Exercises 1: Preliminaries

1 Linear regression

Consider the simple linear regression model

$$y = X\beta + e\,,$$

where $y = (y_1, \ldots, y_N)$ is an N-vector of responses, X is an $N \times P$ matrix of features whose ith row is x_i, and e is a vector of model residuals. The goal is to estimate β, the unknown P-vector of regression coefficients.

Let's say you trust the precision of some observations more than others, and therefore decide to estimate β by the principle of weighted least squares (WLS):

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

where w_i is the weight for observation i.

(A) Rewrite the WLS objective function above in terms of vectors and matrices, and show that $\hat{\beta}$ is the solution to the following linear system of P equations in P unknowns:

$$(X^T W X)\hat{\beta} = X^T W y\,,$$

where W is the diagonal matrix of weights.

Let

$$\mathbf{W} = \begin{pmatrix} w_1 & 0 & \cdots & 0 \\ 0 & w_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & w_n \end{pmatrix}$$

Then,

$$\arg\min_{\beta \in \mathcal{R}^P} \left\{ \sum_{i=1}^{n} w_i(y_i - \mathbf{x_i}^T\boldsymbol{\beta})^2 \right\} = \arg\min_{\beta \in \mathcal{R}^P} \left\{ (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T \mathbf{W} (\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) \right\}.$$

Take the derivative and set it equal to zero,

$$\begin{aligned}\frac{\delta}{\delta\boldsymbol{\beta}}\left((\mathbf{y}-\mathbf{X}\boldsymbol{\beta})^T\mathbf{W}(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})\right) &= 0\\ -2\mathbf{X}^T\mathbf{W}\left(\mathbf{y}-\mathbf{X}\boldsymbol{\beta}\right) &= 0\\ -2\mathbf{X}^T\mathbf{W}\mathbf{y}+2\mathbf{X}^T\mathbf{W}\mathbf{X}\boldsymbol{\beta} &= 0\\ \mathbf{X}^T\mathbf{W}\mathbf{X}\boldsymbol{\beta} &= \mathbf{X}^T\mathbf{W}\mathbf{y}\end{aligned}$$

The second derivative is $2\mathbf{X}^T\mathbf{W}\mathbf{X}$, which is positive, so this is a minimum, making $\hat{\beta}$ the solution to

$$\mathbf{X}^T\mathbf{W}\mathbf{X}\hat{\boldsymbol{\beta}} = \mathbf{X}^T\mathbf{W}\mathbf{y}$$

(B) One way to calculate $\hat{\beta}$ is to: (1) recognize that, trivially, the solution to the above linear system must satisfy $\hat{\beta} = (X^TWX)^{-1}X^TWy$; and (2) to calculate this directly, i.e. by inverting X^TWX. Let's call this the "inversion method" for calculating the WLS solution.

Numerically speaking, is the inversion method the fastest and most stable way to actually solve the above linear system?

Let $A = \mathbf{X}^T\mathbf{W}\mathbf{X}$ and $C = \mathbf{X}^T\mathbf{W}\mathbf{y}$, then we are trying to solve the linear system of equations: $A\hat{\boldsymbol{\beta}} = C$. Rather than taking the inverse of A directly, one can decompose A into a multiple of several matrices that are easier to work with. This is both faster and more stable. Below I present three common decompositions.

QR Decomposition

- Decompose A into Q and R, where $A = QR$; Q is an orthogonal matrix and R is an upper triangular matrix.Then let $Q\hat{\boldsymbol{\beta}} = Z$
- Solve the linear system of equations $QZ = C$. Since Q is orthogonal, its inverse is Q^T, so $Z = Q^TZ$.
- Then solve $R\hat{\boldsymbol{\beta}} = Z$ for $\hat{\boldsymbol{\beta}}$ using forward substitution.

LU Decomposition

- Decompose A into L and U, where $A = LU$; L is a lower triangular matrix with 1's on the diagonal and U is an upper triangular matrix. Then let $U\hat{\boldsymbol{\beta}} = Z$.
- Solve the linear system of equations $LZ = C$ for Z using forward substitution.
- Then solve $U\hat{\boldsymbol{\beta}} = Z$ for $\hat{\boldsymbol{\beta}}$ using backward substitution.

Cholesky Decomposition

- Decompose A into U and U^T, where $A = U^T U$; U is an upper triangular matrix. Then let $U\hat{\boldsymbol{\beta}} = Z$.
- Solve the linear system of equations $U^T Z = C$ for Z using forward substitution.
- Then solve $U\hat{\boldsymbol{\beta}} = Z$ for $\hat{\boldsymbol{\beta}}$ using backward substitution.

Of these matrix factorizations, Cholesky decomposition is the fastest, while QR decomposition is the most stable. By default, R uses QR decomposition.

(C) Code up functions that implement both the inversion method and your method for an arbitrary X, y, and weights W. Now simulate some data from the linear model for a range of values of N and P. Make sure that you explore values of P up into the thousands, and that $N > P$. Benchmark the performance of the inversion solver and your solver across a range of scenarios.

I compared the inversion method to a Cholesky decomposition and the default *solve* function in R. Below are the results of the performance comparison for randomly simulated data.

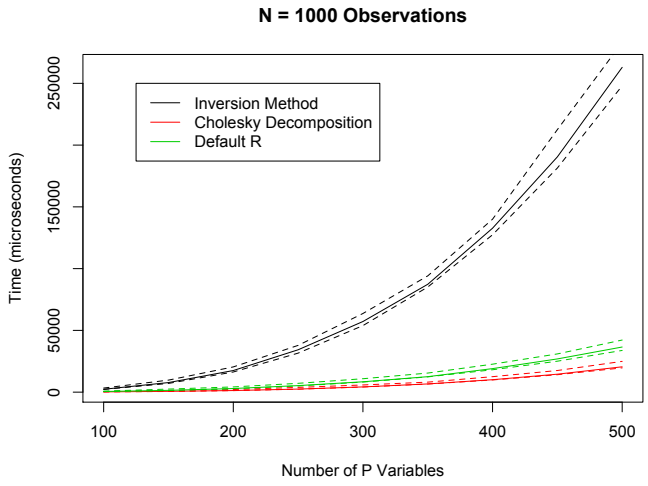

Figure 1: Processing Time of Different Inversion Algorithms for N=1000 Observations and Varying Numbers of Covariate, P

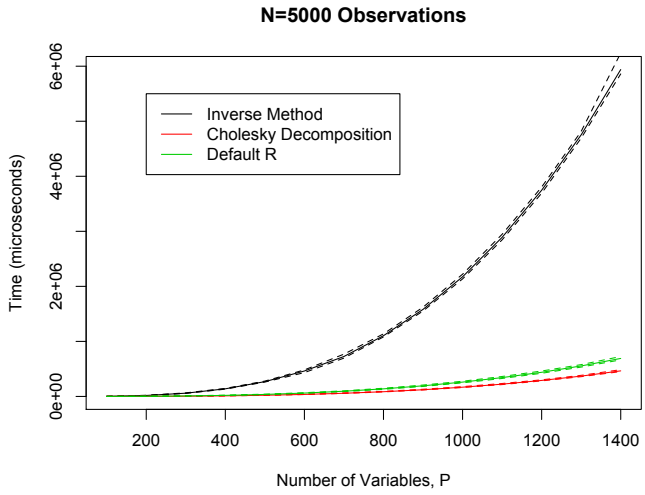

Figure 2: Processing Time of Different Inversion Algorithms for N=5000 Observations and Varying Numbers of Covariate, P

(D) Now what happens if X is a highly sparse matrix, in the sense that most entries are zero? Repeat the previous benchmarking exercise with this new recipe for simulating a sparse X, except add another solver to the mix: one that can solve a linear system $Ax = b$ in a way that exploits the sparsity of A. Benchmark the inversion method, your method, and the sparse method across different sparsity levels.

I used the *Matrix* package in R to store the sparse matrices in a special way that takes advantage of their sparsity. Then, all algorithms are run on the same matrix stored as a dense matrix and again stored as a sparse matrix. The performance results are below for varying levels of sparsity.

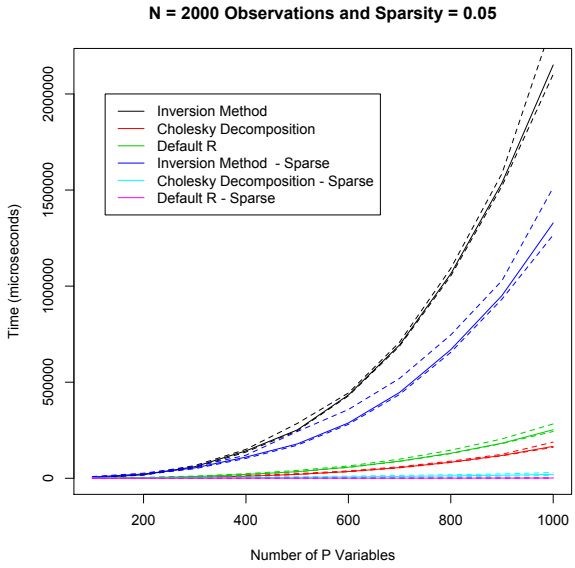

Figure 3: Processing Time of Different Inversion Algorithms for N=2000 Observations and 5% Sparsity for Varying Levels of Covariates, P

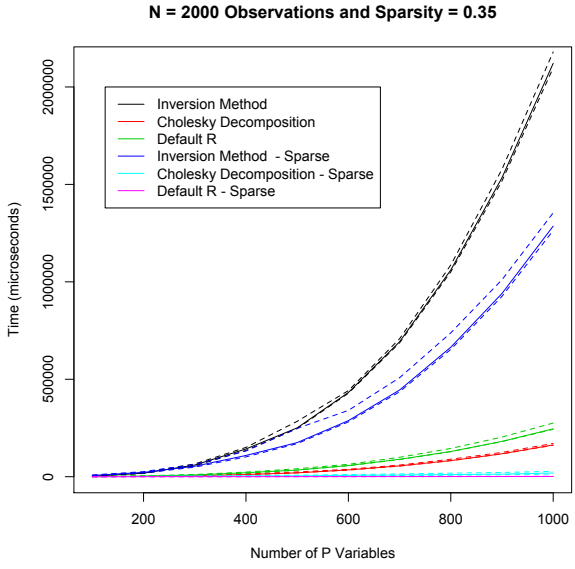

Figure 4: Processing Time of Different Inversion Algorithms for N=2000 Observations and 35% Sparsity for Varying Levels of Covariates, P

2 Generalized linear models

As an archetypal case of a GLM, we'll consider the binomial logistic regression model: $y_i \sim \text{Binomial}(m_i, w_i)$, where y_i in an integer number of "successes," m_i is the number of trials for the ith case, and the success probability w_i is a regression on a feature vector $\mathbf{x_i}$ given by the inverse logit transform:

$$w_i = \frac{1}{1 + \exp\{-\mathbf{x_i}^T \boldsymbol{\beta}\}} .$$

We want to estimate $\boldsymbol{\beta}$ by the principle of maximum likelihood. Note: for binary logistic regression, $m_i = 1$ and y_i is either 0 or 1.

(A) Start by writing out the negative log likelihood,

$$l(\boldsymbol{\beta}) = -\log \left\{ \prod_{i=1}^{N} p(y_i \mid \boldsymbol{\beta}) \right\} .$$

Simplify your expression as much as possible. This is the thing we want to minimize to compute the MLE. (By longstanding convention, we phrase optimization problems as minimization problems.)

Derive the gradient of this expression, $\nabla l(\boldsymbol{\beta})$. Note: your gradient will be a sum of terms $l_i(\boldsymbol{\beta})$, and it's OK to use the shorthand

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

in your expression.

If $y_i \sim \text{Binomial}(m_i, w_i(\boldsymbol{\beta}))$, then:

$$p(y_i \mid w_i(\boldsymbol{\beta}), m_i) = \frac{m_i!}{y_i!(m_i - y_i)!} w_i(\boldsymbol{\beta})^{y_i} (1 - w_i(\boldsymbol{\beta}))^{m_i - y_i} .$$

So:

$$\begin{aligned}
l(\boldsymbol{\beta}) &= -\log \left\{ \prod_{i=1}^{N} p(y_i \mid \boldsymbol{\beta}) \right\} \\
&= -\sum_{i=1}^{N} \log \{p(y_i \mid \boldsymbol{\beta})\} \\
&= -\sum_{i=1}^{N} \log \left\{ \frac{m_i!}{y_i!(m_i - y_i)!} w_i(\boldsymbol{\beta})^{y_i} (1 - w_i(\boldsymbol{\beta}))^{m_i - y_i} \right\} \\
&= -\sum_{i=1}^{N} \log \left\{ \frac{m_i!}{y_i!(m_i - y_i)!} \right\} - \sum_{i=1}^{N} y_i \log \{w_i(\boldsymbol{\beta})\} - \sum_{i=1}^{N} (m_i - y_i) \log \{1 - w_i(\boldsymbol{\beta})\}
\end{aligned}$$

Let $-\sum_{i=1}^{N} \log\left\{\frac{m_i!}{y_i!(m_i-y_i)!}\right\} = c$, then:

$$
\begin{aligned}
l(\boldsymbol{\beta}) &= c - \sum_{i=1}^{N} y_i \log\left\{\frac{1}{1+\exp\{-\mathbf{x_i}^T\boldsymbol{\beta}\}}\right\} - \sum_{i=1}^{N}(m_i - y_i)\log\left\{1 - \frac{1}{1+\exp\{-\mathbf{x_i}^T\boldsymbol{\beta}\}}\right\} \\
&= c - \sum_{i=1}^{N} y_i \log\left\{\frac{1}{1+\exp\{-\mathbf{x_i}^T\boldsymbol{\beta}\}}\right\} - \sum_{i=1}^{N}(m_i - y_i)\log\left\{\frac{\exp\{-\mathbf{x_i}^T\boldsymbol{\beta}\}}{1+\exp\{-\mathbf{x_i}^T\boldsymbol{\beta}\}}\right\} \\
&= c + \sum_{i=1}^{N} y_i \log\left\{1+\exp\{-\mathbf{x_i}^T\boldsymbol{\beta}\}\right\} + \sum_{i=1}^{N}(m_i - y_i)\mathbf{x_i}^T\boldsymbol{\beta} + \sum_{i=1}^{N}(m_i - y_i)\log\left\{1+\exp\{-\mathbf{x_i}^T\boldsymbol{\beta}\}\right\} \\
&= c + \sum_{i=1}^{N}(m_i - y_i)\mathbf{x_i}^T\boldsymbol{\beta} + \sum_{i=1}^{N} m_i \log\left\{1+\exp\{-\mathbf{x_i}^T\boldsymbol{\beta}\}\right\}
\end{aligned}
$$

The gradient is:

$$
\begin{aligned}
\nabla l(\boldsymbol{\beta}) &= \nabla\left\{c + (m_i - y_i)\mathbf{x_i}^T\boldsymbol{\beta} + \sum_{i=1}^{N} m_i \log\left\{1+\exp\{-\mathbf{x_i}^T\boldsymbol{\beta}\}\right\}\right\} \\
&= \left\{\sum_{i=1}^{N}(m_i - y_i)\mathbf{x_i} - \sum_{i=1}^{N} m_i \frac{\exp\{-\mathbf{x_i}^T\boldsymbol{\beta}\}}{1+\exp\{-\mathbf{x_i}^T\boldsymbol{\beta}\}}\mathbf{x_i}\right\} \\
&= \left\{\sum_{i=1}^{N}(m_i - y_i)\mathbf{x_i} - \sum_{i=1}^{N} m_i(1 - w_i(\boldsymbol{\beta}))\mathbf{x_i}\right\} \\
&= \left\{\sum_{i=1}^{N}(m_i w_i(\boldsymbol{\beta}) - y_i)\mathbf{x_i}\right\}
\end{aligned}
$$

(B) Write your own function that will fit a logistic regression model by gradient descent. Grab the data "wdbc.csv" from the course website. The WDBC file has information on 569 breast-cancer patients from a study done in Wisconsin. The first column is a patient ID, the second column is a classification of a breast cell (Malignant or Benign), and the next 30 columns are measurements computed from a digitized image of the cell nucleus. These are things like radius, smoothness, etc. For this problem, use the first 10 features for X, i.e. columns 3-12 of the file. If you use all 30 features you'll run into trouble.

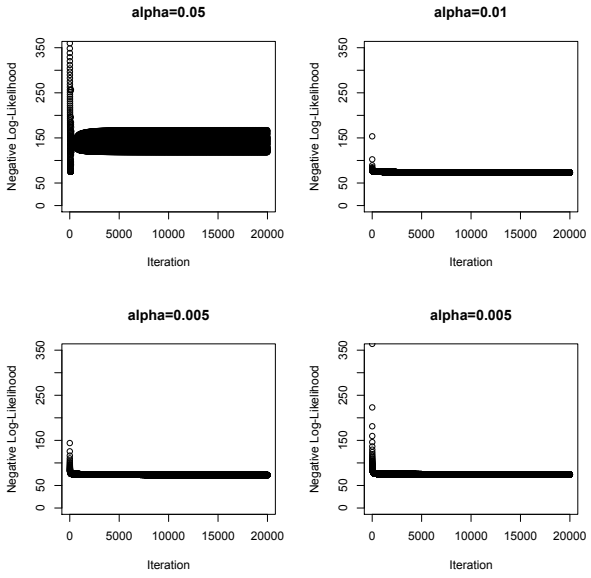

Figure 5: Convergence of Gradient Descent for Varying Levels of α

(C) Now consider a point $\beta_0 \in \mathcal{R}^P$, which serves as an intermediate guess for our vector of regression coefficients. Show that the second-order Taylor approximation of $l(\beta)$, around the point β_0, takes the form

$$q(\beta;\beta_0) = \frac{1}{2}(z - X\beta)^T V(z - X\beta) + c\,,$$

where z is a vector of "working responses" and W is a diagonal matrix of "working weights," and c is a constant that doesn't involve β. Give explicit expressions for the diagonal elements W_{ii} and for z_i (which will necessarily involve the point β_0, around which you're doing the expansion).

The second order Taylor series of a function, $f(x)$, around the point a is:

$$f(a) + f'(a)(x-a) + \frac{f''(a)}{2}(x-a)^2$$

So we want to find:

$$l(\boldsymbol{\beta_0}) + (\boldsymbol{\beta} - \boldsymbol{\beta_0})^T \nabla l(\boldsymbol{\beta_0}) + \frac{1}{2}(\boldsymbol{\beta} - \boldsymbol{\beta_0})^T \nabla^2 l(\boldsymbol{\beta_0})(\boldsymbol{\beta} - \boldsymbol{\beta_0})$$

Previously, we have shown that:

$$l(\boldsymbol{\beta}) = c + \sum_{i=1}^{N}(m_i - y_i)\mathbf{x_i}^T\boldsymbol{\beta} + \sum_{i=1}^{N} m_i \log\left\{1 + \exp\{-\mathbf{x_i}^T\boldsymbol{\beta}\}\right\}$$

and

$$\begin{aligned}\nabla l(\boldsymbol{\beta}) &= \left\{\sum_{i=1}^{N}(m_i w_i(\boldsymbol{\beta}) - y_i)\mathbf{x_i}\right\} \\ &= \mathbf{X}^T(\mathbf{MW1}_{N\times 1} - \mathbf{y}) \ .\end{aligned}$$

Where:

$$\mathbf{W} = \begin{pmatrix} w_1(\boldsymbol{\beta}) & 0 & \cdots & 0 \\ 0 & w_2(\boldsymbol{\beta}) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & w_N(\boldsymbol{\beta}) \end{pmatrix}$$

$$\mathbf{M}\begin{pmatrix} m_1 & 0 & \cdots & 0 \\ 0 & m_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & m_N \end{pmatrix} \ .$$

Next we must find $\nabla^2 l(\boldsymbol{\beta})$,

$$\begin{aligned}\nabla^2 l(\boldsymbol{\beta}) &= \nabla(\nabla l(\boldsymbol{\beta})) \\ &= \nabla\left\{\sum_{i=1}^{N}(m_i - y_i)\mathbf{x_i} - \sum_{i=1}^{N} m_i \frac{\exp\{-\mathbf{x_i}^T\boldsymbol{\beta}\}}{1+\exp\{-\mathbf{x_i}^T\boldsymbol{\beta}\}}\mathbf{x_i}\right\} \\ &= -\left\{\sum_{i=1}^{N} + m_i \frac{\exp\{-\mathbf{x_i}^T\boldsymbol{\beta}\}}{1+\exp\{-\mathbf{x_i}^T\boldsymbol{\beta}\}}\mathbf{x_i}\mathbf{x_i}^T + \sum_{i=1}^{N} - m_i \frac{\exp\{-\mathbf{x_i}^T\boldsymbol{\beta}\}^2}{(1+\exp\{-\mathbf{x_i}^T\boldsymbol{\beta}\})^2}\mathbf{x_i}\mathbf{x_i}^T\right\} \\ &= -\left\{\sum_{i=1}^{N}\mathbf{x_i}\mathbf{x_i}^T m_i \left(\frac{\exp\{-\mathbf{x_i}^T\boldsymbol{\beta}\}}{1+\exp\{-\mathbf{x_i}^T\boldsymbol{\beta}\}}\right)\left(1 - \frac{\exp\{-\mathbf{x_i}^T\boldsymbol{\beta}\}}{1+\exp\{-\mathbf{x_i}^T\boldsymbol{\beta}\}}\right)\right\} \\ &= -\left\{\sum_{i=1}^{N}\mathbf{x_i}\mathbf{x_i}^T m_i(1 - w_i(\boldsymbol{\beta}))w_i(\boldsymbol{\beta})\right\} \ . \\ &= -\mathbf{X}^T\mathbf{MW}(\mathbf{I}-\mathbf{W})\mathbf{X}\end{aligned}$$

Now we can compute the second order Taylor series approximation:

$$\begin{aligned}& l(\boldsymbol{\beta_0}) + (\boldsymbol{\beta} - \boldsymbol{\beta_0})^T\nabla l(\boldsymbol{\beta_0}) + \frac{1}{2}(\boldsymbol{\beta} - \boldsymbol{\beta_0})^T\nabla^2 l(\boldsymbol{\beta_0})(\boldsymbol{\beta} - \boldsymbol{\beta_0}) \\ &= l(\boldsymbol{\beta_0}) + \boldsymbol{\beta}^T\nabla l(\boldsymbol{\beta_0}) - \boldsymbol{\beta_0}^T\nabla l(\boldsymbol{\beta_0}) + \frac{1}{2}\left\{\boldsymbol{\beta}^T\nabla^2 l(\boldsymbol{\beta_0})\boldsymbol{\beta} - 2\boldsymbol{\beta}^T\nabla^2 l(\boldsymbol{\beta_0})\boldsymbol{\beta_0} + \boldsymbol{\beta_0}^T\nabla^2 l(\boldsymbol{\beta_0})\boldsymbol{\beta_0}\right\} \\ &= c + \boldsymbol{\beta}^T\left(\nabla l(\boldsymbol{\beta_0}) - \nabla^2 l(\boldsymbol{\beta_0})\boldsymbol{\beta_0}\right) + \frac{1}{2}\boldsymbol{\beta}^T\nabla^2 l(\boldsymbol{\beta_0})\boldsymbol{\beta}\end{aligned}$$

where $c = l(\boldsymbol{\beta_0}) - \boldsymbol{\beta_0}^T \nabla l(\boldsymbol{\beta_0}) + \frac{1}{2}\boldsymbol{\beta_0}^T \nabla^2 l(\boldsymbol{\beta_0})\boldsymbol{\beta_0}$,

$$= c + \boldsymbol{\beta}^T \left(\mathbf{X}^T(\mathbf{MW1}_{N\times 1} - \mathbf{y}) + \mathbf{X}^T\mathbf{MW}(\mathbf{I}-\mathbf{W})\mathbf{X})\boldsymbol{\beta_0}\right) - \frac{1}{2}\boldsymbol{\beta}^T\mathbf{X}^T\mathbf{MW}(\mathbf{I}-\mathbf{W})\mathbf{X}\boldsymbol{\beta}$$

$$= c + \boldsymbol{\beta}^T\mathbf{X}^T\left(\mathbf{MW1}_{N\times 1} - \mathbf{y} + \mathbf{MW}(\mathbf{I}-\mathbf{W})\mathbf{X}\boldsymbol{\beta_0}\right) - \frac{1}{2}\boldsymbol{\beta}^T\mathbf{X}^T\mathbf{MW}(\mathbf{I}-\mathbf{W})\mathbf{X}\boldsymbol{\beta}$$

$$= -\frac{1}{2}\left((\mathbf{I}-\mathbf{W})^{-1}(\mathbf{1}_{N\times 1} - \mathbf{y}) + \mathbf{X}\boldsymbol{\beta}_0 - \mathbf{X}\boldsymbol{\beta}\right)^T \mathbf{MW}(\mathbf{I}-\mathbf{W})\left((\mathbf{I}-\mathbf{W})^{-1}(\mathbf{1}_{N\times 1} - \mathbf{y}) + \mathbf{X}\boldsymbol{\beta}_0 - \mathbf{X}\boldsymbol{\beta}\right) + c$$

$$= -\frac{1}{2}\left(\mathbf{z} - \mathbf{X}\boldsymbol{\beta}\right)^T \mathbf{V}\left(\mathbf{z} - \mathbf{X}\boldsymbol{\beta}\right) + c'$$

where:

$$\mathbf{z} = (\mathbf{I}-\mathbf{W})^{-1}(\mathbf{1}_{N\times 1} - \mathbf{y}) + \mathbf{X}\boldsymbol{\beta}_0$$
$$\mathbf{V} = \mathbf{MW}(\mathbf{I}-\mathbf{W})$$
$$c' = c - \frac{1}{2}\mathbf{z}^T\mathbf{V}\mathbf{z}$$

(D) Implement Newton's method for the logit model and test it out on the same data set you just used to test out gradient descent.

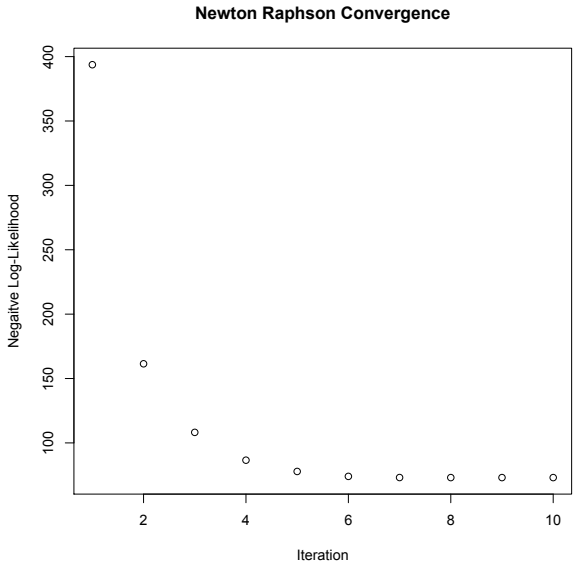

Figure 6: Convergence of Newton Raphson

(E) Reflect broadly on the tradeoffs inherent in the decision of whether to use gradient descent or Newton's method for solving a logistic-regression problem.

Newton's method clearly converges more quickly, reaching a solution in less than 10 steps. However, instead of just using the gradient, or first derivative, it additionally requires that you calculate the Hessian, or second derivative. This add considerable expense to each calculation.