Lisa Over
HW 10
March 31, 2015

CODE

Lisa Over 6/5/2015 11:23 AM
Comment [1]: Include the values of alpha-hat and beta-hat from the code in addition to the radius of convergence.

```
#nextX function receives a vector (theta), the ith vector, of two logistic regression coefficients - alpha and beta - and data vectors x and y. It calculates a new vector of alpha and beta (ntheta), the (i + 1)st vector, as the point at which the line tangent to the log likelihood function at the original vector theta crosses the x-axis.
nextX <- function(theta, x, y) {
	#break out vector into single alpha (a) and beta (b) values
	a = theta[1]
	b = theta[2]

	#Logistic probability function
	pi = exp(a + b*x)/(1 + exp(a + b*x))

	#Partial derivatives
	#derivative of the log likelihood function with respect to alpha
	da = sum(y - pi)
#derivative of the log likelihood function with respect to beta
	db = sum(x*(y - pi))
#second derivative of the log likelihood function with respect to alpha, then alpha
	da2 = -sum(pi*(1 - pi))
	#second derivative of the log likelihood function with respect to beta, then beta
	db2 = -sum((x^2)*pi*(1 - pi))
#second derivative of the log likelihood function with respect to alpha, then alpha
	dadb = -sum(x*pi*(1 - pi))

	#Matrix of first and second partial derivatives
	M = matrix(c(da2, dadb, dadb, db2), ncol = 2)

	#Use package "matrixcalc" for is.singular.matrix(M) - load with "library(matrixcalc)"
	#Or check if detminant is zero - if(det(M) == 0) - det being zero indicates singularity
	#If matrix is singular OR if it contains NaN, the function does not converge. Return ntheta as vector of "nc" values
	if(is.singular.matrix(M) | is.element("NaN", M)) {
		return(c("nc", "nc"))
	}
	#vector of first partial derivatives
	V = c(da, db)

	#New theta is theta minus the inverse of the matrix M times the vector V
	ntheta = theta - solve(M)%*%V
	return(ntheta)
}
```

Lisa Over
HW 10
March 31, 2015

#newt_raphson function receives two logistic coefficient vectors - alpha and beta, two data vectors x and y, and a threshhold value indicating a threshhold for acceptance of ntheta as the root of the log likelihood function. If the difference in magnitude between the two coefficient vectors is less than or equal to the threshhold, the ntheta is returned as the root of the log likelihood function. If not, the current ntheta is stored as theta and used as the ith vector to calculate the (i + 1)st vector.

```
newt_raphson <- function(theta, ntheta, x, y, threshhold) {

	#If the newly calculated theta - ntheta - contains "nc", then the function does not converge. Return ntheta and exit function.
	if(is.element("nc",ntheta)) {
		return(ntheta)
	}
	#If the difference in magnitude is less than the threshhold, return ntheta as the root
	else if(abs((sum(theta)^2) - (sum(ntheta)^2)) <= threshhold) {
		return(ntheta)
	}
	else {
		#If not, store the current ntheta is as theta and use the new theta as the ith vector to calculate the (i + 1)st vector.
		theta = ntheta
		ntheta = nextX(theta, x, y)
		#If the newly calculated theta - ntheta - contains "nc", then the function does not converge. Return ntheta and exit function.
		if(is.element("nc",ntheta)) {
			return(ntheta)
		}
		#If the difference in magnitude is greater than the threshhold, run newt_raphson again
		newt_raphson(theta, ntheta, x, y, threshhold)
	}
}
```

Radius of Convergence Code

```
#Data
x = c(21,24,25,26,28,31,33,34,35,37,43,49,51,55,25,29,43,44,46,46,51,55,56,58)
y = c(0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1)
#Obtain alpha and beta from data to use for a0 and b0
output = summary(glm(y~x,family=binomial))

#Set threshhold
threshhold = 0.000000001

#Create two vectors of the same length, one with a range of possible alpha values and one with a range of possible beta values. Combine the two vectors into a matrix of alpha-beta pairs
xa = seq(-12,8,by=.5)
```

Lisa Over
HW 10
March 31, 2015

```
yb = seq(-.5,.5,by=0.01)
k = 1
pair = NULL
m <- matrix(, nrow = length(xa)*length(yb), ncol = 2)
 for(i in 1:length(xa)) {
        for(j in 1:length(yb)) {
                pair = c(xa[i],yb[j])
                m[k ,] <- pair
                k=k+1
  }
 }

#Run the newt_raphson function with each alpha-beta pair and record in a vector - convec -
a 1 if the function converges and a 0 otherwise.
convec = NULL
for(i in 1:dim(m)[1]) {
        theta = c(m[i,1],m[i,2])
        ntheta = nextX(c(m[i,1],m[i,2]),x,y)
        root = newt_raphson(theta, ntheta, x, y, threshhold)
        if(is.element("nc", root)) {
                convec = c(convec, 0)
        }
        else {
                convec = c(convec, 1)
        }
}

#Create a matrix of only the points that converge and plot nonconverging points with pch=1
and add converging points with pch=19
k = 1
pair = NULL
mcon <- matrix(, nrow = length(convec), ncol = 2)
for(i in 1:length(convec)) {
 if(convec[i] == 1) {
                pair = c(m[i, 1], m[i, 2])
                mcon[k ,] = pair
                k = k + 1
  }
 }
matplot(m[,1],m[,2], pch=1)
matpoints(mcon[,1],mcon[,2], pch=19)
```

Lisa Over
HW 10
March 31, 2015

RESULTS

Radius of Convergence