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) * diag(W)) %*% y     #b'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) * diag(W)) %*% y     #b'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) 0.037212 0.040238 0.05936496 0.0450785 0.0569075 0.271046   100   a
       lu_method(X, W, y) 0.093671 0.099491 0.20882553 0.1033555 0.1091390 8.197705   100   a
 cholesky_method(X, W, y) 0.083947 0.092698 0.11715468 0.0975250 0.1090805 0.387297   100   a

$`N=100,P=50`
Unit: milliseconds
                  expr      min        lq      mean    median        uq      max neval cld
      inv_method(X, W, y) 1.341063 1.3678615 1.5117139 1.4564635 1.5214265 2.513666   100   c
       lu_method(X, W, y) 0.390543 0.4061115 0.5049025 0.4599225 0.5322970 1.659894   100 a
 cholesky_method(X, W, y) 0.621523 0.6479115 0.8298841 0.7355420 0.7939465 4.908503   100  b

$`N=500,P=250`
Unit: milliseconds
                  expr       min        lq      mean    median        uq       max neval cld
      inv_method(X, W, y) 138.90659 145.22391 155.94202 149.10521 154.35837 239.12252   100   c
       lu_method(X, W, y)  27.34140  29.40955  32.22681  30.77907  32.97846  58.51686   100 a
 cholesky_method(X, W, y)  52.52199  54.83336  59.69976  57.26330  59.70452 148.27444   100  b

$`N=1500,P=500`
Unit: milliseconds
                  expr       min        lq      mean    median        uq       max neval cld
      inv_method(X, W, y) 1174.8760 1206.1232 1274.3722 1256.3489 1302.6356 1555.0684   100   c
       lu_method(X, W, y)  222.5136  236.7374  265.0399  247.9875  270.4645  570.9992   100 a
 cholesky_method(X, W, y)  428.3355  443.2046  476.3716  457.0468  496.0194  707.8721   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   #Display benchmarking results.
$`N=10,P=5`
Unit: microseconds
                  expr     min      lq     mean   median       uq     max neval cld
      inv_method(X, W, y) 197.114 206.778 236.1316 219.7015 242.5755 468.736   100   b
       lu_method(X, W, y) 143.408 157.140 194.8048 168.2205 193.5445 713.170   100  a
 cholesky_method(X, W, y) 182.562 197.844 249.8120 210.9740 235.8355 858.560   100   b

$`N=100,P=50`
Unit: microseconds
                  expr      min       lq      mean   median        uq      max neval cld
      inv_method(X, W, y) 1341.837 1364.188 1497.3468 1445.077 1516.0435 2395.601   100   c
       lu_method(X, W, y)  390.190  417.544  565.8643  479.133  526.2605 7247.968   100 a
 cholesky_method(X, W, y)  620.480  637.720  760.6811  696.628  751.4090 4567.732   100  b

$`N=500,P=250`
Unit: milliseconds
                  expr       min        lq      mean    median        uq       max neval cld
      inv_method(X, W, y) 140.63455 143.62599 148.22453 145.93308 149.74304 240.68631   100   c
       lu_method(X, W, y)  27.16558  29.65238  30.93671  30.41549  32.46541  36.01050   100 a
 cholesky_method(X, W, y)  52.70320  55.31940  56.99092  56.39157  58.54747  66.72288   100  b

$`N=1000,P=500`
Unit: milliseconds
                  expr       min        lq      mean    median        uq       max neval cld
      inv_method(X, W, y) 1126.7305 1182.8251 1218.8713 1196.6185 1235.5535 1635.3733   100   c
       lu_method(X, W, y)  217.9616  225.8616  243.1317  232.4415  238.2647  337.5194   100 a
 cholesky_method(X, W, y)  421.1639  435.7663  452.5527  442.5019  450.5010  565.0799   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) 1332.446 1353.585 1473.2416 1375.6350 1487.5065 2581.711   100   c
       lu_method(X, W, y)  382.151  402.017  506.2568  414.5310  459.1020 4306.896   100 a
 cholesky_method(X, W, y)  663.337  680.102  832.3181  702.2065  771.8245 9082.253   100  b

$`10%`
Unit: microseconds
                     expr      min        lq      mean    median       uq      max neval cld
      inv_method(X, W, y) 1340.080 1362.8130 1554.1333 1436.9385 1564.785 6372.561   100   c
       lu_method(X, W, y)  390.401  417.8465  471.6156  444.4730  500.706  774.812   100 a
 cholesky_method(X, W, y)  626.050  636.8025  704.9477  658.9095  725.636 1407.048   100  b

$`25%`
Unit: microseconds
                     expr      min       lq      mean    median       uq      max neval cld
      inv_method(X, W, y) 1342.764 1359.093 1420.0282 1376.7410 1414.642 2105.746   100   c
       lu_method(X, W, y)  394.281  411.026  467.8431  425.5870  464.818 1205.806   100 a
 cholesky_method(X, W, y)  626.638  645.232  747.6545  660.9175  718.165 4705.607   100  b

$`50%`
Unit: microseconds
                     expr      min        lq      mean    median        uq      max neval cld
      inv_method(X, W, y) 1339.807 1359.8605 1424.0768 1383.7105 1443.6945 2237.184   100   c
       lu_method(X, W, y)  389.093  417.0355  519.0709  437.3550  468.4490 4794.174   100 a
 cholesky_method(X, W, y)  625.482  641.2865  701.4168  656.7395  701.5745 1407.299   100  b
```

Part 2. Generalized Linear Regression

PART A)

The negative log-likelihood is:

$$l(\beta) = -\log \{\}$$

The gradient is: