}

#--------------------------------------------------
```

Performance benchmarking output is below. The inverse method was fastest for very small N and P values, but as N and P increased, LU and Cholesky performed more quickly than inverse. LU was the fastest method of the three.

```
> perf_results
$`N=10,P=5`
Unit: milliseconds
                   expr      min       lq      mean    median        uq      max neval cld
     inv_method(X, W, y) 0.036420 0.043232 0.0577904 0.0467835 0.0570515 0.274707   100  a
      lu_method(X, W, y) 0.096052 0.104844 0.1387691 0.1117485 0.1281640 0.367469   100   b
 cholesky_method(X, W, y) 0.082762 0.094476 0.1261794 0.1028505 0.1187925 0.446454   100   b

$`N=100,P=50`
Unit: milliseconds
                   expr      min       lq      mean   median       uq      max neval cld
     inv_method(X, W, y) 1.341538 1.3554330 1.4766650 1.417177 1.476745 2.775517   100   c
      lu_method(X, W, y) 0.730991 0.7500990 0.8728723 0.789655 0.864949 4.991523   100 a
 cholesky_method(X, W, y) 0.957423 0.9747275 1.0632554 1.021463 1.094125 1.845775   100  b

$`N=500,P=250`
Unit: milliseconds
                   expr       min        lq      mean    median        uq       max neval cld
     inv_method(X, W, y) 142.42425 149.67348 153.49734 153.42562 156.85662 166.61408   100   c
      lu_method(X, W, y)  65.41487  70.40461  72.42133  72.19728  74.69622  80.13181   100 a
 cholesky_method(X, W, y)  90.09651  96.49447  99.59850  98.19492 101.04675 185.89610   100  b

$`N=1500,P=500`
Unit: milliseconds
                   expr       min        lq      mean    median        uq       max neval cld
     inv_method(X, W, y) 1125.1834 1205.3996 1227.0725 1221.7252 1239.6840 1365.2525   100   c
      lu_method(X, W, y)  518.6956  556.4815  576.5880  565.1607  581.3506  696.1063   100 a
 cholesky_method(X, W, y)  720.8321  768.2105  786.0076  778.0312  793.0973  891.6376   100  b
```

Part D)

Since both LU and Cholesky are good for sparse matrices, I benchmarked both of these methods against the inverse method for a sparse matrix X.

I performed two types of benchmarking:

a) Benchmarking various N and P at 10% sparse.
b) Benchmarking various sparsity levels (5%, 10%, 20%, 50%).

For benchmarking at various N and P levels, results were similar to the results above. Inverse was superior for small N and P, and as N and P increased, Cholesky and LU were superior. LU again performed the most efficiently.

```
> perf_results
$`N=10,P=5`
Unit: microseconds
                    expr     min       lq     mean   median       uq     max neval cld
      inv_method(X, W, y) 199.747 205.3405 219.3008 211.5675 221.3445 584.565   100  b
       lu_method(X, W, y) 175.178 188.2930 204.0321 197.6335 210.0340 351.863   100 a
 cholesky_method(X, W, y) 212.990 220.4750 239.1594 230.0740 251.6420 366.284   100   c

$`N=100,P=50`
Unit: microseconds
                    expr      min       lq      mean    median        uq      max neval cld
      inv_method(X, W, y) 1341.974 1353.572 1440.3563 1375.5145 1507.0005 2167.188   100   c
       lu_method(X, W, y)  728.932  743.088  861.1276  759.1715  818.0605 2596.278   100 a
 cholesky_method(X, W, y) 1017.767 1033.113 1107.0747 1066.5340 1139.0410 1630.407   100  b

$`N=500,P=250`
Unit: milliseconds
                    expr       min        lq      mean    median        uq      max neval cld
      inv_method(X, W, y) 142.55163 149.27206 152.71361 152.47028 155.55658 162.7983   100   c
       lu_method(X, W, y)  63.94104  68.51884  72.07768  71.17862  74.10243 160.6241   100 a
 cholesky_method(X, W, y)  90.94583  95.85202  99.40865  98.41901 101.29379 188.9986   100  b

$`N=1000,P=500`
Unit: milliseconds
                    expr       min        lq      mean    median        uq       max neval cld
      inv_method(X, W, y) 1157.3838 1206.5452 1256.8446 1224.7319 1274.6652 2227.2327   100   c
       lu_method(X, W, y)  536.5109  555.3277  589.1192  568.2743  617.8695  805.6736   100 a
 cholesky_method(X, W, y)  731.4014  770.5118  817.5885  797.1329  855.3310 1028.9628   100  b
```

For benchmarking at various sparsity levels (with N=100, P=50 for all levels), the LU method performed most efficiently again, followed by Cholesky. For all methods, performance slowed as the matrix became less sparse.

```
> perf_results
$`5%`
Unit: microseconds
                    expr      min        lq      mean   median        uq      max neval cld
      inv_method(X, W, y) 1328.213 1343.8160 1443.5275 1361.141 1439.2770 3886.004   100   c
       lu_method(X, W, y)  712.385  729.1105  812.8257  746.409  781.6665 2516.260   100 a
 cholesky_method(X, W, y)  965.342  987.4815 1060.1466 1009.776 1065.3850 1882.526   100  b

$`10%`
Unit: microseconds
                    expr      min       lq     mean    median       uq      max neval cld
      inv_method(X, W, y) 1341.015 1350.375 1428.181 1362.9345 1458.950 2770.475   100   c
       lu_method(X, W, y)  723.338  739.680  810.809  756.7495  810.918 2066.349   100 a
 cholesky_method(X, W, y)  986.939 1003.977 1094.834 1034.6690 1153.302 1852.951   100  b

$`25%`
Unit: microseconds
                    expr      min        lq      mean    median       uq      max neval cld
      inv_method(X, W, y) 1347.748 1358.5250 1453.4610 1384.9700 1500.861 2490.823   100   c
       lu_method(X, W, y)  727.179  738.1875  817.3124  763.9665  851.536 2087.283   100 a
 cholesky_method(X, W, y)  956.943  970.2995 1102.2511  991.2705 1118.192 3194.545   100  b

$`50%`
Unit: microseconds
                    expr      min       lq      mean    median        uq      max neval cld
      inv_method(X, W, y) 1341.290 1351.774 1445.6323 1373.4890 1464.0255 3315.878   100   c
       lu_method(X, W, y)  728.378  737.150  800.2568  754.5095  796.3615 2269.798   100 a
 cholesky_method(X, W, y)  955.476  966.563 1016.5325  988.4150 1042.2260 1375.619   100  b
```