Linear Regression

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

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

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 $X^TWX = X^T(W^{1/2})^TW^{1/2}X \overset{set\ W^{1/2}=V}{=} (VX)^TVX$ 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.

```
$`N=20, P=5`
Unit: milliseconds
                 expr      min        lq      mean    median       uq       max neval
   inv_method(X, W, y) 0.081536 0.0849575 0.1931034 0.0892335 0.091799  5.458343   100
    lu_method(X, W, y) 0.177897 0.1818880 0.3844447 0.1898705 0.195287 18.898650   100
  chol_method(X, W, y) 0.172195 0.1778970 0.3056914 0.1830280 0.193292 10.990810   100

$`N=100, P=25`
Unit: milliseconds
                 expr      min        lq      mean   median       uq      max neval
   inv_method(X, W, y) 0.625489 0.6323310 0.6990878 0.679086 0.689349 2.409014   100
    lu_method(X, W, y) 0.271977 0.2845205 0.3290688 0.315025 0.346385 0.679086   100
  chol_method(X, W, y) 0.319301 0.3355515 0.3833101 0.367482 0.380596 2.109670   100

$`N=400, P=100`
Unit: milliseconds
                 expr       min        lq      mean    median        uq        max neval
   inv_method(X, W, y) 30.104417 30.618435 33.188846 31.425812 32.949621 157.426978   100
    lu_method(X, W, y)  4.234735  4.344780  4.814803  4.610484  4.897285   7.127263   100
  chol_method(X, W, y)  5.469176  5.583782  6.206546  5.826680  6.420238   8.372538   100

$`N=1200, P=300`
Unit: milliseconds
                 expr       min       lq     mean   median       uq       max neval
   inv_method(X, W, y) 771.87970 794.7408 885.7643 819.2258 878.1566 1739.3503   100
    lu_method(X, W, y)  97.14744 100.9289 122.3051 104.6884 113.5510  281.4505   100
  chol_method(X, W, y) 122.67388 128.7483 148.1157 133.4865 140.8815  322.5651   100
```

Figure 1: Performance benchmarking results

(D) Since both LU and Cholesky perform well for sparse matrices, we benchmarked both of these methods and sparse Cholesky decomposition against inverse method for a sparse matrix with various sparsity level (1%, 5%, 25%). Here, *theta* represents the density of X, i.e., the proportion of entries which are non-zero. In the benchmark below, we can see more noticeable efficiency increase with higher sparsity and LU again performed the most efficiently.

```
$`theta=0.01`
Unit: milliseconds
                  expr        min         lq      mean    median        uq       max neval
    inv_method(X, W, y) 12853.4133 14131.1496 15649.875 16316.100 16981.361 17780.898    10
     lu_method(X, W, y)  2746.3071  2832.3491  3188.309  3182.086  3551.914  3700.431    10
   chol_method(X, W, y)  3326.3916  3956.6246  4783.463  4965.668  5419.242  6048.513    10
 sparse_method(X, W, y)   682.8665   822.4416  1046.696  1036.324  1190.962  1570.341    10

$`theta=0.05`
Unit: milliseconds
                  expr        min         lq       mean     median       uq       max neval
    inv_method(X, W, y) 11427.7201 14825.9745 15919.8157 16308.1549 17487.184 19604.344    10
     lu_method(X, W, y)  3183.9826  3483.1195  3573.6130  3548.6364  3695.268  3954.929    10
   chol_method(X, W, y)  3827.7877  4200.4905  4823.9148  4891.4851  5404.459  5574.472    10
 sparse_method(X, W, y)   805.8619   826.8298   977.1668   967.8372  1074.542  1215.062    10

$`theta=0.25`
Unit: milliseconds
                  expr        min         lq      mean    median        uq       max neval
    inv_method(X, W, y) 13071.8719 15303.1995 15945.594 16223.690 16977.776 17819.406    10
     lu_method(X, W, y)  3014.7278  3067.4558  3395.305  3287.320  3604.779  4155.951    10
   chol_method(X, W, y)  4581.1142  4645.9313  5255.740  5144.875  5574.287  6454.916    10
 sparse_method(X, W, y)   697.6045   951.5397  1046.293  1091.212  1146.796  1240.047    10
```

Figure 2: Benchmarking for various values of N, P, and density level

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 :

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

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

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

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

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

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

(C) First, we need to calculate the Hessian matrix of log likelihood function $\nabla^2 l(\beta)$, which is a key part of Taylor series expansion.

From (A), we have $\nabla l(\beta) = -\sum_{i=1}^{N} (y_i - m_i w_i) x_i = -X^T(y - mw)$ and $\dfrac{\partial w_i}{\partial \beta} = x_i w_i (1-w_i)$

$$\Rightarrow \nabla^2 l(\beta) = -\frac{\partial}{\partial \beta}\sum_{i=1}^{N} (y_i - m_i w_i) x_i = \sum_{i=1}^{N} m_i x_i \frac{\partial w_i}{\partial \beta} = \sum_{i=1}^{N} m_i x_i x_i w_i (1-w_i)$$

Written in matrix form, $\nabla^2 l(\beta) = X^T W X$

where $W = diag[m_1 w_1(1-w_1),...,m_N w_N(1-w_N)]$

Let $v = y - mw$, then $\nabla l(\beta) = -X^T v$

Recall Taylor series second-order expansion in general form:

$$q(x;a) = f(a) + h(a)^T(x-a) + \frac{1}{2}(x-a)^T H(a)(x-a)$$

Where, $h(a)$ gradient evaluated at point a.

$H(a)$ Hessian evaluated at point a.

Then, we have $f(x) = c + bx + \frac{1}{2}x^T ax = \frac{1}{2}(x^T ax + 2bx + c) = \frac{1}{2}(x-u)^T a(x-u)$

$$= \frac{1}{2}(x^T ax - 2u^T ax + u^T au)$$

$$\Rightarrow b = -u^T a, c = \frac{1}{2}u^T au \Rightarrow u = -(ba^{-1})^T, c = \frac{1}{2}b(a^{-1})^T b^T$$

So $q(\beta;\beta_0) = l(\beta_0) + (-X^T v)^T(\beta - \beta_0) + \frac{1}{2}(\beta-\beta_0)^T X^T WX(\beta-\beta_0)$

$$= \frac{1}{2}[2c - 2v^T X(\beta-\beta_0) + (X(\beta-\beta_0))^T WX(\beta-\beta_0)] + l(\beta_0) - c$$

$$= \frac{1}{2}[(u - X(\beta-\beta_0))^T W((u - X(\beta-\beta_0))] + l(\beta_0) - c$$

Where $u = -(-v^T W^{-1})^T = W^{-1}v, c = \frac{1}{2}(-v^T)(W^{-1})^T(-v^T)^T = \frac{1}{2}v^T W^{-1}v$

$$\Rightarrow q(\beta;\beta_0) = \frac{1}{2}[(W^{-1}v + X\beta_0 - X\beta)^T W(W^{-1}v + X\beta_0 - X\beta)] + l(\beta_0) - c$$

$$= \frac{1}{2}[(z - X\beta)^T W(z - X\beta)] + c^*$$

Where

$$z = W^{-1}v + X\beta_0 = W^{-1}(y - mw) + X\beta_0$$

$$c^* = l(\beta_0) - c = l(\beta_0) - \frac{1}{2}v^T W^{-1}v = l(\beta_0) - \frac{1}{2}(y-mw)^T W^{-1}(y-mw)$$

c* is a constant that doesn't involve β.

(D) Here we use Newton's method to estimate. This iterative process requires far fewer iterations to achieve convergence since we are taking the curvature of the objective function $l(\beta)$ into account. We actually only use 10 iterations and achieve estimates which are exactly in line with estimates from ***glm***.

Newton's Method: $\hat{\beta}_{t+1} = \hat{\beta}_t - (\nabla^2 l(\hat{\beta}_t))^{-1} \nabla l(\hat{\beta}_t)$

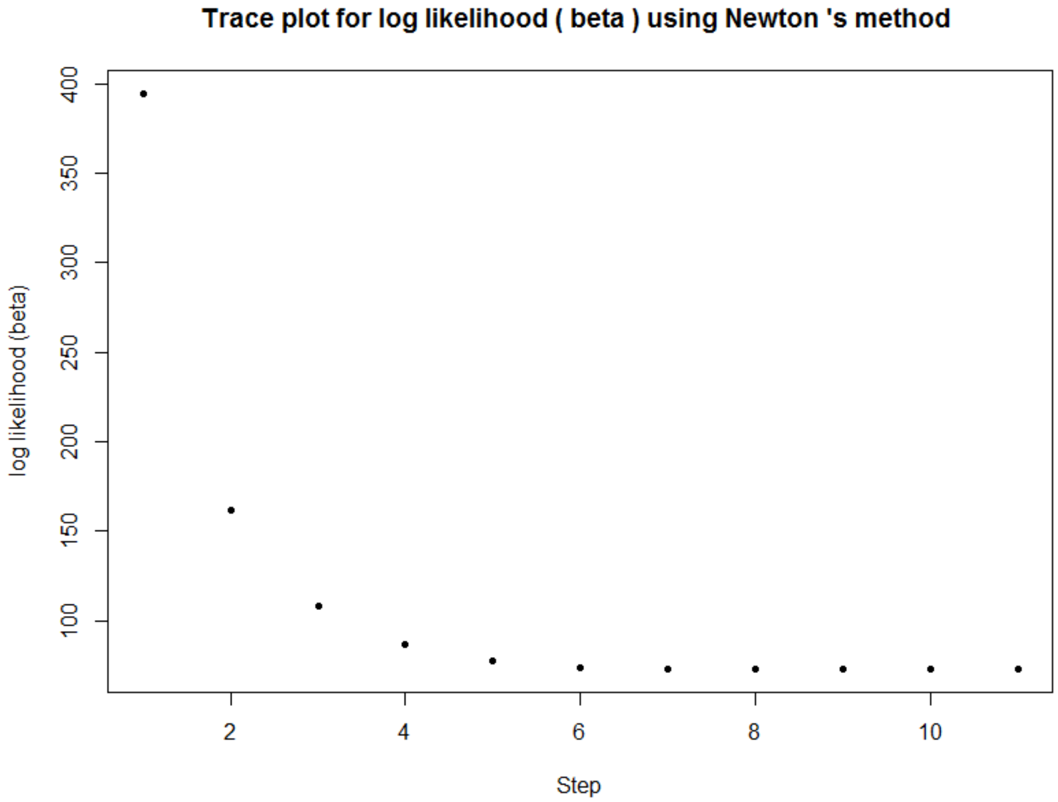

Figure 3: Log likelihood trace plot for Newton's method

	Newton's method		R: glm
	0.48701675	[1,]	0.48553491
V3	-7.22185053	[2,]	-7.14617798
V4	1.65475615	[3,]	1.65480926
V5	-1.73763027	[4,]	-1.80712669
V6	14.00484560	[5,]	13.99289624
V7	1.07495329	[6,]	1.07426088
V8	-0.07723455	[7,]	-0.07318783
V9	0.67512313	[8,]	0.67573353
V10	2.59287426	[9,]	2.59382838
V11	0.44625631	[10,]	0.44615349
V12	-0.48248420	[11,]	-0.48275720

Figure 4: Comparison of results from Newton's method and glm