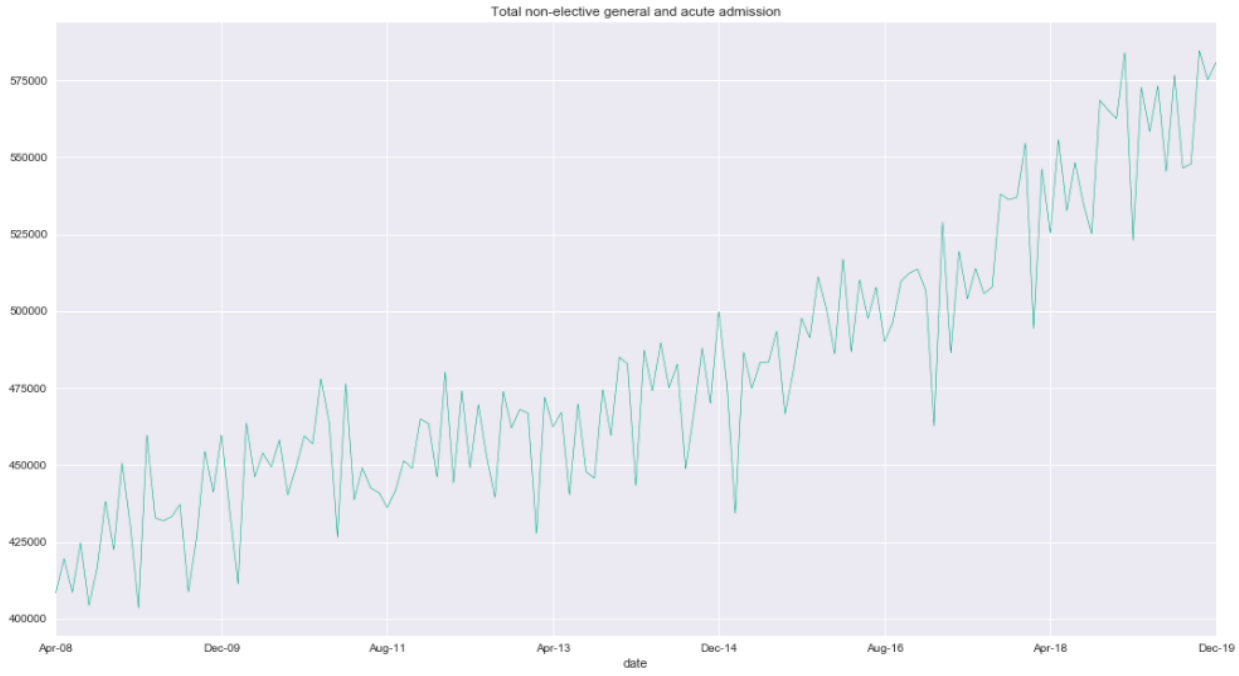
STAT8178: Statistical Computing

Student: Francesco Palermo
Id: 45539669

Question1: MLE with Newton-Raphson

The aim of this exercise is to fit a seasonal model via maximum likelihood estimation. In particular, we want to model the "total non-elective general and acute admission" from the excel file provided.
First at all, a time series graph is shown below in order to have a better understanding of data.



The time series does not come from a stationary process because there is a positive trend throughout the years.
The monthly admission counts (the observed data) can be model with Poisson distribution,

$y_i \sim \text{Poisson}\ (\mu_i)$

where,

$$\mu_i = \exp\left(\beta_1 + \beta_2 i + \beta_3 \cos\left(\emptyset_i\right) + \beta_4 \sin\left(\emptyset_i\right)\right) \qquad \textit{(model 1)}$$

a) In order to describe the matrix X, we simply need to rewrite the model 1 with a more convenient model 2, by using a matrix that describes the coefficient of betas.

μ_i= exp $(\sum_{j=1}^{J} x_{ij}\,\beta_j)$ *(model 2)*

With J=4.

Therefore, with N = 141:

$$X_{N*4} = \begin{bmatrix} x_{11} & \cdots & x_{14} \\ \vdots & x_{ij} & \vdots \\ x_{N1} & \cdots & x_{N4} \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cos(\emptyset_1) & \sin(\emptyset_1) \\ 1 & 2 & \cos(\emptyset_2) & \sin(\emptyset_2) \\ 1 & 3 & \cos(\emptyset_3) & \sin(\emptyset_3) \\ . & . & . & . \\ . & . & . & . \\ 1 & n & \cos(\emptyset_N) & \sin(\emptyset_N) \end{bmatrix}$$

b) The probability mass function of the independent and identical distributed y_i is the following

$$f_{y_i}(y_i) = \frac{{\mu_i}^{y_i}\, e^{-\mu_i}}{y_i!} \qquad y_i = 0,1,2,.... \qquad \mu_i > 0$$

Let's first calculate the <u>likelihood function</u> as the joint pmf of the observed sample values. Since we are dealing with independent sample values, the likelihood function is a product:

$L = \prod_{i=1}^{N} f_{y_i}(y_i)$

Therefore,

$$L(\mu_i|y_i) = \prod_{i=1}^{N} f_{y_i}(y_i) = \prod_{i=1}^{N} \frac{{\mu_i}^{y_i}\, e^{-\mu_i}}{y_i!} = \prod_{i=1}^{N} {\mu_i}^{y_i} \prod_{i=1}^{N} e^{-\mu_i} \prod_{i=1}^{N} \frac{1}{y_i!} = e^{-\sum_{i=1}^{N}\mu_i} \prod_{i=1}^{N} {\mu_i}^{y_i} \prod_{i=1}^{N} \frac{1}{y_i!}$$

Since we are dealing with a tedious function that would be hard to differentiate, we are transforming the above function through logarithms.

The following expression denotes the log likelihood function.

$$\log L(\mu_i|y_i) = \log\left(e^{-\sum_{i=1}^{N}\mu_i}\prod_{i=1}^{N}\mu_i{}^{y_i}\prod_{i=1}^{N}\frac{1}{y_i!}\right) = \log\left(e^{-\sum_{i=1}^{N}\mu_i}\right)+$$
$$+\log\left(\prod_{i=1}^{N}\mu_i{}^{y_i}\right)+\log\left(\prod_{i=1}^{N}\frac{1}{y_i!}\right) = -\sum_{i=1}^{N}\mu_i + \sum_{i=1}^{N}y_i\log(\mu_i)+$$
$$-\log(y_i!).$$

Here we replace μ_i with the espression found in *(model 1).* Hence,

$$\log L(\beta_j|y_i) = -\sum_{i=1}^{N} e^{\left(\sum_{j=1}^{J}x_{ij}\beta_j\right)} + \sum_{i=1}^{N}y_i\log\left(e^{\left(\sum_{j=1}^{J}x_{ij}\beta_j\right)}\right)+$$
$$-\log(y_i!) = -\sum_{i=1}^{N} e^{\left(\sum_{j=1}^{J}x_{ij}\beta_j\right)} + \sum_{i=1}^{N}y_i\sum_{j=1}^{J}x_{ij}\beta_j - \log(y_i!).$$

c) Next *model 2* is differentiated with respect to β_j. We rely on the fact that:

$$\frac{de^{f(x)}}{dx} = e^{f(x)}f'(x)$$

Hence,

$$\frac{d\mu_i}{d\beta_j} = e^{\left(\sum_{j=1}^{J}x_{ij}\beta_j\right)}x_{ij} = \mu_i\, x_{ij}$$

d) Now I will show the steps to find the derivate of the log likelihood function found in point b.

$$\frac{d\log L(\beta_j|y_i)}{d\beta_j} = \frac{d}{d\beta_j}\left(-\sum_{i=1}^{N} e^{\left(\sum_{j=1}^{J}x_{ij}\beta_j\right)} + \sum_{i=1}^{N}y_i\sum_{j=1}^{J}x_{ij}\beta_j - \log(y_i!)\right)$$
$$= -\sum_{i=1}^{N} e^{\left(\sum_{j=1}^{J}x_{ij}\beta_j\right)}x_{ij} + \sum_{i=1}^{N}y_i\,x_{ij} = \sum_{i=1}^{N}x_{ij}\left[y_i - e^{\left(\sum_{j=1}^{J}x_{ij}\beta_j\right)}\right] =$$
$$= \sum_{i=1}^{N}x_{ij}\left[y_i - \mu_i\right]$$

Therefore, the gradient is $\frac{d\log L(\beta_j|y_i)}{d\beta_j} = \sum_{i=1}^{N}x_{ij}\left[y_i - \mu_i\right]$

e) For the purpose of finding the Hessian of the log likelihood function, another differentiation with respect to β_j is required.

$$\frac{d^2 \log L(\beta_j|y_i)}{d\beta_j d\beta_k} = \frac{d}{d\beta_k}(\sum_{i=1}^{N} x_{ij}\, y_i - \sum_{i=1}^{N} x_{ij}\, \mu_i) =$$

$$\frac{d}{d\beta_k}(\sum_{i=1}^{N} x_{ij}\, e^{\left(\sum_{j=1}^{J} x_{ij}\beta_j\right)}) = -\sum_{i=1}^{N} x_{ij}\, x_{ik} e^{\left(\sum_{j=1}^{J} x_{ij}\beta_j\right)}) =$$

$$-\sum_{i=1}^{N} x_{ij}\ x_{ik}\, \mu_i$$

Therefore, the Hessian is $\frac{d^2 \log L(\beta_j|y_i)}{d\beta_j d\beta_k} = -\sum_{i=1}^{N} x_{ij}\ x_{ik}\, \mu_i$

f) Let β, y and μ be the column vectors and let M be a diagonal matrix with the value $\mu_1, \mu_2, \ldots\ldots, \mu_n$. That is,

$$\beta_{J*1} = \begin{pmatrix} \beta_1 \\ \cdot\cdot \\ \beta_J \end{pmatrix}, y_{N*1} = \begin{pmatrix} y_1 \\ \cdot\cdot \\ y_N \end{pmatrix}, \mu_{N*1} = \begin{pmatrix} \mu_1 \\ \cdot\cdot \\ \mu_N \end{pmatrix}, M_{N*N} = \begin{pmatrix} \mu_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \mu_N \end{pmatrix}$$

I will express the vector g as the gradient vector and H as the Hessian matrix.

$g_{J*1} = X^T_{J*N}\,(y_{N*1} - \mu_{N*1})$ (Simplified g = X' (y - μ))

$H_{J*J} = (-X^T_{J*N})(\,M_{N*N})(\,X_{N*J})$ (Simplified H = -X' M X)

g) My initial starting values for the parameters β are the following:

$$\beta_{4*1} = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}$$

This vector will be the first initial value for the Newton – Raphson algorithm. This original guess is meant to be those values that are going to minimize/maximize the log likelihood function.

h) Finally, we can show the Matlab script. Part of the script and the results will be described after the screenshot. However, the original file will be attached in the submission (Newton_Raphson.m)

```
%Defining a function that accept a filename path, a column where we want to
%read and the initial vector and return the betas estimates

function [beta,k,eucl] = Netwon_Raphson(file_path,column,init)
    %A-Reading the observed values
    file_all=xlsread(file_path);
    Yi=file_all(:,column);

    %B-Creating the X matrix
    x1 = ones(141,1);
    x2 = [1:141]';
    x3 = [cosd(30*x2)];
    x4 = [sind(30*x2)];
    X=[x1,x2,x3,x4];

    %C-transposing the initial guess estimate and assign to beta
    beta=init';
    %eucl will measure the distance between 2 successive betas vectors
    eucl=4;
    % k will count the number of iteration needed to find the best
    % estimates
    k=0;

    %D-do this operation until the beta vector does not change substantially
    while (eucl > 3e-16)
            %Creating mu and M (diag(mu)
            mu= (exp(X*beta));
            M = diag(mu);
            %gradient
            g= X' * (Yi - mu);
            %Hessian
            H=-(X'*(M*X));
            %update beta values
            betak=beta; %this vector keeps the value of bj's at iteration k

            beta = beta - (H\g);
            k=k+1;
            eucl=norm(betak - beta);
    end
end
```

Before going through the required points, let's evaluate the function definition in the following page.

```
function [beta,k,eucl] = Netwon_Raphson(file_path,column,init)
```

The following function takes 3 input parameters and return 3 output values back to the user. In particular, the input parameters are:

- *file_path=* It describes the path where we can find our given excel file. It is a string.
- *column=* It specifies the column which we are interested to model. It is a numeric value, so the column k will be inserted as 11.
- *init =* This is the starting value for the parameter vector β. It is a numeric vector.

On the other hand, this function returns these following values:

- *beta =* This is the best estimate found by the Newton-Raphson algorithm. It is a numeric vector.
- *k =* It tells us how many iterations the algorithm needs in order to find the best estimate.
- *eucl =* This is the Euclidian distance between $\beta_{(k)}$ and $\beta_{(k-1)}$

Now, I can answer the assignments points as follow (A-E from the provided screenshot)

A. The function *xlsread* allows us to read from an external excel file. For convenience, I first read all the spreadsheet (*file_all*) and then selected the *column* which I am interested in.

B. The matrix X has been created by building individual vectors first, and then concatenate those vectors into a single matrix.

C. In my application, the initialization of the vector β is actually a parameter that the user input during the function call *(init).* However, this input parameter will be transposed and assigned to *beta.*

D. Let us evaluate the statements inside the while loop first. All the elements which are required to update *beta* are created and updated each iteration (these elements have been deeply discussed in previous points).

Of particular importance are the following lines of code

```
%update beta values
betak=beta; %this vector keeps the value of bj's at iteration k

beta = beta - (H\g);
k=k+1;
eucl=norm(betak - beta);
```

Here, *betak* is responsible of "saving" the value of the vector β at iteration k before being updated. In the following line *beta* is updated by using the N-R algorithm method:

$$\beta_{(k+1)} = \beta_{(k)} + \frac{g}{H}$$

The last two lines of code are simply accountable of taking track of the number of iterations of the loop and calculate the distance between $\beta_{(k)}$ and $\beta_{(k-1)}$

In conclusion, although I decided to slightly change the convergence criteria, the final result is conceptually the same. Instead of *"perform iterations of the N-R algorithm method until the estimated value of μ are no longer changing very much"* I have done the same but with vector β instead.

The statements inside the loop are going to perform until the Euclidian distance is very small and so there is pretty much no variation between $\beta_{(k)}$ and $\beta_{(k-1)}$

```
while (eucl > 3e-16)
```

E. Two function calls will be shown in order to better understand the logic of the algorithm.
The following page illustrates the output from Matlab and screenshots have been taken.

```
COMMAND WINDOW
>> [betas,iteration,distance]=Netwon_Raphson('MAR_Comm-Timeseries-Dec-19-REVISED-Apr-18-to-Nov-19-9tun8',11,[1 2 3 4])

betas =

   12.9345
    0.0020
    0.0017
   -0.0087


iteration =

   269


distance =

   2.1974e-16
```

The algorithm takes 269 iterations to find the above values of betas when the initial guess is what provided in point g.

Now, let us suppose we had initialised the vector β with the actual found estimates (or really close values). We are now showing how faster the algorithm would have been if that happened.

```
>> [betas,iteration,distance]=Netwon_Raphson('MAR_Comm-Timeseries-Dec-19-REVISED-Apr-18-to-Nov-19-9tun8',11,[12 0 0 0])

betas =

   12.9345
    0.0020
    0.0017
   -0.0087


iteration =

     8


distance =

   2.0994e-16
```

The above result shows how important the initial guess might be Only 8 iterations are needed.

Question2: Posterior distribution after a single coin toss

a) We all know that the probability of getting tail in a single toss is 0.5. However, since we cannot see the coin, we do not know whether it is a real coin or it is biased. All we know is the outcome of this experiment and it has shown tail.
We have to model this current state of believe through a model distribution.

Therefore, the probability of getting tail in single toss is a value between 0 and 1. The easiest and more natural distribution is the Uniform distribution.

The prior distribution can be described as:

Let π = "Probability of getting tail on a single coin toss"

$\pi \sim U\ (0,1)$

$f_\pi(\pi) = \frac{1}{b-a} = 1 \qquad 0 \leq \pi \leq 1$

This prior information is:

- Uninformative: Although this definition contrasts the concept of the prior distribution as previous knowledge of the parameter π, we are classifying this prior as uninformative because it assigns equal probabilities to all possibilities.
- Flat: This is due to the fact that the pdf is constant over the parametric space.
- Proper: The prior distribution is a proper probability distribution. In fact, we can simply show that the pdf integrates to 1 over the parametric space

$$\int_0^1 f_\pi(\pi)\ d\pi = \int_0^1 1\ d\pi = [\pi]_0^1 = (1\text{-}0) = 1$$

b) In order to create a histogram of the posterior distribution we are going to use a *stochastic approach* through the use of JAGS.
By Bayes' theorem, the posterior distribution can be written as

$$p\ (\pi|x) = \frac{p\ (x|\pi)\ p\ (\pi)}{p\ (x)} \propto p\ (x|\pi)\ p\ (\pi) \propto L(\pi;x)\ p\ (\pi)$$

where:

- L(π;x) is likelihood function
- p (π) is the prior distribution

Let the prior distribution be:

p (π) ~ U (0,1)

Tossing a coin can be described with Bernoulli distribution, which likelihood function is the following:

$L(\pi;x) = \pi^x \ (1 - \pi)^{1-x}$

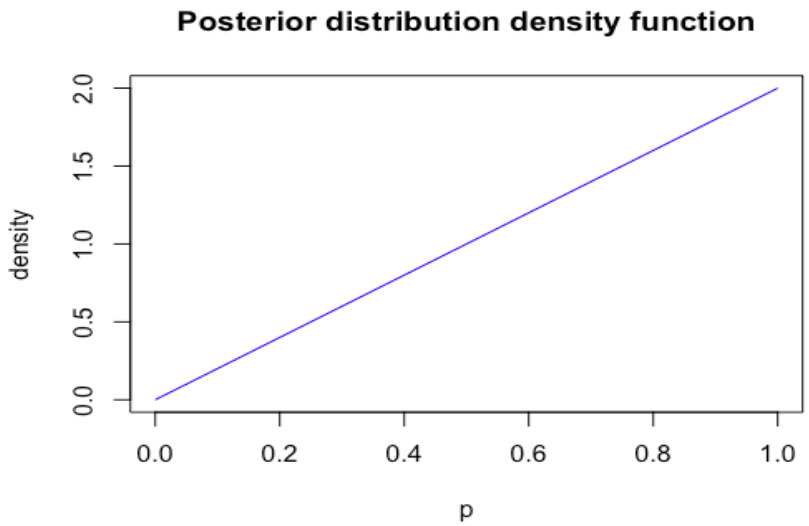
Finally, the posterior distribution is:

$p(\pi|x) = \pi^x (1 - \pi)^{1-x} \sim$ beta (x+1, x)

Since the posterior distribution is a well-known distribution we can simply draw samples from that and plot it.
In addition of that, we observed one success in one trial.
(x=k=1 and n=1)

The following line of code in R will be able to plot the density of the posterior distribution.

```
p = seq(0,1, length=100)
plot(p, dbeta(p, 2, 1), ylab="density", type ="l", col=4,title="bb",main="Posterior distribution density function")
```



Next, I illustrate how an appropriate Bayesian inference method can replicate the above graph. It is important to remind that most of the time, we are not so lucky to find a closed form expression for the Bayesian mean estimator (and therefore a well-known curve for the posterior distribution).

A very similar approach has been taken from slide 33 of the Bayesian Analysis lecture. I simply let k=1, n=1 and N=1.

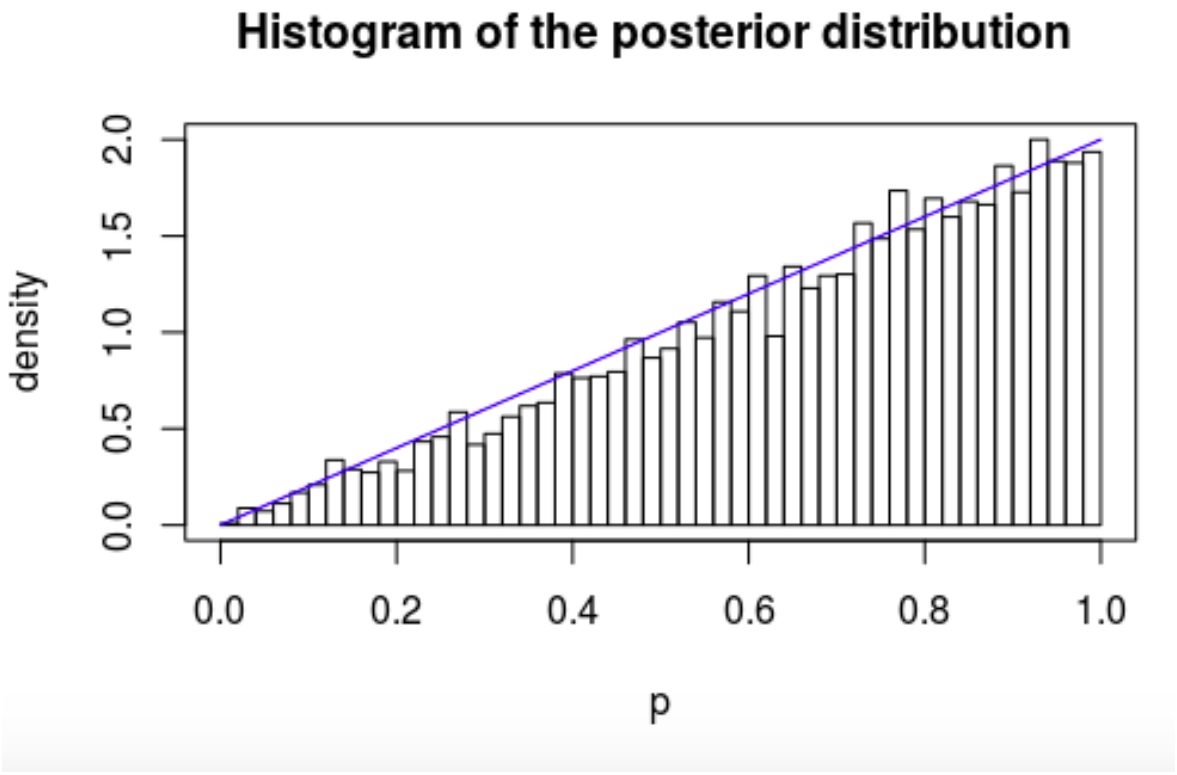
The following model codes have been saved as separate txt file and the remained R code has been developed on R studio.cloud (file attached: R_Jags.R)

```
model
{
  for (i in 1:N) {
  k[i] ~ dbin(p, n[i])
 }
 p~ dunif(0, 1)
}
```

```
#Compile/initiate the model
model=jags.model(file="Model_model.R",data=list(k=1,n=1,N=1),inits=list(p=0.5),n.chain=1)
#Update models as burn-in
update(model,n.iter=6000)
#obtain some stationary MCMC samples
model.out <-coda.samples(model,c("p"),n.iter=6000)
#summary the results and create the histogram
lst=model.out[1]
lst2 <- unlist(lst, use.names = FALSE)
hist(lst2,main="",xlab="",ylab="",breaks = 40,yaxt='n',xaxt='n', ann=FALSE)
#overlap graph
par(new=TRUE)
p = seq(0,1, length=1000)
plot(p, dbeta(p, 2, 1), ylab="density", type ="l", col=4,main="Histogram of the posterior distribution ")
```

This snippet of code simply:

- Compile and initiate the model
- Update models as burn-in
- Obtain some stationary MCMC samples
- Plot the histogram of the posterior distribution



According to the graph, it is clear to see how the produced histogram is very close to the probability density curve (Gamma distribution).
This emphasis the power of the Stochastic Approximation.