ASSIGNMENT -5

Deepak Karunakaran
STP-598

PART A

1. Modify the code displayed for logistic regression (binomial responses) to create a Fisher-Scoring algorithm that produces parameter estimates and confidence intervals for the Poisson regression model.
2. Are there differences between Newton-Raphson and Fisher-Scoring in this case?
3. Compare your results versus the ones obtained from the glm function for the following data:

Given :

```
counts <- c( 18,17,15,20,10,20,25,13,12)
outcome <- gl( 3,1,9)
treatment<-gl(3,3)
print (d.AD <- data.frame(treatment,outcome,counts))
glm.D93 <- glm(counts~outcome + treatment,family = poisson())
```

INDEX

1. **Introduction**
2. **THEORY**
3. **Newton-Raphson**
4. **Fisher Scoring**
5. **Using GLM**
6. **Newton vs Fisher vs GLM**
7. **General comments**

1.Introduction

Poisson regression - Poisson regression is often used for modeling count data.

In Poisson regression

- Link functions: g(x) = log(x);
- Variance function: = λ_i
- the canonical link is log λ

gl function creates a factor variable.

The code given for logistic regression in the slides was used as a base for the Poisson regression.

```
class(outcome)
[1] "factor"
```

Given Data

9x3 DataFrame

#	treatment	outcome	counts
1	1	1	18
2	1	2	17
3	1	3	15
4	2	1	20
5	2	2	10
6	2	3	20
7	3	1	25
8	3	2	13
9	3	3	12

Dummy Variables.

We need to define the dummy variables when we do Newton Raphson and Fisher iterations. This is done based on the factor 'Levels'.

There are two dummy variables each for

1) Treatment
2) Outcome

Convert the Data to this format:

counts	outcome2	outcome3	treatment2	treatment3
18	0	0	0	0
17	1	0	0	0
15	0	1	0	0
20	0	0	1	0
10	1	0	1	0
20	0	1	1	0
25	0	0	0	1
13	1	0	0	1
12	0	1	0	1

Glm- R by default creates two dummy variables namely outcome2 and outcome3 treatment2 and treatment3. They will be similar to the variables manually created.

2.THEORY

• Loglinear Poisson regression model

$$\log(\lambda_i) = \beta_0 + \beta_1 x_{i1} + \ldots \beta_p x_{ip} = x^i \beta\,, \quad i = 1, \ldots, n$$

$$\lambda_i = \lambda_i(\beta) = e^{x\beta} \qquad or \qquad log(\lambda_i) = x_i \beta$$

$$\lambda = E(X) = var(X)$$

Likelihood and MLE

The PMF for the data is the likelihood function:

$$L = \prod_{i=1}^{n} (\lambda^{k_i} * e^{\lambda}) / k_i!$$

By differentiating the **log of joint density function** with respect to λ we get:

$$l(\lambda) = -n\lambda + \sum_{i=1}^{n} k_i * 1/\lambda$$

MLE for λ is =sample mean

$$\lambda = \sum_{i=1}^{n} \frac{k_i}{n}$$

3.Newton-Raphson

- Finding the MLE =finding the root of score function.
- And no analytical solutions for l'(λ)=0 where l is log likelihood function.

$$\beta_{n+1} = x_n - \frac{f'(\beta_n)}{f''(\beta_n)}$$

- f' is the Likelihood gradient
- f'' is the second derivative of likelihood.
- Lline search done to converge faster by choosing appropriate step size.
- Newton's method converges depends on the shape of function and the starting value.

Implementaion

We use **maxNR()** function from **maxLik package()**

Looking at the **max NR** object that was produced

```
out2
```

Maximum value of the function

```
$maximum
[1] 274.7377
```

estimate of coefficents

$estimate

```
[1]  3.0428743805 -0.4537286295 -0.2924604819  0.0009048162  0.0009048162
```

last gradient value which was calculated.

```
$gradient
```

```
[1] 0.11084935   0.02071914    0.02434499    0.02188293      0.02188293
```

- **Gradient should be close to 0 if normal convergence. Values obtained are close to zero.**

hessian matrix

```
$hessian
          [,1]      [,2]      [,3]      [,4]      [,5]
[1,] -149.8892   0.00000   0.00000   0.00000   0.00000
[2,]    0.0000 -39.97928   0.00000   0.00000   0.00000
[3,]    0.0000   0.00000 -46.97566   0.00000   0.00000
[4,]    0.0000   0.00000   0.00000 -49.97812   0.00000
[5,]    0.0000   0.00000   0.00000   0.00000 -49.97812


$message
[1] "successive function values within tolerance limit"
```

Number of iterations

```
$iterations
[1] 60
```

Method used for maximization

```
$type
[1] "Newton-Raphson maximisation"
```

4. Fisher Scoring

- Fisher's scoring algorithm is a variation of Newton's method for solving maximum likelihood problems numerically.

The **score function** is given by

$$\frac{\partial l}{\partial \beta} = 0$$

$$variance = \begin{matrix} \lambda_1 & 1 & 1 \\ 1 & \ddots & 1 \\ 1 & 1 & \lambda_n \end{matrix}$$

Fisher scoring algorithm

$$\beta_{i+1} = \beta_i + (inverse\ info)(score\ function)$$

Fisher object

Out- list of important values:

```
##  out<-list()
##  out$beta.MLE<-beta.1
##  out$iter<-i-1
##  out$NR.hist<-NR.hist
##  out$beta.hist<-beta.hist
##  v.1<-diag(as.vector(exp(X%*%beta.2)))
##  Iinv.1<-solve(t(X)%*%v.1%*%X) #inverse information matrix
##  out$beta.cov<-Iinv.1
```

> out
$beta.MLE

	[,1]
int	3.044522e+00
outcome2	-4.542553e-01
outcome3	-2.929871e-01
treatment2	-4.780415e-18
treatment3	-4.780415e-18

$iter
[1] 5

$NR.hist

i		diff.beta	diff.like	llike.1	step.size
1	1	Inf	1.000000e+09	-9.0000	1.0
2	2	4.414622e+00	2.420191e+02	233.0191	0.2
3	3	1.600383e+00	4.146886e+01	274.4880	1.5
4	4	1.136354e-01	2.495306e-01	274.7375	1.0
5	5	2.843972e-03	2.775356e-04	274.7378	1.0
6	6	4.259557e-06	4.220055e-10	274.7378	1.0

$beta.hist

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	0.000000	0.0000000	0.0000000	0.000000e+00	0.000000e+00
[2,]	4.000000	-1.5333333	-1.0666667	0.000000e+00	0.000000e+00
[3,]	3.076943	-0.4129304	-0.3929547	4.455148e-16	4.455148e-16
[4,]	3.045042	-0.4521216	-0.2911735	-5.418280e-18	-5.418280e-18
[5,]	3.044523	-0.4542519	-0.2929845	-4.779700e-18	-4.779700e-18
[6,]	3.044522	-0.4542553	-0.2929871	-4.780415e-18	-4.780415e-18

##$beta.cov

	int	outcome2	outcome3	treatment2	treatment3
int	0.02920632	-1.587301e-02	-1.587301e-02	-1.999996e-02	-1.999996e-02
outcome2	-0.01587301	4.087293e-02	1.587301e-02	-7.739816e-18	-7.739816e-18
outcome3	-0.01587301	1.587301e-02	3.714955e-02	-9.610654e-18	-9.610654e-18
treatment2	-0.01999996	-7.739816e-18	-9.610654e-18	3.999993e-02	1.999996e-02
treatment3	-0.01999996	-7.739816e-18	-9.610654e-18	1.999996e-02	3.999993e-02

5.Using GLM

glm is used to fit generalized linear models in R.

GLM object

```
##  summary(glm.D93)

Call:
glm(formula = counts ~ outcome + treatment, family = poisson())
Deviance Residuals:
       1         2         3         4         5         6         7         8         9
-0.67125   0.96272  -0.16965  -0.21999  -0.95552   1.04939   0.84715  -0.09167  -0.96656
Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  3.045e+00  1.709e-01  17.815   <2e-16 ***
outcome2    -4.543e-01  2.022e-01  -2.247   0.0246 *
outcome3    -2.930e-01  1.927e-01  -1.520   0.1285
treatment2   1.338e-15  2.000e-01   0.000   1.0000
treatment3   1.421e-15  2.000e-01   0.000   1.0000
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
(Dispersion parameter for poisson family taken to be 1)
    Null deviance: 10.5814  on 8  degrees of freedom
Residual deviance:  5.1291  on 4  degrees of freedom
AIC: 56.761

Number of Fisher Scoring iterations: 4
```

Calculating the Log-Likelihood

```
## (sum(dpois(counts,
+            lambda=exp(predict(glm.D93)),log=TRUE)))
[1] -23.38066

##  (logLik(glm.D93))

'log Lik.' -23.38066 (df=5)
```

Confidence Interval of GLM

```
confint(glm.D93)
Waiting for profiling to be done...
                 2.5 %      97.5 %
(Intercept)  2.6958215  3.36655581
outcome2    -0.8577018 -0.06255840
outcome3    -0.6753696  0.08244089
treatment2  -0.3932548  0.39325483
treatment3  -0.3932548  0.39325483

## confint.default(glm.D93)  # based on  normality
                 2.5 %      97.5 %
(Intercept)  2.7095672  3.37947764
outcome2    -0.8505027 -0.05800787
outcome3    -0.6707552  0.08478093
treatment2  -0.3919928  0.39199279
```

```
treatment3  -0.3919928  0.39199279
```

ANOVA GLM

```
## anova(glm.D93)
Analysis of Deviance Table
Model: poisson, link: log
Response: counts
Terms added sequentially (first to last)
          Df Deviance Resid. Df Resid. Dev
NULL                          8    10.5814
outcome    2   5.4523         6     5.1291
treatment  2   0.0000         4     5.1291
```

Calculating confidence intervals, standard errors Z value and p value

1. Method mentioned in the slides.
2. R package **sandwich** to obtain **robust standard errors.**

1)Method mentioned in the slides

```
## beta.Est<-out$beta.MLE
## beta.SE<-sqrt(diag(out$beta.cov))
## beta.z<-beta.Est/beta.SE
## beta.pval<-2*pnorm(-abs(beta.z))
## beta.2.5<-paste((format(round((beta.Est-1.96*beta.SE), 4), nsmall =
4)),",",format(round((beta.Est+1.96*beta.SE), 4)), nsmall = 4)
## beta.2.5<-beta.Est-1.96*beta.SE
## beta.97.5<-beta.Est+1.96*beta.SE
## beta.coef<-data.frame(beta.Est,beta.SE,beta.z,beta.pval,beta.2.5,beta.97.5)
## out1<-newton(d.AD,beta.1)
```

beta.coef

	beta.Est	beta.SE	beta.z	beta.pval	beta.2.5	beta.97.5
int	3.044522e+00	0.1708986	1.781479e+01	5.425979e-71	2.7095612	3.37948364
outcome2	-4.542553e-01	0.2021705	-2.246892e+00	2.464696e-02	-0.8505095	-0.05800102
outcome3	-2.929871e-01	0.1927422	-1.520099e+00	1.284862e-01	-0.6707618	0.08478757
treatment2	-4.780415e-18	0.1999998	-2.390210e-17	1.000000e+00	-0.3919996	0.39199964
treatment3	-4.780415e-18	0.1999998	- 2.390210e-17	1.000000e+00	-0.3919996	0.39199964

- Since estimate of $\beta_1 > 0$, This implies increase in value of count on the multiplicative order of *exp*(3.044522) = 20.99999
- The coefficient for **treatment2 and treatment3 is -4.780415e-18** .This means that the expected log count for a one-unit increase in **treatment2 and treatment3 is -4.780415e-18.**

2)R package sandwich

```
## cov.glm.D93 <- vcovHC(glm.D93, type="HC0")
## std.err <- sqrt(diag(cov.glm.D93))
## r.est <- cbind(Estimate= coef(glm.D93), "Robust SE" = std.err,
##                "Pr(>|z|)" = 2 * pnorm(abs(coef(glm.D93)/std.err), lower.tail=FALSE),
##                LL = coef(glm.D93) - 1.96 * std.err,
##                UL = coef(glm.D93) + 1.96 * std.err)

r.est
```

	Estimate	Robust SE	Pr(>|z|)	LL (2.5)	UL(97.5)
(Intercept)	3.044522e+00	0.1162668	3.870897e-151	2.8166395	3.272405328
outcome2	-4.542553e-01	0.1482142	2.177737e-03	-0.7447550	-0.163755533
outcome3	-2.929871e-01	0.1460779	4.488925e-02	-0.5792998	-0.006674412
treatment2	1.337909e-15	0.1466667	1.000000e+00	-0.2874667	0.287466669
treatment3	1.421085e-15	0.1448371	1.000000e+00	-0.2838807	0.283880663

OBSERVATION

- **Robust method gives smaller range of confidence interval as it provides smaller Standard error values**

chi-squared test

RESIDUAL DEVIANCE As a measure of GOODNESS OF FIT ***

LL = loglikelihood

Residual Deviance = 2(LL(Saturated Model) - LL(Proposed Model)) df = df_Sat - df_Res

- Residual deviance is the difference between the deviance of the current model and the maximum deviance of the ideal model where the predicted values are identical to the observed.
- Small residual difference means the good fit.
- If the model is correct, the residual deviance should be close to χ^2 with the stated degrees of freedom.

```
## with(glm.D93, cbind(res.deviance = deviance, df = df.residual,
## +                   p = pchisq(deviance, df.residual, lower.tail=FALSE)))
##plot(counts,col="Blue",xlim = range(1:9),main="Plot of Obseved vs
##Predicted",ylab="COUNTS",xlab="INDEX")
##points(glm.D93$fitted,pch="p",col="red")
##legend(8,22,c("obs","pred"),pch=c("o","p"),col=c("blue","red"))
```

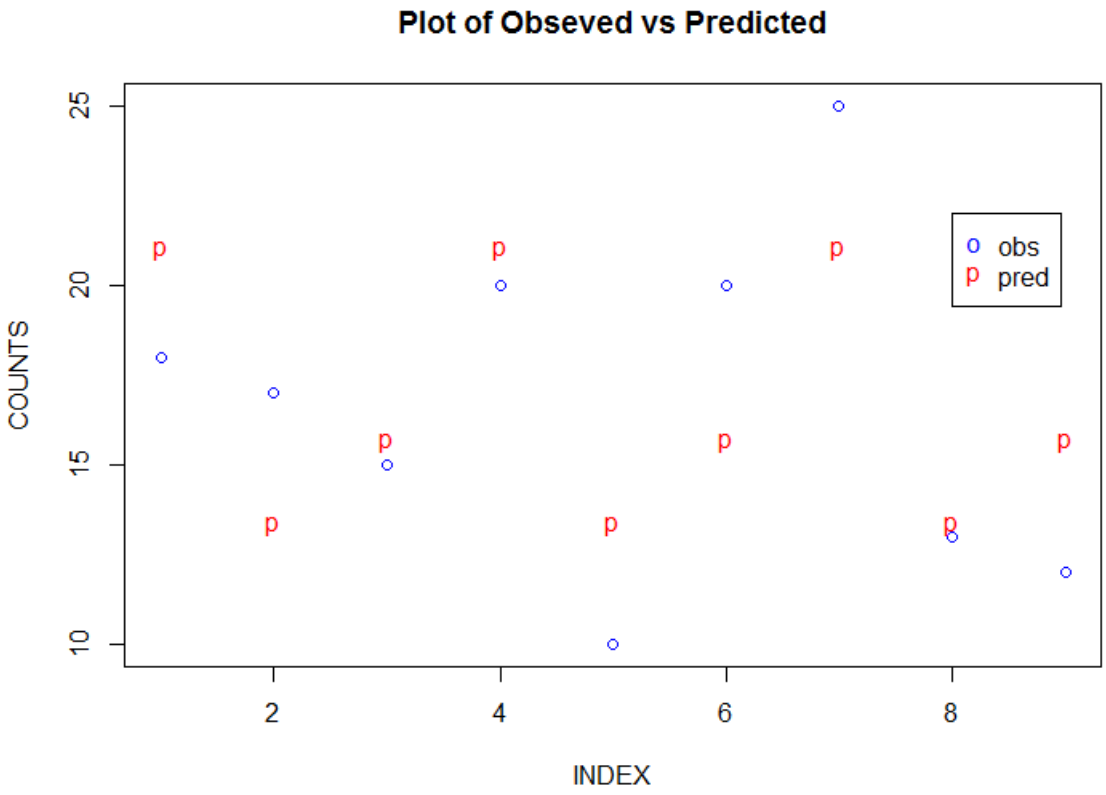
```
     res.deviance     df           p
[1,]     5.129141     4   0.2743016
```

- **P value is low but not low enough to be considered to be a bad fit.(the p-value is above 0.05)**
- **Loss of 4 degrees of freedom from 8 to 4.**

***http://statweb.stanford.edu/~jtaylo/courses/stats306b/restricted/notebooks/quasilikelihood.pdf

PLOT of FITTED VALUES vs ACTUAL Values

```
plot(counts,col="Blue",xlim = range(1:9),main="Plot of Obseved vs
Predicted",ylab="COUNTS",xlab="INDEX")
points(glm.D93$fitted,pch="p",col="red")
legend(8,22,c("obs","pred"),pch=c("o","p"),col=c("blue","red"))
```



Residuals

```
residuals(glm.D93)
```

1	2	3	4	5	6	7	8	9
-0.67124923	0.96272360	-0.16964662	-0.21998507	-0.95552353	1.04938637	0.84715368	-0.09167147	-0.96656372

- **Residual** values obtained were pretty high

6.Newton vs Fisher vs glm

Parameters	Fisher Scoring	Newton-Raphson	glm
intercept	3.044522	3.042874381	3.044522
outcome2	-0.45426	-0.45372863	-0.454255
outcome3	-0.29299	-0.292460482	-0.292987
treatment2	-4.78E-18	0.000904816	1.338E-15

treatment3	-4.78E-18	0.000904816	1.421E-15
Iterations	5	60	4

Estimated Fisher MODEL

$$log(\mu_i) = 3.044522 - 0.45426\beta_1 - 0.29299\beta_2 - (4.78E-18)\beta_3 - (4.78E-18)\beta_4$$

Estimated Newton MODEL

$$log(\mu_i) = 3.042874381 - 0.45372863\beta_1 - 0.292460482\beta_2 - (0.000904816)\beta_3 - (0.000904816)\beta_4$$

Estimated GLM MODEL

$$log(\mu_i) = 3.044522 - (0.454255)\beta_1 - 0.292987\beta_2 - (1.338\text{E}-15)\beta_3 - (1.421\text{E}-15)\beta_4$$

7. Observations

- Almost similar coefficents for **outcome and intercept** for Newton, Fisher and glm
- '**treatment**' is insignificant in all the three cases. Due to very low magnitude of coefficents
- Number of iterations minimum**(4)** in **glm** and maximum in **Newton Raphson(60).**
- This makes **glm** the most efficient and **Newton Raphson** the least.
- But the time taken measured through **system.time()** was neglibible in the three cases
- To make a good comparsion of efficiency of **Newton vs Fisher vs glm** we need more data points. In this can we can say they Newton Raphson took more steps(60) to converge.
- **Residual** values obtained were pretty high

PART B

PART a)

1)First we define **theta** and **x**

```
## theta <- seq(-pi, pi,0.01)
## x <- c(3.91, 4.85, 2.28, 4.06, 3.70, 4.04, 5.46, 3.53, 2.28, 1.96, 2.53, 3.88,2.22,
## 3.47, 4.82, 2.46, 2.99, 2.54, 0.52, 2.50)
```

The following distribution has the density function:

$$f(\boldsymbol{x};\boldsymbol{\theta}) = \frac{\mathbf{1} - \boldsymbol{cos}(\boldsymbol{x_i} - \boldsymbol{\theta})}{\mathbf{2} * \boldsymbol{pi}}$$

$$for\ 0 \leq x \leq 2\pi, -\pi < \theta < \pi$$

Since the observations are **iid** according to f, the log-likelihood function is

$$l(\theta) = -nlog2\pi + \sum_{i=1}^{n} log\{1 - cos(x_i - \theta)\}$$

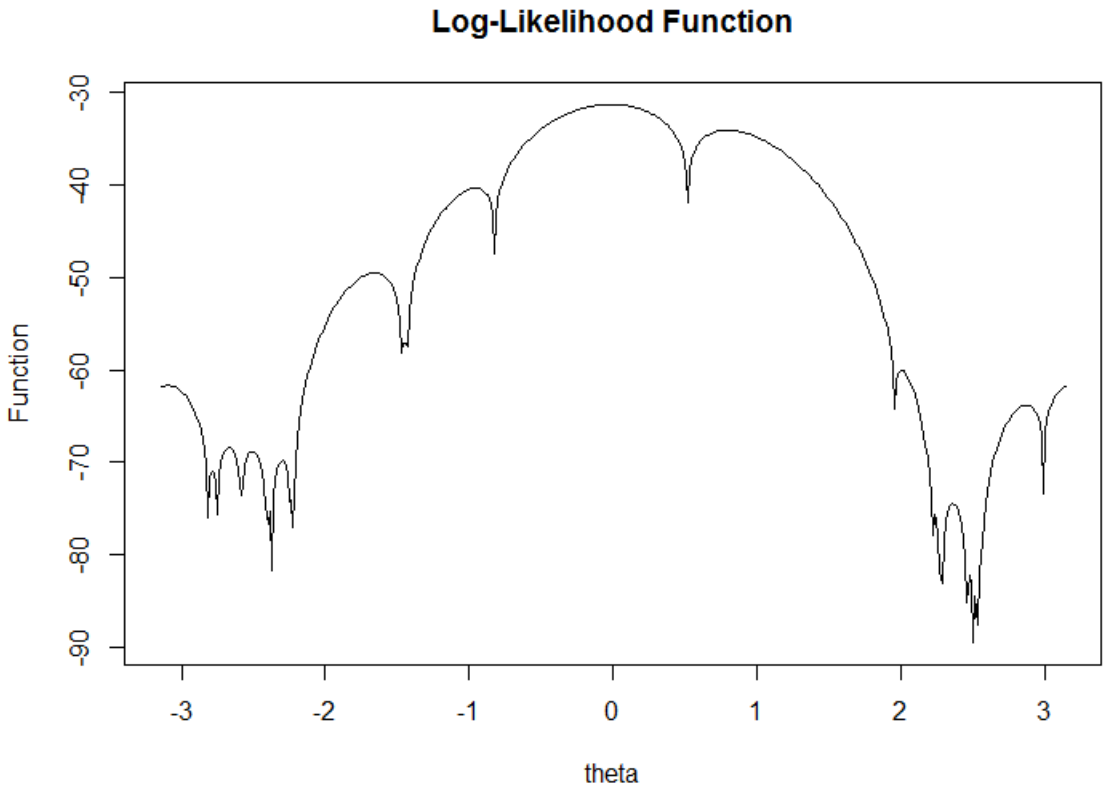
Code for implementing the log-likelihood function

```
## log.lik <- function(x, theta) {-20 * log(2*pi) + sum(log(1-cos(x-theta)))}
## plot(theta, sapply(theta, log.lik, x=x), typ='l')
```

Easier Way of plotting: Using Vectorize

```
## vec = Vectorize(log.lik,"theta")
## plot(theta, vec(theta,x))
```

The plot of the log-likelihood is given . We see many local maximas so the optimization routine will be highly sensitive to the starting point.



PART b

The method-of-moments estimator ~~θ~~ is found by setting the mean function equal to:

μ(θ) = ∫ x * f(x) dx

The integral is equal to the sample mean $\bar{X}$ and solving for θ. First:

$$\mu(\theta) = \pi - \frac{1}{2\pi}\int_0^{2pi} cos(x-\theta)dx = \pi - sin\theta$$

$$\pi - \frac{1}{2\pi}\int_0^{2pi} x * cos(x-\theta)dx$$

Using integration by parts on the right-hand side above, we get:

$$\int_0^{2pi} cos(x-\theta)dx =$$

$$= x * sin(x-\theta) - \int_0^{2pi} xsin(x-\theta) = 2pi * sin(2pi-\theta)$$

We know that

sin(−x) = sin(x) and sin(2π+x) = sin(x).

Putting everything together, we have

$$\mu(\theta) = \pi - \frac{1}{2pi}\int_0^{2pi} x * cos(x-\theta)dx = \pi - sin\theta$$

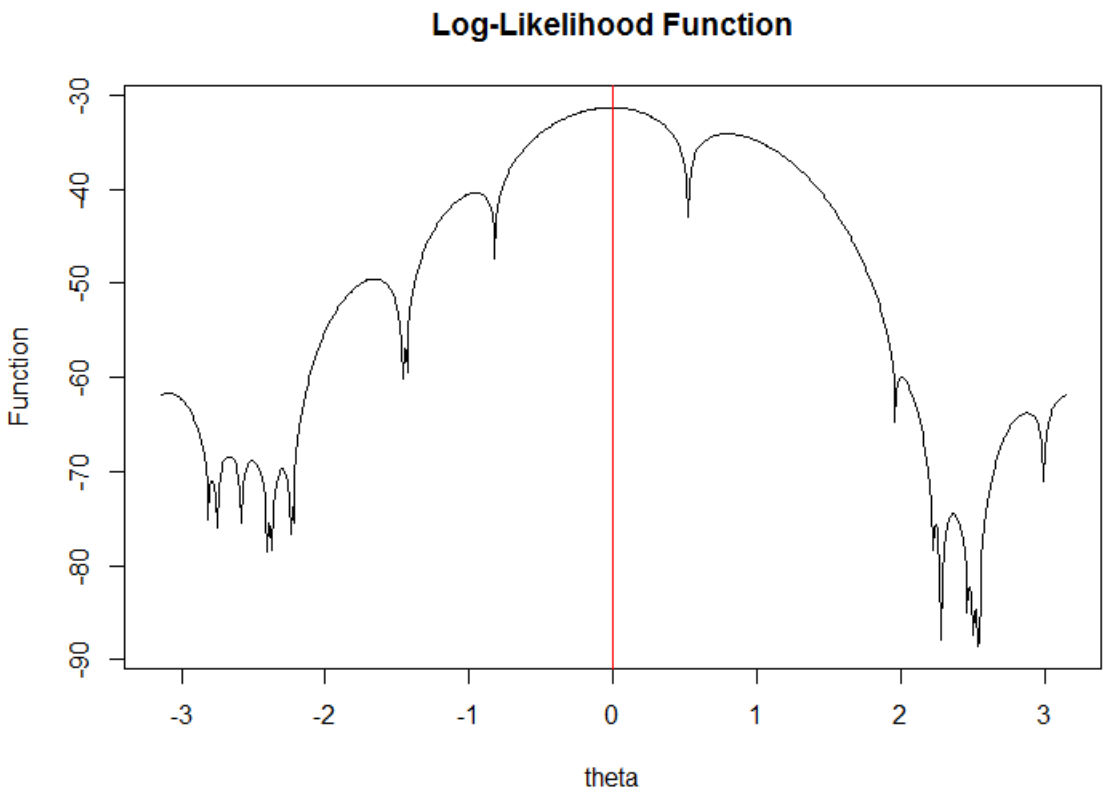
Now, the method of moments estimator ˜θ is the solution of μ(θ) = X. In particular

$$\mu(\theta) = \bar{X} \Rightarrow \pi - sin\theta = \bar{X} \Rightarrow \theta = arcsin(\pi - \bar{X}).$$

$$\boldsymbol{\theta = arcsin(\pi - \acute{X})}$$

X=3.2 $\boldsymbol{arcsin(\pi - 3.2) = 0.0584406061 rad}$

.



PART c

To implement Newton's method, we will also need the first and second derivatives of the log likelihood function:

 # First derivative of log-likelihood function

$$\frac{sin(x_i - \theta)}{1 - cos(x_i - \theta)}$$

$$\partial\, l / \partial\, \theta = \sum_{i=1}^{n}$$

```
## first_derivative <- function(theta){return (-sum(sin(x-theta)/(1-cos(x-heta))))}
```

 # Second derivative of log-likelihood function

$$\frac{1}{1 - cos(x_i - \theta)}$$

```
## second_derivative<- function(theta){return (-sum(1/(1-cos(x.i-theta))))}}
```

Newton's method is used to find the MLE for θ.
Suggested starting value for theta= Answer from PART B->–0.0584

Ways to find starting value(if we don't have one):

- **optimize()** is devoted to one dimensional optimization problem.
- **optim(), nlm(), ucminf() (ucminf)** can be used for multidimensional optimization problems.
- **nlminb()** for constrained optimization.
-

***optimize* function is the best in our case.**

```
optimize(log.lik2, c(-pi, pi), tol=1e-8, maximum=T)
```

Newton Raphson Method

The basic idea behind Newton Raphson comes from Taylors Expansion from where we get the equation:

$$x_2 = x_1 - \left(\frac{f(x_2)}{f'(x_2)}\right)$$

```
We do this until
   1. Maximum iteration reached
   2. We reach the tolerance level(eps) for the root.
newton <- function(f, f.derivative, x0, eps, maxter){
x1 <- x0 - f(x0)/f.derivative(x0)
ITERATION <- 1
while(abs(x1 - x0) > eps & ITERATION < maxter){
  x0 <- x1
  x1 <- x0 - f(x0)/f.derivative(x0)
  ITERATION <- ITERATION + 1
  }
   return (x1)
}
```

1. **Code for staring value =-0.0584:**

```
newton(first_derivative, second_derivative, -0.0584, 1e-6, 100)
```

OUTPUT:

```
[ITERATION] 1    -0.01197187
[ITERATION] 2    -0.011972
```

Estimate of theta for staring value =-0.0584

```
[1] -0.011972
```

2. **Code for staring value =-2.7**

```
newton(first_derivative, second_derivative, -2.7, 1e-6, 100)
```

OUTPUT:

```
[ITERATION] 1    -2.666794
[ITERATION] 2    -2.6667
[ITERATION] 3    -2.6667
```

Estimate of theta for staring value =-2.7

```
[1] -2.6667
```

3. **Code for staring value = 2.7**

```
newton(first_derivative, second_derivative, 2.7, 1e-6, 100)
```

OUTPUT:

```
[ITERATION] 1    2.877549
[ITERATION] 2    2.873184
[ITERATION] 3    2.873095
[ITERATION] 4    2.873095
```

Estimate of theta for staring value = 2.7

```
[1] 2.873095
```

PART d

200 equally spaced values between –pi and pi

```
## starting_value=seq(-pi,pi,length.out=200)
```

Find 200 roots

```
## thetas=NULL
## for(i in 1:200)
## {  thetas[i]=newton(first_derivative, second_derivative,starting_value [i], 1e-6, 100)
## }
```

Combine roots and starting values for creating table

```
## roots200=cbind(starting_value,thetas)
```

S.N0	starting value	ending value	Roots	S.N0	Starting Value	ending value	Roots
1	-3.1415927	-2.825854698	**-3.093091735**	**11**	-0.80513179	0.48939383	**-0.011972**
2	-2.7942809	-2.731133312	**-2.786166763**	**12**	0.520967626	0.520967626	**0.52193358**
3	-2.7311333	-2.60483813	**-2.666699927**	**13**	0.552541421	1.941788424	**0.79060002**
4	-2.5732643	-2.415395357	**-2.507614267**	**14**	1.97336222	2.194378788	**2.00364478**
5	-2.3838216	-2.383821561	**-2.38819785**	**15**	2.225952584	2.257526379	**2.23621722**
6	-2.3522478	-2.257526379	**-2.297256219**	**16**	2.289100175	2.446969152	**2.36071812**
7	-2.2259526	-2.225952584	**-2.232166637**	**17**	2.478542948	2.478542948	**2.47537287**
8	-2.1943788	-1.468181491	**-1.658280516**	**18**	2.510116743	2.510116743	**2.51358999**
9	-1.4366077	-1.447472893	**-1.447472893**	**19**	2.541690539	2.983723676	**2.87309459**
10	-1.4050339	-0.868279377	**-0.953336315**	**20**	3.015297472	3.141592654	**3.19009358**

For 200 values between −π and π, we get 20 different solutions.

PART e

We look at the results at the previous step we see that in some cases a different root was found between consecutive values.

Starting Value	Theta
-2.82555	-3.093092
-2.794281	-2.786167

```
## tol=1e-1
## r=rep(0,100)
## for (i in 1:100)
## {
## r[i]= newton(first_derivative, second_derivative, (s[11]+i*tol), 1e-6, 100)
## value=s[11]+i*tol
##  if (theta[11]!=s[11])
## {
##   break;
##   print(i)
## }
##   }
```

We add a value of 10^{-1} in search to the starting value and try to find a different root.

We find:

Starting Value	Theta
-2.725855	-2.6667

How close are the starting values?

$$(-2.794281) - (-2.725855) = -0.0068426$$

-0.0068426 is a small value the function is very sensitive to the starting value. We could have missed many potential solutions since we looked at the 200 values between –p1 and pi had we considered more values we might get more closer values.