Linear Regression

(A) Rewrite the WLS objective functions in terms of vectors and matrices as follows:

$$\hat{\beta}=\arg\min_{\beta\in R^P}\sum_{i=1}^{N}\frac{w_i}{2}(y_i - x_i^T\beta)^2 = \arg\frac{1}{2}\min_{\beta\in R^P}(y-X\beta)^T W(y-X\beta) \quad (1)$$

Since W is a diagonal matrix of weights (so symmetric), then

$$\begin{aligned}
&(y-X\beta)^T W(y-X\beta)\\
&=(y^T-\beta^T X^T)W(y-X\beta)\\
&=y^T Wy - y^T X\beta - \beta^T X^T Wy + \beta^T X^T WX\beta\\
&=y^T Wy - (X^T Wy)^T\beta - \beta^T X^T Wy + (X\beta)^T WX\beta
\end{aligned}$$

To get the minimum of the above equation, take derivative of β with the following formulas

$$\frac{\partial x^T a}{\partial x}=\frac{\partial a^T x}{\partial x}=a,\frac{\partial x^T bx}{\partial x}=(b+b^T)x$$

$$\Rightarrow \frac{\partial[\ (y-X\beta)^T W(y-X\beta)]}{\partial\beta}=-2X^T Wy+2X^T WX\beta=0$$

$$\Rightarrow (X^T WX)\hat{\beta}=X^T Wy$$

(B) Numerically speaking, I do not think inversion method is the fastest and most stable way to solve the linear system. Computing and applying the inverse matrix or pseudoinverse is a tremendously bad idea, since it is much more expensive and often numerically less stable than applying other algorithms, especially for high dimensional matrices.

Here are several ways to solve a system of linear equations Ax = b which provide more stability and are computationally efficient compared to inversion method.

(a) Matrix decomposition: Gaussian or Gauss-Jordan elimination (considered as LU decomposition), Cholesky decomposition, QR decomposition, or SVD, and
(b) Iterative method: conjugate gradient method.

Which method is optimal depends on the size and properties of the system matrix.

(a) LU requires A to be square and performs well when A is sparse; it can be used for most linear systems.
(b) Cholesky performs well for ***Hermitian positive-definite matrix*** A (A=LL*, where L is a lower triangular matrix with real and positive diagonal entries, and L* denotes the conjugate transpose of L)
(c) QR decomposition requires A has linearly independent columns.
(d) Conjugate gradient requires A to be symmetric and positive definite; it performs well when

A is sparse and too large to be inverted directly for Cholesky.

(C) Since $X^TWX = X^T(W^{1/2})^T W^{1/2} X \overset{set\ W^{1/2}=V}{=} (VX)^T VX$ is positive-definite, my method will be the Cholesky decomposition. To solve $(X^TWX)\hat{\beta} = X^TWy$, here is the pseudo-code:

Function inputs:
X: N x P matrix
Y: N x 1 vector of responses
W: N x N diagonal matrix of weights

Function outputs:

$\hat{\beta}$: P x 1 vector of coefficient estimates

Pseudo-code:

1) Set $A = X^TWX$;
2) Set $b = X^TWy$;
3) Set U = Cholesky decomposition of A. Only the upper triangular part of A is used in R so that $A = U^TU$ when A is symmetric.
4) Solve $U^Tz = b \Rightarrow z = (U^T)^{-1}b$;
5) Solve $Ux = z \Rightarrow x = U^{-1}z$;
6) Return $\hat{\beta}$ as $\hat{\beta} = x$.

From the following R output, the inverse method is fastest for very small N and P values, but as N and P increase, LU and Cholesky methods perform much more quickly than inverse. LU is the most efficient method of the three and it is a partial Gaussian elimination method.

```
> res # Display benchmarking results
$`N=20, P=5`
Unit: milliseconds
                  expr      min       lq      mean   median       uq       max neval
  inv_method(X, W, y) 0.075834 0.080395 0.1765681 0.084102 0.086668  4.808907   100
   lu_method(X, W, y) 0.161932 0.170484 0.3659880 0.179037 0.194717 17.948158   100
 chol_method(X, W, y) 0.160221 0.165923 0.2938887 0.175616 0.194717  9.791719   100

$`N=100, P=25`
Unit: milliseconds
                  expr      min       lq      mean    median        uq      max neval
  inv_method(X, W, y) 0.629480 0.662265 0.8013725 0.6987570 0.7557745 2.577788   100
   lu_method(X, W, y) 0.274827 0.287942 0.3764907 0.3312755 0.3646315 2.410726   100
 chol_method(X, W, y) 0.321582 0.346670 0.4112489 0.3703330 0.4079645 0.713297   100

$`N=600, P=150`
Unit: milliseconds
                  expr      min        lq      mean    median        uq      max neval
  inv_method(X, W, y) 98.66754 100.70594 103.65010 101.69606 102.99835 228.0673   100
   lu_method(X, W, y) 13.00013  13.45684  15.50683  13.83658  14.88543 141.9334   100
 chol_method(X, W, y) 16.78841  17.30043  18.00251  17.69015  18.69025  21.1731   100

$`N=2000, P=500`
Unit: milliseconds
                  expr       min        lq      mean    median        uq       max neval
  inv_method(X, W, y) 4114.8367 5645.3391 5603.0128 5721.7032 5837.2093 7025.8754   100
   lu_method(X, W, y)  492.1905  680.8708  699.6537  689.3546  713.6434  911.0215   100
 chol_method(X, W, y)  627.7260  872.8408  870.3740  886.8396  910.2068 1052.0467   100
```

Figure 1 Performance benchmarking results

solution_1c.R

Shuchen

Wed Sep 13 00:57:14 2017

```
# SDS385 Exercise 1 Part C
# compare various matrix decomposition methods with inversion method
# and benchmarks peformance of the Cholesky and LU vesus inversion

library ( Matrix )
library ( microbenchmark ) # For benchmarking
```

```
# Inversion method function
inv_method <- function (X,W,y){
  beta_hat <- solve (t(X) %*% W %*% X) %*% t(X) %*% W %*% y
  return (beta_hat)
}
```

```
# Cholesky decomposition
chol_method <- function (X,W,y){
  A = t(X) %*% (X*diag (W))   # Efficient way of A = t(X) %*% W %*% X as W is diagonal
  b = t(X) %*% (y*diag (W))   # Avoid multiply by 0's

  # Cholesky decomposition of A
  U <- chol(A)  # Upper Cholesky decomposition of A and U'U = A

  # Replace Ax=b with U'Ux=b, solve U'z=b for z
  z <- solve(t(U))%*%b
  # Solve Ux = z for x (x=beta_hat)
  beta_hat_chol <- solve(U) %*% z

  return(beta_hat_chol)
}
```

```
# LU method function
lu_method <- function (X,W,y){
  A = t(X) %*% (X*diag (W))   # Efficient way of A = t(X) %*% W %*% X as W is diagonal
  b = t(X) %*% (y*diag (W))   # Avoid multiply by 0's

  # LU decomposition of A
  decomp <- lu(A)
  L <- expand ( decomp )$L  # Upper triangular matrix
  U <- expand ( decomp )$U  # Lower triangular matrix

  # Replace Ax=b with LUx=b, solve Lz=b for z
  z <- solve (L) %*% b
  # Solve Ux=z for x (x=beta_hat)
  beta_hat_lu <- solve(U) %*% z

  return(beta_hat_lu)
}
```

```
# Simulate data from the linear model for a range of values of N and P
# Assume weights w_i are all 1, data are Gaussian

N <- c (20, 100, 400, 1200)
P <- N/4 # N>P

res <- list () # Performance results
for (i in 1: length (N)){
  n <- N[i]
  p <- P[i]
  print (n)

  # Set up matrices of size N, P parameters: (dummy data)
  X <- matrix ( rnorm (n*p),nrow =n, ncol =p)
  y <- rnorm (n)
  W <- diag (1, nrow =n)

  # Perform benchmarking:
  res[[i]] <- microbenchmark (
    inv_method (X,W,y),
    lu_method (X,W,y),
    chol_method (X,W,y),
    unit ='ms'
  )
}
```

```
## [1] 20
## [1] 100
## [1] 400
## [1] 1200
```

```
names(res) <- (c('N=20, P=5', 'N=100, P=25', 'N=400, P=100', 'N=1200, P=300'))

res # Display benchmarking results
```

```
## $`N=20, P=5`
## Unit: milliseconds
##                  expr      min        lq      mean   median       uq
##   inv_method(X, W, y) 0.076404 0.0798255 0.1334967 0.083532 0.089233
##    lu_method(X, W, y) 0.163642 0.1687740 0.3182182 0.173336 0.197283
##  chol_method(X, W, y) 0.157941 0.1625010 0.2971386 0.167633 0.188160
##        max neval
##   4.678906   100
##  12.778897   100
##   9.303074   100
##
## $`N=100, P=25`
## Unit: milliseconds
##                  expr      min        lq      mean    median        uq
##   inv_method(X, W, y) 0.628910 0.6788010 0.7098014 0.6899190 0.7058845
##    lu_method(X, W, y) 0.269125 0.3030510 0.3616489 0.3204420 0.3572185
##  chol_method(X, W, y) 0.317021 0.3549375 0.4032436 0.3680515 0.3894340
##       max neval
##  1.276065   100
##  2.023002   100
##  2.097696   100
##
## $`N=400, P=100`
## Unit: milliseconds
##                  expr       min        lq      mean    median        uq
##   inv_method(X, W, y) 31.444913 32.104898 34.481651 32.892888 33.919784
##    lu_method(X, W, y)  4.365306  4.629015  5.098359  4.819455  5.217442
##  chol_method(X, W, y)  5.685845  5.927887  6.412872  6.148548  6.615526
##         max neval
##  158.224091   100
##    8.047535   100
##    9.385750   100
##
## $`N=1200, P=300`
## Unit: milliseconds
##                  expr      min       lq      mean    median        uq
##   inv_method(X, W, y) 844.7061 986.6886 1122.2103 1196.6246 1209.0617
##    lu_method(X, W, y) 106.2886 114.4154  145.2011  151.0524  153.7698
##  chol_method(X, W, y) 134.2309 139.3554  176.9943  192.2784  195.7186
##        max neval
##  1344.5684   100
##   297.3477   100
##   337.5027   100
```

```
# SDS385 Exercise 1 Part D ????????Still working on it

library(Matrix)

N <- 10000
P <- N/4

# simulate a sparse matrix X
X = matrix(rnorm(N*P), nrow=N)
mask = matrix(rbinom(N*P,1,0.05), nrow=N)
X = mask*X
X[1:10, 1:10] # quick visual check
```

```
##       [,1] [,2] [,3]      [,4] [,5]     [,6] [,7]       [,8] [,9] [,10]
##  [1,]    0    0    0 3.0864389    0 0.000000    0  0.0000000    0     0
##  [2,]    0    0    0 0.0000000    0 0.000000    0  0.0000000    0     0
##  [3,]    0    0    0 0.0000000    0 0.000000    0 -0.9737562    0     0
##  [4,]    0    0    0 0.8790703    0 0.000000    0  0.0000000    0     0
##  [5,]    0    0    0 0.0000000    0 0.000000    0  0.0000000    0     0
##  [6,]    0    0    0 0.0000000    0 0.000000    0  0.0000000    0     0
##  [7,]    0    0    0 0.0000000    0 0.000000    0  0.0000000    0     0
##  [8,]    0    0    0 0.0000000    0 1.892181    0  0.0000000    0     0
##  [9,]    0    0    0 0.0000000    0 0.000000    0  0.0000000    0     0
## [10,]    0    0    0 0.0000000    0 0.000000    0  0.0000000    0     0
```

```
# Or X <- Matrix(data=rnorm(N*P)*rbinom(N*P, 1, 0.01), nrow=N, sparse=TRUE)
```

Generalized Linear Regression

(A) The negative log likelihood is

$$l(\beta) = -\log\{\prod_{i=1}^{N} p(y_i \mid \beta)\} = -\log\{\prod_{i=1}^{N} \binom{m_i}{y_i} w_i^{y_i} (1-w_i)^{m_i - y_i}\}$$

$$= -\sum_{i=1}^{N} \log\{\binom{m_i}{y_i} w_i^{y_i} (1-w_i)^{m_i - y_i}\}$$

$$= -\sum_{i=1}^{N} \{\log\binom{m_i}{y_i} + y_i \log(w_i) + (m_i - y_i)\log(1-w_i)\}$$

With $w_i = \dfrac{1}{1+\exp(-x_i^T\beta)} \Rightarrow \exp(-x_i^T\beta) = \dfrac{1}{w_i} - 1$

Taking derivative of $l(\beta)$ w.r.t β :

$$\nabla l(\beta) = -\sum_{i=1}^{N} \left(\frac{y_i}{w_i}\frac{\partial w_i}{\partial \beta} - \frac{m_i - y_i}{1-w_i}\frac{\partial w_i}{\partial \beta}\right)$$

Since $\dfrac{\partial w_i}{\partial \beta} = -(1+\exp(-x_i^T\beta))^{-2}\dfrac{\partial}{\partial \beta}\exp(-x_i^T\beta)$

$$= \frac{-\exp(-x_i^T\beta)(-x_i)}{(1+\exp(-x_i^T\beta))^2} = \frac{x_i\exp(-x_i^T\beta)}{(1+\exp(-x_i^T\beta))^2} = w_i^2 x_i \left(\frac{1}{w_i} - 1\right) = w_i x_i (1-w_i)$$

$$\Rightarrow \nabla l(\beta) = -\sum_{i=1}^{N} \left(\frac{y_i}{w_i} w_i x_i (1-w_i) - \frac{m_i - y_i}{1-w_i} w_i x_i (1-w_i)\right)$$

$$= -\sum_{i=1}^{N} (y_i x_i (1-w_i) - (m_i - y_i) w_i x_i) = -\sum_{i=1}^{N} (y_i - m_i w_i) x_i$$

$$\overset{\text{In matrix form}}{=} -X^T(y - mw) = X^T(mw - y)$$