Homework 1

Tavis Abrahamsen, Ray Bai, Syed Rhaman, Andrey Skripnikov

April 23, 2015

1 a.) To simulate the data, we first generate a 25000×3000 matrix of random values.

```
> set.seed(12345)
> n <- 25000
> p <- 3000
> dataMat <- matrix(runif(n*p,min=-1,max=1),n,p)
```

We then perform a singular value decomposition on the above matrix.

```
> A <- svd(dataMat)
> U <- A$u
> V <- A$v
> d <- A$d
```

Note that for a matrix $X \in \mathbb{R}^{n \times p}$ $(n \geq p)$, the singular value decomposition of X is given by $X = U\Sigma V^T$ where $U \in \mathbb{R}^{n \times n}$ and $V \in \mathbb{R}^{p \times p}$ are orthogonal matrices. The matrix $\Sigma \in \mathbb{R}^{n \times p}$ is defined as

$$\Sigma = \begin{bmatrix} D \\ 0 \end{bmatrix},$$

where $D \in \mathbb{R}^{p \times p}$ is a diagonal matrix. The diagonal entries of D are called the *singular values* of X. These values are also the square roots of the eigenvalues of $X^T X$. Thus, in order to generate our covariate data such that the Gram matrix, $X^T X$ has a condition number of 1, we simply replace D with the identity matrix.

```
> X <- U%*%t(V)
```

Note that in R, U is an $n \times p$ matrix of the first p left singular vectors. Since the singular values are all 1 in this case, the condition number of $X^T X$ will be 1/1=1.

To generate data with a condition number 30, we randomly draw the singular values from a Uniform$(1,\sqrt{30})$ distribution. We then set one of the values equal to 1 and another equal to $\sqrt{30}$. Hence the condition number of the corresponding Gram matrix will be $(\sqrt{30})^2/1 = 30$.

```
> d.1 <- runif(p,min=1, max=sqrt(30))
> d.1[1] <- sqrt(30)
> d.1[p] <- 1
> X <- U%*%diag(d.1)%*%t(V)
```

Similarly, for a condition number of 500, we have

```
> d.1 <- runif(p,min=1, max=sqrt(500))
> d.1[1] <- sqrt(500)
> d.1[p] <- 1
> X <- U%*%diag(d.1)%*%t(V)
```

To generate our response data, we randomly draw the components of β.

```
> true.beta <- rnorm(p,0,2)
```

Then for each of the covariate matrices above, we generate the corresponding response data $Y_i \sim \text{Bernoulli}(p_i)$ where

$$p_i := \frac{e^{x_i^T\beta}}{1+e^{x_i^T\beta}}, \quad i = 1, 2, \ldots, 2500.$$

```
> prob <- exp(X%*%true.beta)/(1+exp(X%*%true.beta))
> y <- rbinom(n,1,prob)
```

1 b.) We chose to estimate β using the maximum likelihood estimator. This estimator is obtained by maximizing the log-likelihood function with respect to to β. Let $L(\beta)$ denote the likelihood function, then

$$L(\beta) = \prod_{i=1}^{n} \left(\frac{e^{x_i^T\beta}}{1+e^{x_i^T\beta}}\right)^{y_i} \left(\frac{1}{1+e^{x_i^T\beta}}\right)^{1-y_i}.$$

The log-likelihood, $l(\beta)$ is then given by

$$\begin{aligned} l(\beta) &= \sum_{i=1}^{n} \left\{ y_i \log\left(\frac{e^{x_i^T\beta}}{1+e^{x_i^T\beta}}\right) + (1-y_i)\log\left(\frac{1}{1+e^{x_i^T\beta}}\right)\right\} \\ &= \sum_{i=1}^{n} \left\{ y_i x_i^T\beta - \log\big(1+e^{x_i^T\beta}\big)\right\}. \end{aligned}$$

To find the MLE of β, we could maximize $l(\beta)$, or equivialently we could minimize the function $f(\beta) = -l(\beta)$. To minimize the negative log-likelihood, we use the steepest descent algorithm which is given by

$$\beta^{k+1} = \beta^k - \alpha^k \nabla(f(\beta^k)) = \beta^k + \alpha^k \nabla l(\beta^k), \quad k = 0, 1, 2, \ldots.$$

Notice that

$$\frac{\partial l}{\partial \beta_j} = \sum_{i=1}^{n} \left\{ y_i x_{ij} - \frac{e^{x_i^T\beta}}{1+e^{x_i^T\beta}} x_{ij} \right\},$$

Thus,

$$\nabla l(\beta) = X^T(y - \eta(\beta)),$$

where $y^T = (y_1, y_2, \ldots, y_n)^T$ and $\eta(\beta) \in \mathbb{R}^n$ with

$$\eta_i = \frac{e^{x_i^T\beta}}{1+e^{x_i^T\beta}}, \quad i = 1, 2, \ldots, n.$$

Hence, our steepest descent algorithm is

$$\beta^{k+1} = \beta^k + \alpha^k X^T(y - \eta(\beta^k)), \quad k = 0, 1, 2, \ldots.$$

It is worth noting that $f(\beta)$ is a convex function. To see this, consider the function

$$g(x) = -\left(ax - \log(1+e^x)\right).$$

Indeed,

$$g''(x) = -\left(-\frac{e^x}{(1+e^x)^2}\right) = \frac{e^x}{(1+e^x)^2} > 0 \ \ \forall x \in \mathbb{R},$$

and thus $f(\beta)$ has a unique global minimum.

For each condition number (1,30,500) we compare the performace of the steepest descent algorithm for three different choices of step size, α^k. The first choice is using a constant step size, i.e., $\alpha^k = s$ where s is a constant for $k = 0, 1, 2, \ldots$. The second is using a diminishing step size where $\alpha^k \to 0$ such that

$$\sum_{k=0}^{\infty} \alpha^k = \infty.$$

Lastly, we choose the step size using Armijo's rule. Here, we start with a constant s, and for $\gamma \in (0, 1)$, we take $\alpha^k = s\gamma^l$ where l is the smallest non-negative integer satisfying the inequality

$$f(\beta^k) - f(\beta^k - s\gamma^l \nabla f(\beta^k)) \geq -\sigma s\gamma^l (-\nabla f(\beta^k))^T \nabla f(\beta^k)$$

for some $\sigma \in (0, 1]$. In terms of the log-likelihood function, the inequality becomes

$$l(\beta^k + s\gamma^l \nabla l(\beta^k)) - l(\beta^k) \geq \sigma s\gamma^l ||\nabla l(\beta^k)||^2.$$

For a condition number equal to 1, and using a constant step size our steepest descent algorithm is

```
> comp.grad <- function(y,X,beta){
+   t(X)%*%(y-(exp(X%*%beta)/(1+exp(X%*%beta))))
+ }
> log.lik <- function(y,X,beta){
+   p <- exp(X%*%beta)/(1+exp(X%*%beta))
+   sum(y*log(p)+(1-y)*log(1-p))
+ }
> alpha <- 5
> beta.old <- rep(1,p)
> delta <- .00001
> maxIter <- 1000
> d.f <- rep(0,maxIter)
> d.beta <- rep(0,maxIter)
> err.beta <- rep(0,maxIter)
> iter <- 1
> eps <- 1
> while(iter<maxIter && eps>delta ){
+   beta.new <- beta.old + alpha*comp.grad(y,X,beta.old)
+   eps <- abs(log.lik(y,X,beta.new)-log.lik(y,X,beta.old))
+   d.f[iter] <- eps
+   d.beta[iter] <- sqrt(t(beta.new-beta.old)%*%(beta.new-beta.old))
+   err.beta[iter] <- sqrt(crossprod((beta.new-true.beta),(beta.new-true.beta)))
+   beta.old <- beta.new
+   iter <- iter+1
+ }
```

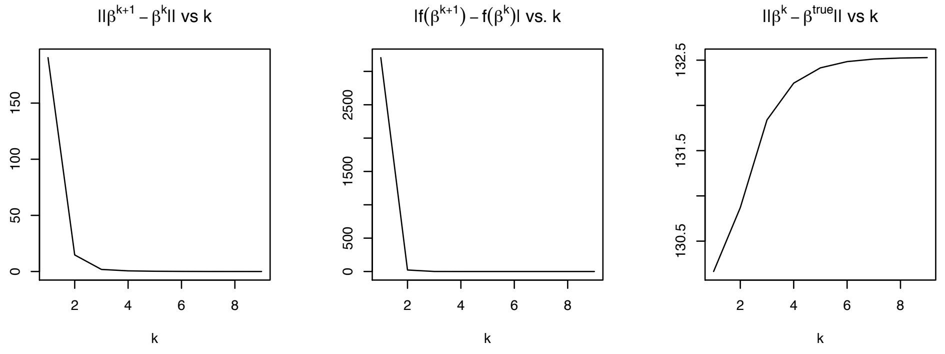
We ran our algorithm using a step size equal to 5 until the quantity $|f(\beta^{k+1}) - f(\beta^k)|$ is less than the specified tolerance of 10^{-5}. Shown below is the number of iterations the algorithm took to converge and $||\hat{\beta}_{MLE} - \beta||$, and plots of various convergence diagnostics.

```
> # Number of iterations
> iter

[1] 10
```

```
> # Error
> sqrt(t(true.beta-beta.old)%*%(true.beta-beta.old))

         [,1]
[1,] 132.5281
```



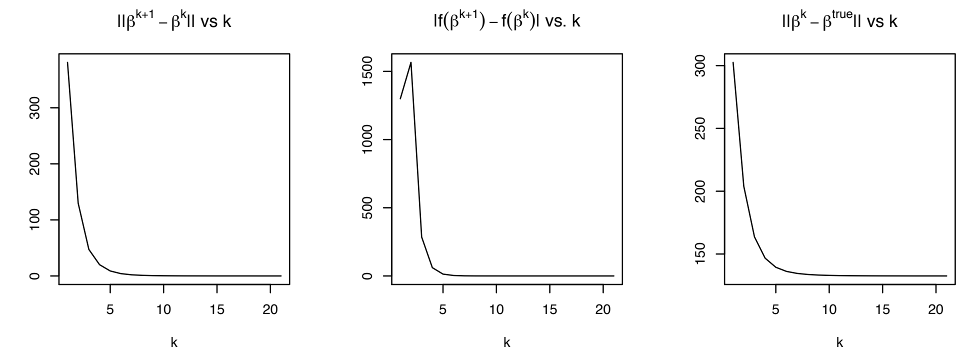
For the decreasing step size algorithm, we took $\alpha^k = \frac{10}{\sqrt{k}}$.

```
> alpha0 <- 10
> alpha <- alpha0
> beta.old <- rep(1,p)
> delta <- .00001
> maxIter <- 1000
> d.f <- rep(0,maxIter)
> d.beta <- rep(0,maxIter)
> err.beta <- rep(0,maxIter)
> iter <- 1
> eps <- 1
> while(iter<maxIter && eps>delta ){
+   beta.new <- beta.old + alpha*comp.grad(y,X,beta.old)
+   eps <- abs(log.lik(y,X,beta.new)-log.lik(y,X,beta.old))
+   d.f[iter] <- eps
+   d.beta[iter] <- sqrt(t(beta.new-beta.old)%*%(beta.new-beta.old))
+   err.beta[iter] <- sqrt(crossprod((beta.new-true.beta),(beta.new-true.beta)))
+   beta.old <- beta.new
+   iter <- iter+1
+   alpha <- alpha0/sqrt(iter)
+ }
> # Number of iterations
> iter

[1] 22

> # Error
> sqrt(t(true.beta-beta.old)%*%(true.beta-beta.old))

         [,1]
[1,] 132.5397
```



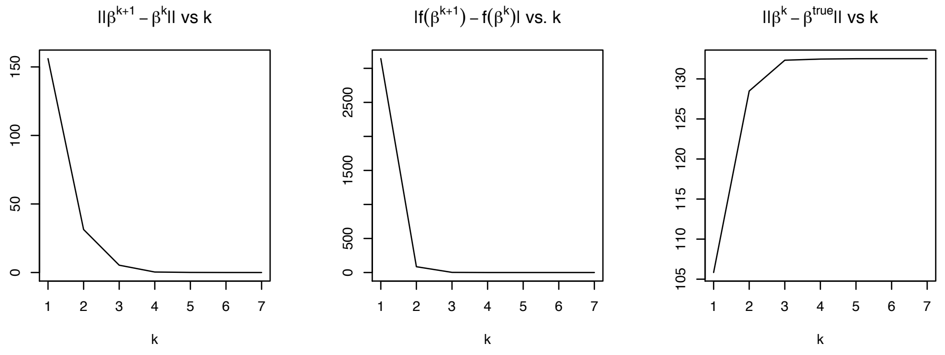
For the Armijo's rule step size we took $s = 10$, $\gamma = 0.8$ and $\sigma = 0.5$.

```
> s <- 10
> sigma <- .5
> gamma <- .8
> beta.old <- rep(1,p)
> delta <- .00001
> maxIter <- 1000
> d.f <- rep(0,maxIter)
> d.beta <- rep(0,maxIter)
> err.beta <- rep(0,maxIter)
> iter <- 1
> eps <- 1
> while(iter<maxIter && eps>delta ){
+   check <- 0
+   t <- 1
+   while(check==0){
+     grad.f <- comp.grad(y,X,beta.old)
+     beta.new <- beta.old + s*(gamma^t)*grad.f
+     a <- log.lik(y,X,beta.new)-log.lik(y,X,beta.old)
+     b <- sigma*s*(gamma^t)*t(grad.f)%*%grad.f
+     if(a >= b){
+       check <- 1
+     }
+     else{t <- t+1}
+   }
+   eps <- abs(log.lik(y,X,beta.new)-log.lik(y,X,beta.old))
+   d.f[iter] <- eps
+   d.beta[iter] <- sqrt(t(beta.new-beta.old)%*%(beta.new-beta.old))
+   err.beta[iter] <- sqrt(crossprod((beta.new-true.beta),(beta.new-true.beta)))
+   beta.old <- beta.new
+   iter <- iter+1
+ }
> # Number of iterations
> iter
```

```
[1] 8
```

```
> # Error
> sqrt(t(true.beta-beta.old)%*%(true.beta-beta.old))

       [,1]
[1,] 132.53
```



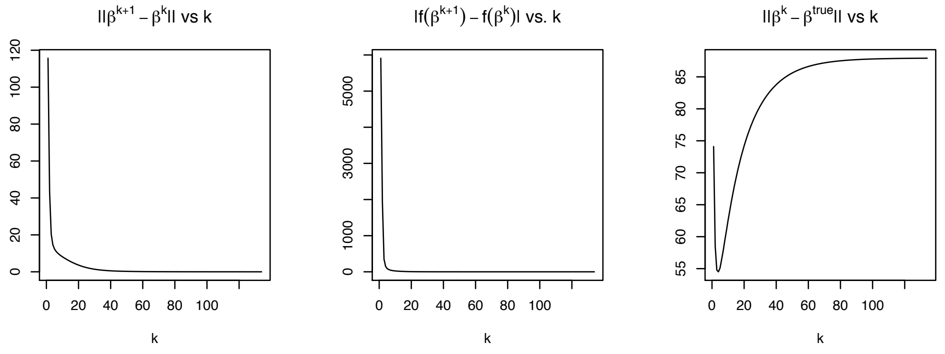
When the condition number is equal to 30, the constant step size algorithm would not converge when $\alpha = 5$. Instead, we had to use a step size of 0.5. The results are shown below.

```
> # Number of iterations
> iter

[1] 135

> # Error
> sqrt(t(true.beta-beta.old)%*%(true.beta-beta.old))

         [,1]
[1,] 87.91646
```



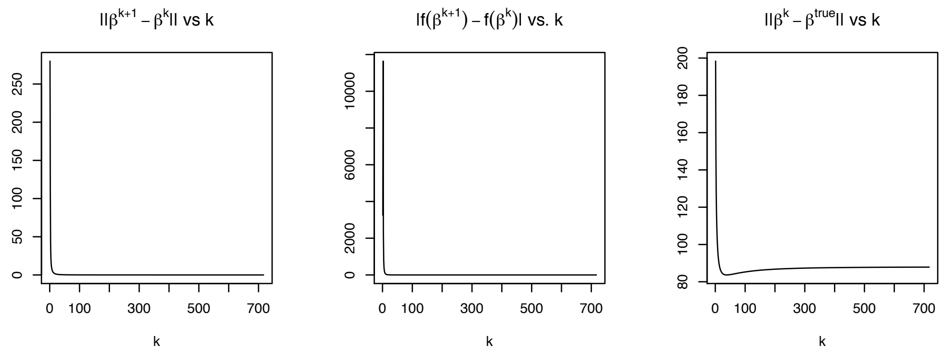
For the decreasing step size algorithm, $\alpha^k = 1/\sqrt{k}$.

```
> # Number of iterations
> iter
```

```
[1] 718

> # Error
> sqrt(t(true.beta-beta.old)%*%(true.beta-beta.old))

         [,1]
[1,] 87.85411
```



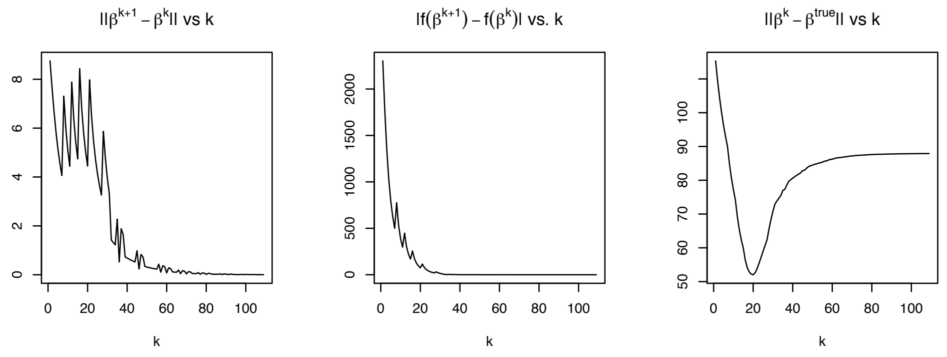
For Armijo's rule, we used the parameterization $s = 2$, $\sigma = 0.9$ and $\gamma = 0.5$.

```
> # Number of iterations
> iter

[1] 110

> # Error
> sqrt(t(true.beta-beta.old)%*%(true.beta-beta.old))

         [,1]
[1,] 87.90057
```



$||\beta^{k+1} - \beta^k||$ vs k
$|f(\beta^{k+1}) - f(\beta^k)|$ vs. k
$||\beta^k - \beta^{true}||$ vs k
k