# SDS385 HW 1

Evan Ott
UT EID: eao466

September 12, 2016

## Linear Regression

### (A)

WLS objective function:

$$\begin{aligned}\sum_{i=1}^{N} \frac{w_i}{2}(y_i - x_i^\top \beta)^2 &= \frac{1}{2}\sum_{i=1}^{N} y_i w_i y_i - \sum_{i=1}^{N} y_i w_i x_i^\top \beta + \frac{1}{2}\sum_{i=1}^{N} x_i^\top \beta w_i x_i^\top \beta \\ &= \frac{1}{2} y^\top W y - y^\top W X \beta + \frac{1}{2}(X\beta)^\top W X \beta \\ &= \frac{1}{2}(y - X\beta)^\top W (y - X\beta).\end{aligned}$$

Minimizing this function means setting the gradient (with respect to $\beta$) to zero:

$$\nabla_\beta \left[\frac{1}{2}(y - X\beta)^\top W (y - X\beta)\right] = 0$$

That is

$$\begin{aligned}\nabla_\beta \left[\frac{1}{2}(y - X\beta)^\top W (y - X\beta)\right] = 0 - (y^\top W X)^\top + \frac{2}{2} X^\top W X \hat{\beta} = 0 \\ \Rightarrow (X^\top W X)\hat{\beta} = X^\top W y \quad \blacksquare\end{aligned}$$

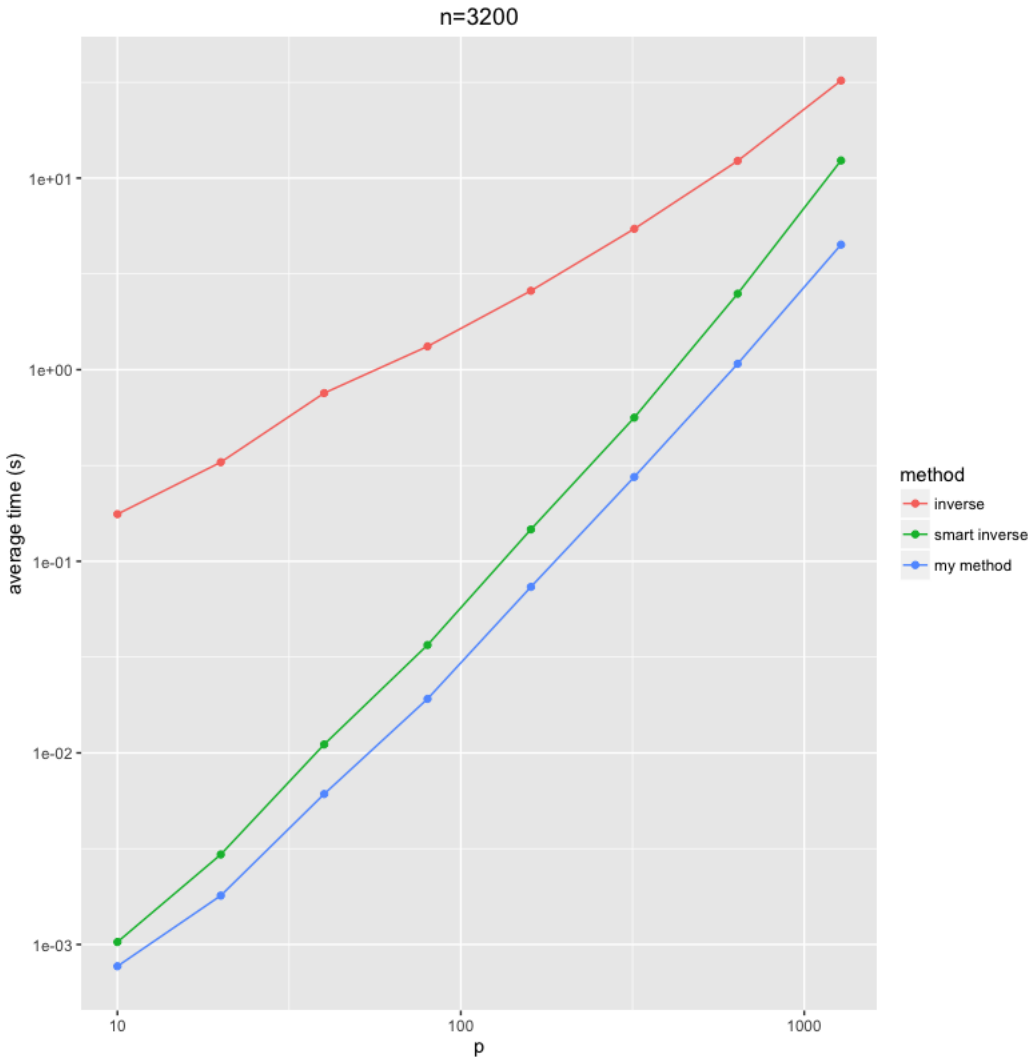
### (B)

The matrix factorization idea basically amounts to trying to prevent the full inverse operation. Overall, you will still probably need something $O(n^3)$, but the constants matter when actually doing computation as opposed to asymptotics. We don't actually want the inverse anyway, we just want to solve $(X^\top W X)\hat{\beta} = X^\top W y$ for $\beta$. Using a matrix decomposition can help with that a lot.

I based my solution on the Cholesky decomposition (see http://www.seas.ucla.edu/~vandenbe/103/lectures/chol.pdf). This creates matrices $X = LL^\dagger$, where $L$ is lower-triangular. So, then solve $Lz = X^\top W y$ for $z$ and $R\beta = z$ for $\beta$. The decomposition is still $O(n^3)$, but faster than inverse. The two final steps are each $O(n^2)$ which are faster.

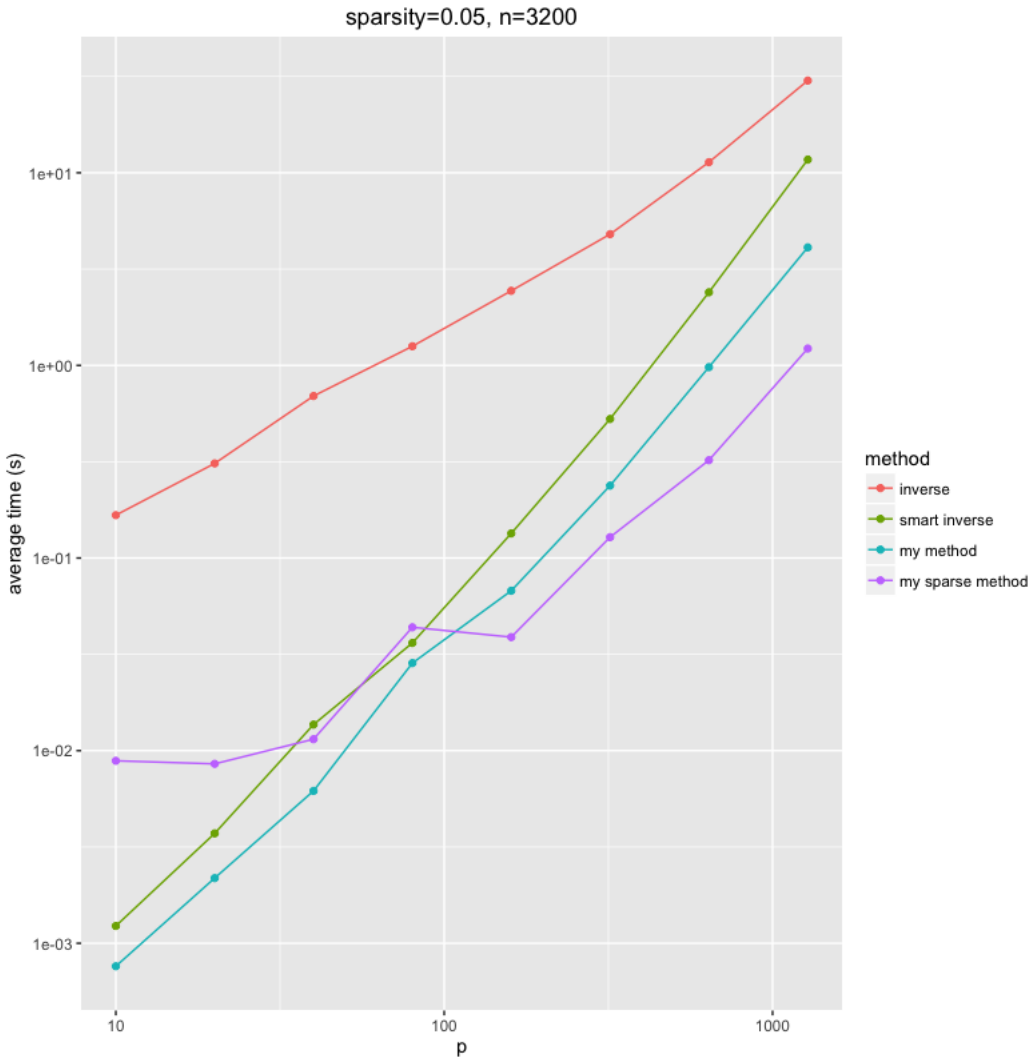
### (C)

See code on GitHub (`matrix_inverse_functions.R` and `matrix_inverse_run.R`).




## (D)

See code on GitHub (`matrix_inverse_functions.R` and `matrix_inverse_run.R`, look for the "Part D" in each).




## Notes from class

Three main matrix decomposition techniques:

1. Cholesky → fast, unstable (susceptible to roundoff error)
2. QR → middle ground
3. SVD → slow, but works for close-to-rank-deficient matrices

Using QR, we get $W^{1/2}X = QR$, where $R$ is $P \times P$ and upper-triangular (and thus invertible) and $Q$ is $N \times P$ with orthonormal columns.

$$
\begin{aligned}
X^\top W y &= X^\top W X \beta \\
X^\top W^{1/2} W^{1/2} y &= X^\top W^{1/2} W^{1/2} X \beta \\
(QR)^\top W^{1/2} y &= (QR)^\top QR\beta \\
Q^\top W^{1/2} y &= IR\beta = R\beta
\end{aligned}
$$

A note on R, `crossprod` computes $X^\top X$ but recognizes the symmetry so it takes half the time.

High level: do `solve(A, b)` better than $x = A^{-1}b$.

Intermediate level: (where we want to be) factorize $A$ using a decomposition to not do the whole inverse. Especially of note is whether we have sparse or dense matrices.

Low level: details of how the factorization works, what pivoting matrix to choose, etc.

Options for sparse matrices:

- Dictionary of keys
- List of lists
- Coordinate list (row, col, val)
- Compressed sparse row format: $A$ stores non-zero elements, $AI$ is row info, $AJ$ is column indexes

**Example to try**

To see numerical precision, take a look at

$$\begin{pmatrix} 10^{-5} & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

using $LL^\top$ and QR decomposition with intermediate rounding (6 decimal places or so).

In terms of sparse matrix decomposition, try using sparse $LL^\top$ or sparse Cholesky, rather than QR because $Q$ will not be sparse by definition, which makes all benefit be lost.

# Generalized linear models

## (A)

$$
\begin{aligned}
w_i(\beta) &= \frac{1}{1+\exp\{-x_i^\top \beta\}} \\
l(\beta) &= -\log\left\{\prod_{i=1}^{N} p(y_i|\beta)\right\} \\
&= -\log\left\{\prod_{i=1}^{N} \binom{m_i}{y_i} w_i(\beta)^{y_i}(1-w_i(\beta))^{m_i-y_i}\right\} \\
&= -\sum_{i=1}^{N} \log\left\{\binom{m_i}{y_i} w_i(\beta)^{y_i}(1-w_i(\beta))^{m_i-y_i}\right\} \\
&= -\sum_{i=1}^{N} \log\binom{m_i}{y_i} + y_i\log(w_i(\beta)) + (m_i-y_i)\log(1-w_i(\beta)) \\
\nabla l(\beta) &= -\sum_{i=1}^{N} 0 + \frac{y_i}{w_i(\beta)}\nabla w_i(\beta) - \frac{m_i-y_i}{1-w_i(\beta)}\nabla w_i(\beta) \\
\nabla w_i(\beta) &= w_i^2(\beta)e^{-x_i^\top\beta}x_i \\
\nabla l(\beta) &= -\sum_{i=1}^{N} w_i^2(\beta)e^{-x_i^\top\beta}x_i\left(\frac{y_i}{w_i(\beta)} - \frac{m_i-y_i}{1-w_i(\beta)}\right) \\
&= -\sum_{i=1}^{N} w_i^2(\beta)e^{-x_i^\top\beta}x_i\left(\frac{y_i - y_i w_i(\beta) - m_i w_i(\beta) + y_i w_i(\beta)}{w_i(\beta)(1-w_i(\beta))}\right) \\
&= -\sum_{i=1}^{N} w_i(\beta)e^{-x_i^\top\beta}x_i\left(\frac{y_i - m_i w_i(\beta)}{1-w_i(\beta)}\right) \\
&= -\sum_{i=1}^{N} w_i(\beta)\left(\frac{1}{w_i(\beta)} - 1\right)x_i\left(\frac{y_i - m_i w_i(\beta)}{1-w_i(\beta)}\right) \\
&= -\sum_{i=1}^{N} w_i(\beta)\frac{1-w_i(\beta)}{w_i(\beta)}x_i\left(\frac{y_i - m_i w_i(\beta)}{1-w_i(\beta)}\right) \\
&= -\sum_{i=1}^{N} \left[y_i - m_i w_i(\beta)\right]x_i \\
&= -X^\top(Y - Mw)
\end{aligned}
$$

where $M$ is a $N \times N$ diagonal matrix and $w$ is an $N$-vector with $M_{ii} = m_i$ and $w_i = w_i(\beta)$.

## (B)

See code on GitHub (`glm.R` and `glm_run.R`).

## (C)

The second order Taylor approximation can be seen in Eq (2.14) in Nocedal & Wright. First, we need the Hessian:

$$
\begin{aligned}
\nabla^2 l(\beta) = \nabla(\nabla l(\beta)) &= \nabla_\beta \left( -\sum_{i=1}^{N} \left[ y_i - m_i w_i(\beta) \right] x_i \right) \\
&= -\sum_{i=1}^{N} \left( \nabla_\beta \left[ y_i - m_i w_i(\beta) \right] x_i \right) \\
&= -\sum_{i=1}^{N} \nabla_\beta \left( y_i x_i - m_i w_i(\beta) x_i \right) \\
&= \sum_{i=1}^{N} m_i \nabla_\beta \left( w_i(\beta) x_i \right) \\
&= \sum_{i=1}^{N} m_i \left( w_i(\beta)(1 - w_i(\beta)) x_i^\top x_i \right) \\
&= X^\top M W (I - W) X
\end{aligned}
$$

where $M$ and $W$ are $N \times N$ diagonal matrices with $M_{ii} = m_i$ and $W_{ii} = w_i(\beta)$. By construction, this matrix will be positive semi-definite, which indicates that the problem itself is convex.

Now, we can start computing the approximation. Let $(w_0)_i = w_i(\beta_0)$ and $(W_0)_{ii} = w_i(\beta_0)$, slightly modifying the previous results.

$$
\begin{aligned}
q(\beta; \beta_0) &= l(\beta_0) + (\beta - \beta_0)^\top \nabla_{\beta_0} l(\beta_0) + \frac{1}{2} (\beta - \beta_0)^\top \nabla^2_{\beta_0} l(\beta_0) (\beta - \beta_0) \\
&= l(\beta_0) - (Y - M w_0)^\top X (\beta - \beta_0) + \frac{1}{2} (\beta - \beta_0)^\top X^\top M W_0 (I - W_0) X (\beta - \beta_0) \\
&= l(\beta_0) + (Y - M w_0)^\top (X\beta_0 - X\beta) + \frac{1}{2} (X\beta_0 - X\beta)^\top M W_0 (I - W_0) (X\beta_0 - X\beta) \\
A &= M W_0 (I - W_0) \\
b &= -A^{-1} (Y - M w_0) \\
q(\beta; \beta_0) &= \frac{1}{2} (X\beta_0 - b - X\beta)^\top A (X\beta_0 - b - X\beta) + c(\beta_0) \\
z &= X\beta_0 - b = X\beta_0 + \left[ M W_0 (I - W_0) \right]^{-1} (Y - M w_0) \\
q(\beta; \beta_0) &= \frac{1}{2} (z - X\beta)^\top A (z - X\beta) + c(\beta_0)
\end{aligned}
$$

That is, $A$ is an $N \times N$ diagonal matrix and $z$ is an $N$-vector with:

$$
\begin{aligned}
A_{ii} &= m_i w_i(\beta_0)(1 - w_i(\beta_0)) \\
z_i &= x_i^\top \beta_0 + \frac{Y_i - m_i w_i(\beta_0)}{m_i w_i(\beta_0)(1 - w_i(\beta_0))}
\end{aligned}
$$

## (D)

See code on GitHub (`glm.R` and `glm_run.R`).

## (E)

At least in my code, it seems that a couple things are true. First is that each iteration of the Newton's method is slower, which makes sense because of the matrix inverse. However, based on my tests, it seems that even with a good multiplier, the steepest direction method doesn't attain as good as a result as the Newton method. Furthermore, the steepest direction method took orders of magnitude more iterations to get to a reasonable result. In this case, it seems like the Newton method is the way to go. But I'd imagine that with high-$P$, the inverse may be too expensive to perform.

## Notes from class

Peer review due September 19.

For this problem, we have three kinds of convergence criteria:

1. an analytical criterion (for example, $\nabla l(\beta) = 0$ known from the problem itself
2. when a sequence $\beta^{(1)},\ \beta^{(2)},\ \beta^{(3)}, \ldots, \beta^{(n)}$ converges
3. $l(\beta^{(1)}),\ l(\beta^{(2)}),\ l(\beta^{(3)}), \ldots, l(\beta^{(n)})$ which is what almost all proofs use. It's the objective function and can be very flat near the optimum so it's better to use than $\beta$ on its own.

Look into `Rcpp` which basically just uses C++ code within R. In particular, can think about using `Armadillo` or `Eigen`.