STAT 428 – Homework #2
Luis Steven Lin

Problem 1

```
# Define Function

f = function(x){
      return(x^2)}

# Define Derivative

df = function(x){
      return(2*x)}

# Initialize variables
table = matrix(nrow=5,ncol=2)
colnames(table)=c('Iteration','Root estimate')
maxiter = 5
iter=1
err=1
x0 = 0.5
x = x0

# Loop
while( err > 10^-12 & iter <= maxiter) {
      xnew = x - f(x)/df(x)
      err = abs( (xnew-x)/xnew )*100
      x = xnew
      table[iter,1]=iter
      table[iter,2]=xnew
      iter = iter + 1;
      }
table
```

```
> table
     Iteration Root estimate
[1,]         1      0.250000
[2,]         2      0.125000
[3,]         3      0.062500
[4,]         4      0.031250
[5,]         5      0.015625
```

Note: if the maximum iterations is increased, then the root estimate gets closer to zero.

Problem 2

```
# Define Function

f = function(x){
      return(x^(1/3))}

# Define Derivative

df = function(x){
      return(1/3*x^(-2/3))}

# Initialize variables
table = matrix(nrow=5,ncol=2)
colnames(table)=c('Iteration','Root estimate')
maxiter = 5
iter=1
err=1
x0 = 0.5
x = x0

# Loop
while( err > 10^-12 & iter <= maxiter) {
      xnew = x - f(x)/df(x)
      #err = abs( (xnew-x)/xnew )*100
      x = xnew
      table[iter,1]=iter
      table[iter,2]=xnew
      iter = iter + 1;
      }
table
```

```
> table
     Iteration Root estimate
[1,]         1            -1
[2,]         2           NaN
[3,]         3           NaN
[4,]         4           NaN
[5,]         5           NaN
```

The algorithm is not able to compute a root estimate after the first iteration because the derivative of the function is not defined in the real space for a negative number (also in R the root cube is not defined for a negative number).

```
# Repeat the algorithm by simplifying f/f' expression
table = matrix(nrow=5,ncol=2)
colnames(table)=c('Iteration','Root estimate')
maxiter = 5
iter=1
err=1
x0 = 0.5
x = x0

# Loop
while( err > 10^-12 & iter <= maxiter) {
      xnew = x - 3*x
      err = abs( (xnew-x)/xnew )*100
      x = xnew
      table[iter,1]=iter
      table[iter,2]=xnew
      iter = iter + 1;
      }
table
```

```
> table
     Iteration Root estimate
[1,]         1            -1
[2,]         2             2
[3,]         3            -4
[4,]         4             8
[5,]         5           -16
```

Note: if the maximum iterations is increased, then the root estimate gets larger in magnitude.

The expression f/f’ in the newton update can be simplified to 3x using algebra. If the algorithm is run with this expression, the algorithm will be diverging to infinity (the Newton update doubles the distance from the solution at each iteration). This is because the derivative does not exist at the root x=0.

Problem 3

a) $Y_i \sim \text{iid Poisson}(\lambda_i)$

$\log \lambda_i = \beta_1 x_{i1} + \beta_2 x_{i2} \rightarrow \lambda_i = e^{x_i'\beta}, \quad \beta = \begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix}, \quad x_i = \begin{pmatrix} x_{i1} \\ x_{i2} \end{pmatrix}$

$$P(Y = y_i) = \frac{e^{-\lambda_i} \lambda_i^{y_i}}{y_i!}$$

$$\ell(\beta) = \prod_{i=1}^{n} \frac{e^{-\lambda_i} \lambda_i^{y_i}}{y_i!}$$

$$L(\beta) = \log(\ell(\beta)) = \sum_{i=1}^{n} (-\lambda_i + y_i \log(\lambda_i) - \log y_i!)$$

$$\boxed{L(\beta \mid x, y) = \sum_{i=1}^{n} -e^{x_i'\beta} + y_i x_i' \beta - \log y_i!}$$

b) $\frac{\partial L}{\partial \beta_j} \quad \sum_{i=1}^{n} -\lambda_i x_{ij} + y_i x_{ij} = \sum_{i=1}^{n} (y_i - \lambda_i) x_{ij} = \sum_{i=1}^{n} (y_i - e^{x_i'\beta}) x_{ij}$

$$\Rightarrow \boxed{L'(\beta \mid x, y)} = \begin{pmatrix} \frac{\partial L}{\partial \beta_1} \\ \frac{\partial L}{\partial \beta_2} \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{n} (y_i - e^{x_i'\beta}) x_{i1} \\ \sum_{i=1}^{n} (y_i - e^{x_i'\beta}) x_{i2} \end{pmatrix}$$

c) $\left. \begin{array}{l} \frac{\partial^2 L}{\partial \beta_j^2} = \sum_{i=1}^{n} -x_{ij}^2 e^{x_i'\beta} \\ \frac{\partial^2 L}{\partial \beta_j \partial \beta_k} = \sum_{i=1}^{n} -x_{ij} x_{ik} e^{x_i'\beta} \end{array} \right\} \Rightarrow L''(\beta \mid x, y) = \begin{pmatrix} \frac{\partial^2 L}{\partial \beta_1^2} & \frac{\partial^2 L}{\partial \beta_1 \partial \beta_2} \\ \frac{\partial^2 L}{\partial \beta_2 \partial \beta_1} & \frac{\partial^2 L}{\partial \beta_2^2} \end{pmatrix}$

$$\Rightarrow \underbrace{\boxed{L''(\beta \mid x y)}}_{\text{Hessian}} = \begin{pmatrix} -\sum_{i=1}^{n} x_{i1}^2 e^{x_i'\beta} & -\sum_{i=1}^{n} x_{i1} x_{i2} e^{x_i'\beta} \\ -\sum_{i=1}^{n} x_{i1} x_{i2} e^{x_i'\beta} & -\sum_{i=1}^{n} x_{i2}^2 e^{x_i'\beta} \end{pmatrix}$$

d) $\beta_j^{(k+1)} = \beta_j^{(k)} - \frac{L'(\beta_j^{(k)})}{L''(\beta_j^{(k)})}$

$$\Rightarrow \begin{bmatrix} \beta_1^{(k+1)} \\ \beta_2^{(k+1)} \end{bmatrix} = \begin{bmatrix} \beta_1^{(k)} \\ \beta_2^{(k)} \end{bmatrix} - \left[L''(\beta \mid x, y)^{(k)} \right]^{-1} L'(\beta \mid x, y)^{(k)}$$

R-CODE to estimate beta 1 and beta 2 by Newton's method

```
newton = function(x,y){
      # Scale x2 by 100 to make computations more manageable
      x[,2]=x[,2]/100

      # Note: x1 is not scaled because beta1 is the intercept

      beta0 = rep(0.5,ncol(x))
      epsis = 10^-10
      epsia = 1
      maxiter=100
      iter = 0
      while(epsia >= epsis & iter<=maxiter){
            # Lambda
            e = as.vector(exp(x%*%beta0))

            # Gradient
            DL=t(x)%*%(y - e)

            # Second partial derivatives
            D2L1= as.numeric(-x[,1]^2%*%e)
            D2L2= as.numeric(-x[,2]^2%*%e)
            D2L12 = as.numeric(-(x[,1]*x[,2])%*%e)

            # Hessian
            D2L = matrix(c(D2L1,D2L12,D2L12,D2L2),nrow=2,ncol=2,byrow=T)

            # Newton update
            beta = beta0-as.vector(solve(D2L)%*%DL)

            # Tolerance error
            epsia=sum(abs((beta-beta0)/beta)*100)
            beta0=beta
            iter=iter+1
      }

# Scale beta 2 back to original units
beta[2]=beta[2]/100

print(paste("Iterations: ",iter))
print("Estimates for beta1 and beta2:")
return(beta)
}

x1 = rep(1, 17)
x2 =
c(36,531,4233,8682,7164,2229,600,164,57,722,1517,1828,1539,2416,3148,3465
,1440)
y = c(0,0,130,552,738,414,198,90,56,50,71,137,178,194,290,310,149)
x = cbind(x1,x2)
```

```
> newton(x,y)


[1] "Iterations:  43"
[1] "Estimates for beta1 and beta2:"
[1] 4.5847691621 0.0002375121
```

R-CODE using single value decomposition leads to the same answer

```
newton2 = function(x,y){
      # Scale x2 by 100 to make computations more manageable
      x[,2]=x[,2]/100

      # Note: x1 is not scaled because beta1 is the intercept

      beta0 = rep(0.5,ncol(x))
      epsis = 10^-10
      epsia = 1
      maxiter=100
      iter = 0
      while(epsia >= epsis & iter<=maxiter){
            # Lambda
            e = as.vector(exp(x%*%beta0))

            # Use single value decomposition
            W=diag(-e)
            ystar = y - e
            my.M = t(x)%*%W%*%x
            my.svd=svd(my.M)
            my.svdinv=my.svd$u%*%diag(1/my.svd$d)%*%my.svd$v

            beta=beta0-my.svdinv%*%t(x)%*%ystar

            # Tolerance error
            epsia=sum(abs((beta-beta0)/beta)*100)

            beta0=beta
            iter=iter+1
      }

# Scale beta 2 back to original units
beta[2]=beta[2]/100

print(paste("Iterations: ",iter))
print("Estimates for beta1 and beta2:")
return(beta)
}

newton2(x,y)
```

R-CODE with GLM function

```
# Use R built-in function to compare

data= data.frame(x2,y)
glm(y~.,data=data,family=poisson)
```

```
Call:  glm(formula = y ~ ., family = poisson, data = data)

Coefficients:
(Intercept)           x2
  4.5847692    0.0002375

Degrees of Freedom: 16 Total (i.e. Null);  15 Residual
Null Deviance:      2884
Residual Deviance: 1142         AIC: 1252
```

Note that the results from the code using Newton method agree with those of the built-in R function, indicating a successful implementation of the algorithm.

Problem 4

$x_1, \dots x_n \overset{iid}{\sim} N(\mu, 1)$ $\left(\begin{matrix}\text{Standard} \\ \text{Normal}\end{matrix} \rightarrow f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \right.$

Let $h(\bar{x}) = \Phi\left[\underbrace{\left((c-\bar{x})\sqrt{\frac{n}{n-1}}\right)}_{u} \right] = \int_{-\infty}^{u} \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}} dt$

By the fundamental theorem of calculus : $\left(\frac{d}{dx} \int_a^x f(t)\,dt = f(x) \right)$

Let $u = (c-\bar{x})\sqrt{\frac{n}{n-1}}$ and $y = h(\bar{x})$

$$\frac{dy}{d\bar{x}} = \frac{dy}{du}\frac{du}{d\bar{x}} = \left(\frac{d}{du} \int_{-\infty}^{u} \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}} dt \right) \left(\frac{du}{d\bar{x}} \right)$$

$$= \left(\frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}} \right) \left(-\sqrt{\frac{n}{n-1}} \right) = -\sqrt{\frac{n}{2\pi(n-1)}}\, e^{-\frac{1}{2}\left(\frac{n}{n-1}\right)(c-\bar{x})^2} = h'(\bar{x})$$

$$h''(\bar{x}) = -\left(\sqrt{\frac{n}{2\pi(n-1)}} \right) \left(\frac{n}{n-1} \right) \left(c-\bar{x} \right) e^{-\frac{1}{2}\left(\frac{n}{n-1}\right)(c-\bar{x})^2}$$

$$= -\left(\frac{n}{n-1} \right) \left(\sqrt{\frac{n}{2\pi(n-1)}} \right) (c-\bar{x})\, e^{-\frac{1}{2}\left(\frac{n}{n-1}\right)(c-\bar{x})^2}$$

$\Rightarrow$ Using proposition 4.3.1 , $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} x_i$

$$\begin{cases} E(h(\bar{x})) = h(\mu) + \frac{\sigma^2}{2n} h''(\mu) + o\left(\frac{1}{n^2}\right) \\ Var(h(\bar{x})) = \frac{\sigma^2}{n} h'(\mu)^2 + o\left(\frac{1}{n^2}\right) \end{cases}$$

$$\Rightarrow \begin{cases} E(h(\bar{x})) = \boxed{\Phi\left((c-\mu)\sqrt{\frac{n}{n-1}} \right) - \left(\frac{1}{2}\right)\left(\frac{1}{n-1}\right)\sqrt{\frac{n}{2\pi(n-1)}}\,(c-\mu)\, e^{-\frac{1}{2}\left(\frac{n}{n-1}\right)(c-\mu)^2} + o\left(\frac{1}{n^2}\right)} \\ Var(h(\bar{x})) = \left(\frac{1}{\cancel{n}}\right)\left(-\sqrt{\frac{\cancel{n}}{2\pi(n-1)}}\right)^2 \left(e^{-\frac{1}{2}\left(\frac{n}{n-1}\right)(c-\mu)^2}\right)^2 + o\left(\frac{1}{n^2}\right) \\ \qquad = \boxed{\frac{1}{2\pi(n-1)} e^{-\frac{n}{n-1}(c-\mu)^2} + o\left(\frac{1}{n^2}\right)} \end{cases}$$

Note that the Mathematica Output shown below agrees.

In[10]:= h = CDF[NormalDistribution[], (c - x) * √(n / (n - 1))]

Out[10]= $$\frac{1}{2}\,\text{Erfc}\left[-\frac{\sqrt{\frac{n}{-1+n}}\,(c-x)}{\sqrt{2}}\right]$$

In[12]:= h1 = D[h, x]; h1 // Simplify

Out[12]= $$-e^{-\frac{n\,(c-x)^2}{2\,(-1+n)}}\sqrt{-\frac{n}{2\,\pi-2\,n\,\pi}}$$

In[14]:= h2 = D[h, {x, 2}]; h2 // Simplify

Out[14]= $$-\frac{e^{-\frac{n\,(c-x)^2}{2\,(-1+n)}}\,n\,\sqrt{-\frac{n}{2\,\pi-2\,n\,\pi}}\,(c-x)}{-1+n}$$

In[19]:= Ex = h + σ^2 / (2 n) * h2; Ex /. {σ → 1, x → μ} // Simplify

Out[19]= $$\frac{e^{-\frac{n\,(c-\mu)^2}{2\,(-1+n)}}\sqrt{-\frac{n}{2\,\pi-2\,n\,\pi}}\,(-c+\mu)}{2\,(-1+n)}+\frac{1}{2}\,\text{Erfc}\left[\frac{\sqrt{\frac{n}{-1+n}}\,(-c+\mu)}{\sqrt{2}}\right]$$

In[20]:= Va = σ^2 / (n) * h1^2; Va /. {σ → 1, x → μ} // Simplify

Out[20]= $$\frac{e^{-\frac{n\,(c-\mu)^2}{-1+n}}}{2\,(-1+n)\,\pi}$$