Statistical Methods for Data Science Laboratory Computational Statistics (2015/2016)

Assignment # 1

Paolo Tamagnini

April 2016

1 Exercise 1

Definition of the model:

$$X_1, X_2, ..., X_n \quad X_i \sim Bern(p) \quad p \in P = [0,1]$$

$$n = 10 \quad \sum_{i=1}^{n} x_i = 3$$

$$Prob(X = x) = p^x (1-p)^{1-x} \quad x \in X : \{0,1\}$$

Computation of the likelihood function:

$$L_x(p) = \prod_{i=1}^{n} p^x \, (1-p)^{1-x} = p^{\sum x} \, (1-p)^{n-\sum x}$$

$$L_x(p) \propto p^{a_{lik}-1}(1-p)^{b_{lik}-1} = g(p, h_{lik} = (a_{lik}, b_{lik}))$$

$$\begin{cases} a_{lik} = 1 + \sum_{i=1}^{n} x_i = 4 \\ b_{lik} = 1 + n - \sum_{i=1}^{n} x_i = 8 \end{cases}$$

We use for the prior distribution $\pi(p)$ the same distribution of the Likelihood function $L_x(p)$.

$$\pi(p) \propto p^{a_{prior}-1}(1-p)^{b_{prior}-1} = g(p, h_{prior} = (a_{prior}, b_{prior}))$$

We choose the class G of prior distributions for p as conjugate for our Bernoulli model.

$$G : g(p, h(a, b)) \; = \; p^{a-1}(1-p)^{b-1}$$

This will mean we can use G also for our posterior distribution $\pi(p/x)$.

$$\pi(p) \in G \;\; \Rightarrow \;\; \pi(p/x) \in G$$

$$\pi(p/x) \propto p^{a_{post}-1}(1-p)^{b_{post}-1} = g(p, h_{post} = (a_{post}, b_{post}))$$

We can clearly see that:

$$g(p, h(a, b)) \; = \; p^{a-1}(1-p)^{b-1} \propto Beta(a, b)$$

$$\pi(p) \propto Beta(a_{prior}, b_{prior})$$

$$\pi(p/x) \propto \pi(p)L_x(p) = p^{\sum x} (1-p)^{n-\sum x} \; p^{a_{prior}-1}(1-p)^{b_{prior}-1} \; =$$

$$= p^{\sum x + a_{prior}-1} (1-p)^{n+b_{prior}-\sum x -1} =$$

$$= p^{a_{post}-1}(1-p)^{b_{post}-1} \propto Beta(a_{post}, b_{post})$$

$$\begin{cases} a_{post} = a_{prior} + \sum_{i=1}^{n} x_i \\ b_{post} = n + b_{prior} - \sum_{i=1}^{n} x_i \end{cases}$$

It's now the time to choose (a_{prior}, b_{prior}).
At first we do not know anything on how p should reflect the result of the n trials. Indeed $\pi(p)$ has to be chosen without using the values x_i and this means that it should be equally distributed for any value in $P = [0, 1]$. According to this it is legit to use a uniform distribution in $[0, 1]$ as prior distribution for p. This can be done by setting $(a_{prior}, b_{prior}) = (1, 1)$.

$$\pi(p) \sim Beta(1, 1) = U_{[0,1]}$$

Then it follows:

$$n = 10 \quad \sum_{i=1}^{n} x_i = 3$$

$$\begin{cases} a_{post} = 1 + 3 = 4 \\ b_{post} = 10 + 1 - 3 = 8 \end{cases}$$

$$\pi(p) \sim Beta(1,1)$$

$$\pi(p/x) \sim Beta(4,8)$$

$$E(p/x) = \frac{a_{post}}{a_{post}+b_{post}} = 0.333$$

$$Mode(p/x) = \frac{a_{post}-1}{a_{post}+b_{post}-2} = 0.3$$

$$\hat{p} = \frac{\sum_{i=1}^{n} X_i}{n} = 0.3$$

We can now use the prior distribution to make some Bayesian inference on p. What we could do, except computing expected value and mode, could be computing a confidence interval over the expected value of the posterior with a certain significance level α. We could make hypothesis testing as well and find if we can reject or not a certain statement H_0 about the value of p.

2 Exercise 2

On 27 seacows (dugongs) have been reported: length (Yi) and age (xi). We are interested in the following non-linear regression:

$$Y_i \sim N(\mu_i, \tau^2)$$

$$\mu_i = \alpha - \beta\gamma^{x_i}$$

Y_i (length): 1.80 1.85 1.87 ... 2.70 2.72 2.57
x_i (age): 1.00 1.50 1.50 ... 22.50 29.00 31.50
n = 27

We want to find the best parameter vector $\theta = (\alpha, \beta, \gamma, \tau^2)$ for our regression model. To do that we compute first our likelihood function $L_Y(\theta)$.

$$L_Y(\theta) = \prod_{i=1}^{n} f(Y_i/\theta) =$$

$$= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\tau^2}} \; exp\left\{-\frac{(Y_i-\alpha-\beta\gamma^{x_i})^2}{2\tau^2}\right\} =$$

$$= (\frac{1}{\sqrt{2\pi\tau^2}})^n \; exp\left\{-\frac{\sum(Y_i-\alpha-\beta\gamma^{x_i})^2}{2\tau^2}\right\}$$

To find a maximum of such function it's better to use a logarithmic scale, that means computing and use the log-likelihood function:

$$log(L_Y(\theta)) = \sum_{i=1}^{n} log(f(Y_i/\theta)) =$$

$$= -\frac{n}{2}log(2\pi\tau^2) - \frac{\sum(Y_i-\alpha-\beta\gamma^{x_i})^2}{2\tau^2}$$

We can use R to define this and compute the maximum likelihood estimator.

$$\hat{\theta}_{MLE} = (\hat{\alpha}_{MLE}, \hat{\beta}_{MLE}, \hat{\gamma}_{MLE}, \hat{\tau}^2_{MLE}) = argmax(L_Y(\theta))$$

```
loglike=function(teta){
  summa = 0
  for(i in 1:n){
    fi = dnorm(y[i],
                    mean = teta[1]-teta[2]*teta[3]^x[i],
                    sd = sqrt(teta[4]), log = TRUE)
                    summa = summa + fi }
  return(summa) }

startMLE = c(0.10,0.70,0.08,0.89)
```

```
MLE_ND=optim(startMLE, loglike,
              method="Nelder-Mead",control=list(fnscale=-1))
print(MLE_ND$par)
print(MLE_ND$value)
```

$$\hat{\theta}_{MLE} = (\hat{\alpha}_{MLE}, \hat{\beta}_{MLE}, \hat{\gamma}_{MLE}, \hat{\tau}^2_{MLE}) = (2.65806, 0.96351, 0.87145, 0.00806)$$

$$log(L_Y(\hat{\theta}_{MLE})) = 26.76327$$

The optimization used the method Nelder-Mead and it is relative to the starting point *startMLE*. To find the starting point that gives the best result, I used four cycles trying many different starting points.

```
bestsofarvalue = -10^6
for(i in (1/10)*1:100){
  for(j in (1/10)*1:100){
    for(k in (1/100)*1:100){
      for(h in (1/100)*1:100){
        startMLE = c(i,j,k,h)
        MLE_ND=optim(startMLE,
                     loglike,
                     method="Nelder-Mead",
                     control=list(fnscale=-1))
        if (MLE_ND$value > bestsofarvalue) {
          bestsofarvalue = MLE_ND$value
          bestsofar = MLE_ND
          bestStart = startMLE }}}}}
```

Another technique to find the best starting point is running the optimization multiple times, each time with *startMLE* equal to the result MLE found in the last optimization.
This will not stop until the log-likelihood result value doesn't change anymore of some ϵ.

```
bestsofarvalue = -10^6
valuenew = -10^6
valueold = -10^7
startMLE = c(1,10,0.001,10)
while(abs(valuenew - valueold)>10^(-10)){
  valueold = valuenew
  MLE_ND=optim(startMLE, loglike,
                method="Nelder-Mead",control=list(fnscale=-1))
  valuenew = MLE_ND$value
  startMLE = MLE_ND$par
  if (MLE_ND$value > bestsofarvalue) {
    bestsofarvalue = MLE_ND$value
    bestsofar = MLE_ND
    bestStart = startMLE }}
```

These strategies require a lot of computation time and they don't really improve the result in this case. Anyway it is good practice for experimenting optimization skills.

We can now compute the MAP: maximum at posteriori probability estimate. First let's choose our prior distributions.

$$\alpha \sim N(0, 10000)$$

$$\beta \sim N(0, 10000)$$

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

$$\tau^2 \sim /\Gamma(0.001, 0.001) = \frac{1}{Gamma(0.001, \frac{1}{0.001})}$$

Then our prior distribution for θ will be:

$$\pi(\theta) = \pi(\alpha)\pi(\beta)\pi(\gamma)\pi(\tau^2)$$

To find the the MAP, which is the mode of the posterior distribution, we compute the maximum of the function $\pi(\theta)L_x(\theta)$.

$$\hat{\theta}_{MAP} = (\hat{\alpha}_{MAP}, \hat{\beta}_{MAP}, \hat{\gamma}_{MAP}, \hat{\tau}^2_{MAP}) = argmax(\pi(\theta)L_x(\theta))$$

It follows the *R* code that computes the MAP over the logarithmic scale of the function.

```
posteriorFunction=function(teta){
  pi = dunif(teta[3],min = 0, max = 1, log = TRUE)
  pi = pi + dnorm(teta[1], mean=0, sd = 100, log = TRUE)
  pi = pi + dnorm(teta[2], mean=0, sd = 100, log = TRUE)
  pi = pi + log(1/dgamma(teta[4],shape= 0.001, scale = 1/0.001))
  summa = loglike(teta) + pi
  return(summa) }

startMAP = c(2.65806,0.96351,0.87145,0.00806)

MAP_ND=optim(startMAP, posteriorFunction,
               method="Nelder-Mead",control=list(fnscale=-1))
print(MAP_ND$par)
print(MAP_ND$value)
```

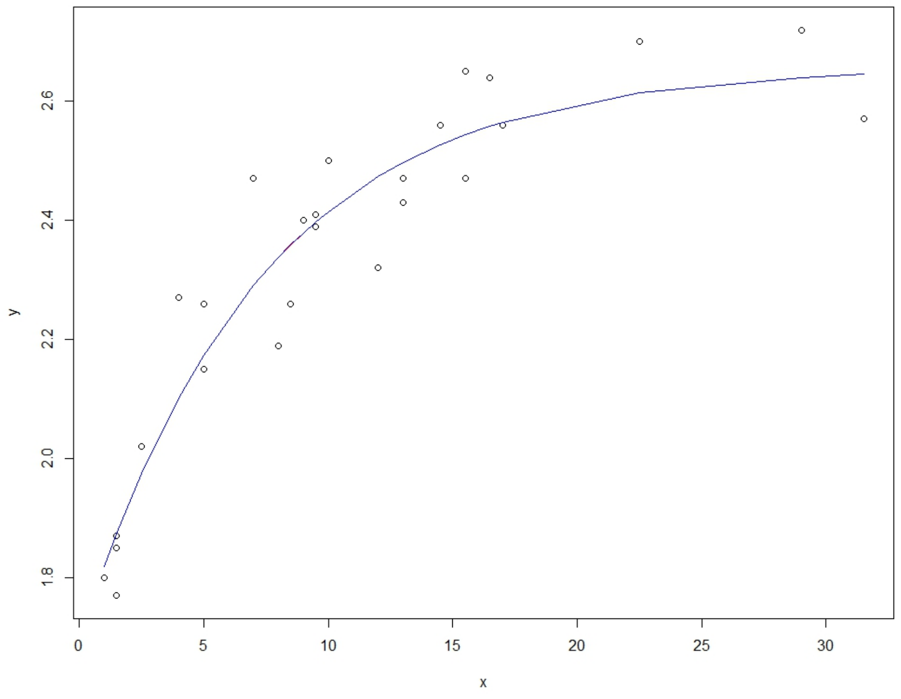
We set the starting point for our optimization equal to the MLE, but we can use the strategies for locating the best starting point here as well. Anyway our results are:

$$\hat{\theta}_{MAP} = (2.65756, 0.96342, 0.87124, 0.00870)$$

$$log(\pi(\hat{\theta}_{MAP})L_x(\hat{\theta}_{MAP})) = 17.8511$$

Even if the functions are really different, MAP and MLE are really alike and we cannot distinguish the relative curves in the following scatter plot:

```
tetaMLE = c(2.65806, 0.96351, 0.87145, 0.00806)
tetaMAP = c(2.65756, 0.96342, 0.87124, 0.00870)

plot(x, y)
lines(x, tetaMLE[1]-tetaMLE[2]*tetaMLE[3]^x, col = "red")
lines(x, tetaMAP[1]-tetaMAP[2]*tetaMAP[3]^x, col = "blue")
```



3 Exercise 3

The solution is as follows:

$$X \sim N(\mu, \sigma^2)$$

$$Y = e^X \;\Rightarrow\; X = log\, Y$$

$$Prob(Y \leq y) = Prob(e^X \leq y) = Prob(X \leq log\, y) =$$

$$= F_X(log\, y) = F_Y(y)$$

$$f_Y(y) = \tfrac{d}{dy} F_Y(y) = \tfrac{d}{dy} F_X(log\, y) = f_X(log\, y)\, \tfrac{d}{dy} log\, y =$$

$$= \tfrac{1}{\sqrt{(2\pi\sigma^2)}}\, \tfrac{1}{y}\, exp\left\{ -\tfrac{(log\, y - \mu)^2}{2\sigma^2} \right\}$$

So we fond that Y has a log-normal distribution.

$$Y \sim logN(\mu, \sigma^2)$$

4 Exercise 4

Let's compute now the expected value of Y.

$$E(Y) = E(e^X)$$

To compute it we must review the moment generating function.

$$g(t) = E(e^{tX}) = \int_{-\infty}^{+\infty} e^{tx} \tfrac{1}{\sqrt{2\pi\sigma^2}} exp\left\{ -\tfrac{(x-\mu)^2}{2\sigma^2} \right\} = e^{\mu t + \frac{\sigma^2 t^2}{2}}$$

It is trivial now to find $E(Y)$ and $V(Y)$.

$$t = 1 \quad g(1) = E(e^x) = e^{\mu + \frac{\sigma^2}{2}} = E(Y)$$

$$t = 2 \quad g(2) = E(e^{2x}) = e^{2(\mu + \sigma^2)} = E(Y^2)$$

$$V(Y) = E(Y^2) - E(Y)^2 = e^{2(\mu+\sigma^2)} - e^{2(\mu + \frac{\sigma^2}{2})} =$$

$$= e^{2\mu}e^{2\sigma^2} - e^{2\mu}e^{\sigma^2} = e^{2\mu}e^{\sigma^2}(e^{\sigma^2} - 1) =$$

$$= e^{2\mu+\sigma^2}(e^{\sigma^2} - 1)$$

To recap:

$$Y \sim logN(\mu, \sigma^2)$$

$$E(Y) = e^{\mu+\frac{\sigma^2}{2}}$$

$$V(Y) = e^{2\mu+\sigma^2}(e^{\sigma^2} - 1)$$

5 Exercise 5

We have the following pareto distribution

$$f_X(x) = \begin{cases} 0 & for \;\; x \leq 1 \\ \frac{2.5}{x^{3.5}} & for \;\; x > 1 \end{cases}$$

$$F_X(x) = \begin{cases} 0 & for \;\; x \leq 1 \\ 1 - \frac{1}{x^{2.5}} & for \;\; x > 1 \end{cases}$$

To generate a sequence of i.i.d. random deviates X_i with Pareto distribution we must now find the inverse function F_X^{-1} so we can apply the integral transform theorem.

$$U_i \sim U(0,1) \Rightarrow X_i = F^{-1}(U_i) \sim Pareto(2.5, 1)$$

If $x > 1$ we have:

$$F_X(X_i) = 1 - \frac{1}{X_i^{2.5}} = U_i$$

$$\frac{1}{X_i^{2.5}} = 1 - U_i$$

$$X_i^{2.5} = \frac{1}{1-U_i}$$

$$X_i = \frac{1}{(1-U_i)^{1/2.5}}$$

$$X_i = (1 - U_i)^{-0.4} \sim Pareto(2.5, 1)$$

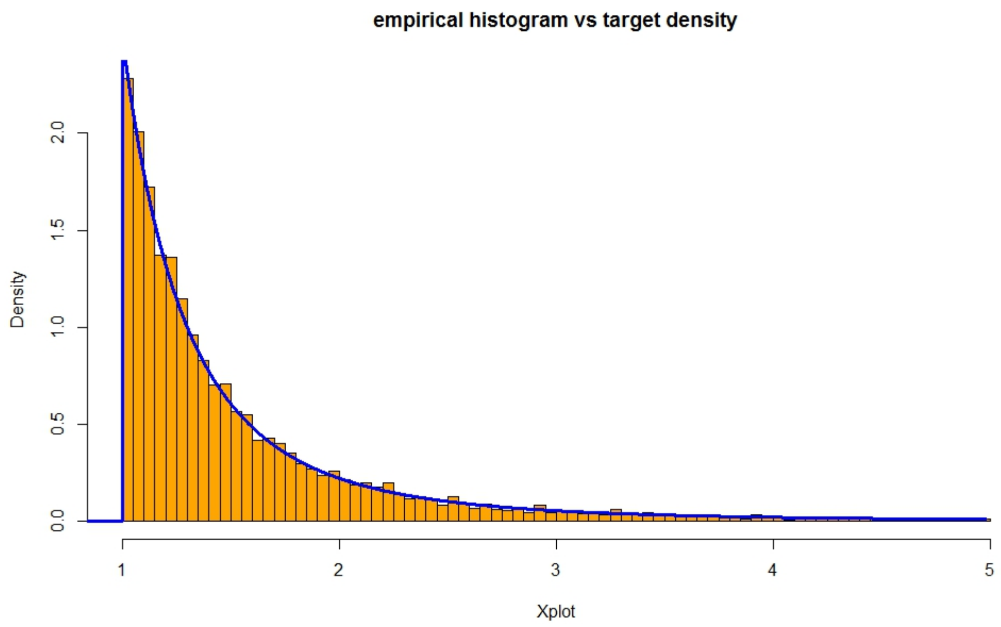
$$F_X^{-1}(u) = (1 - u)^{-0.4}$$

It follows the R code necessary to actually generate 1000 pseudo-random deviates from a Uniform distribution and transform them in i.i.d. random deviates X_i with a Pareto distribution of parameters $(\alpha, \beta) = (2.5, 1)$.

```
N=1000
U = runif(N)
X = (1-U)^(-0.4)
```

As we can see the generated sequence gives an histogram that fits the Pareto distribution.

```
dpar = function(x,a,b){
  if(x>b){
    return(a*(b^a/x^(a+1))) }
  else{
    return(0) } }

Xplot=X[X<5]
hist(Xplot,freq = F,breaks = 75, col = 'orange',
    main = 'empirical histogram vs target density')
xfit<-seq(0,max(Xplot),length=N)
yfit=rep(0,N)
for (i in 1:N){
  yfit[i] = dpar(xfit[i],2.5,1) }
lines(xfit, yfit, col="blue", lwd=3)
```



Each bin of interval $[a,b]$ of the histogram represents the frequency:

$$\frac{\#\, of\, x_i \in [a,b]}{n} = \frac{1}{n}\sum_{i=1}^{n} I_{[a,b]}(x)$$

Therefore each bin is a Monte Carlo approximation of the following probability.

$$Prob(x_i \in [a,b]) = \int_a^b f(x)dx = \int_{-\infty}^{+\infty} I_{[a,b]}(x) f(x) dx \simeq \frac{1}{n}\sum_{i=1}^{n} I_{[a,b]}(x)$$

In the end we can say that the entire histogram is some kind of Monte Carlo approximation of the density function.
To compute the first moment in closed form of the Pareto distribution we have to compute the expected value.

$$I = E(x) = \int_{-\infty}^{+\infty} x f(x) dx = \int_{\beta}^{+\infty} x \frac{\alpha\beta^{\alpha}}{x^{\alpha+1}} dx =$$

$$= \alpha\beta^{\alpha} \int_{\beta}^{+\infty} x^{-\alpha} = \alpha\beta^{\alpha} \left| \frac{x^{1-\alpha}}{1-\alpha} \right|_{\beta}^{+\infty} = \frac{\alpha\beta^{\alpha}}{1-\alpha} \left| \frac{1}{x^{\alpha-1}} \right|_{\beta}^{+\infty} =$$

$$= \begin{cases} \infty & for \;\; \alpha \leq 1 \\ -\frac{\alpha\beta^{\alpha}}{(1-\alpha)\beta^{\alpha-1}} = \frac{\alpha\beta}{\alpha-1} & for \;\; \alpha > 1 \end{cases}$$

$X \sim Pareto(2.5, 1) \quad E(X) = \frac{\alpha\beta}{\alpha-1} = \frac{2.5}{1.5} = 1.666 \;\; (\alpha > 1)$

Let's approximate I with the Monte Carlo approximation $\hat{I}$ as follows:

$h(X) = X \Rightarrow I = \mathbb{E}(h(X)) = \mathbb{E}(X) \Rightarrow \hat{I} = \frac{1}{n}\sum_{i=1}^{n} h(x_i) =$

$= \frac{1}{n}\sum_{i=1}^{n} x_i = 1.653203$

We can use the Central Limit Theorem to state that, if n is big enough, $\hat{I}$ is approximately normally distributed with expected value equal to I and variance equal to σ^2/n.

$V(X) = \sigma^2 \quad V(\hat{I}) = \frac{V(h(X))}{n} = \frac{V(X)}{n} = \frac{\sigma^2}{n}$

$\mathbb{E}(\hat{I}) = \frac{\sum \mathbb{E}(h(X_i))}{n} = \frac{n\,\mathbb{E}(X)}{n} = I \qquad \hat{I} \sim N(I, \frac{\sigma^2}{n})$

To approximate I up to a precision of 10^{-2}, we have to compute a confidence interval so that $I \in [\hat{I} - e, \hat{I} + e]$ with $e = 10^{-2}$. To allow this, since we don't have the variance, we can use the Monte Carlo estimate for σ^2 which is:

$$\hat{\sigma}^2 = \frac{\sum_{i=1}^{n}(X_i - \hat{I})^2}{n-1}$$

If, for n large enough, we assume that:

$$\frac{\hat{I} - I}{\hat{\sigma}/\sqrt{n}} \sim N(0, 1)$$

then, for a significance level of α, our confidence interval will be:

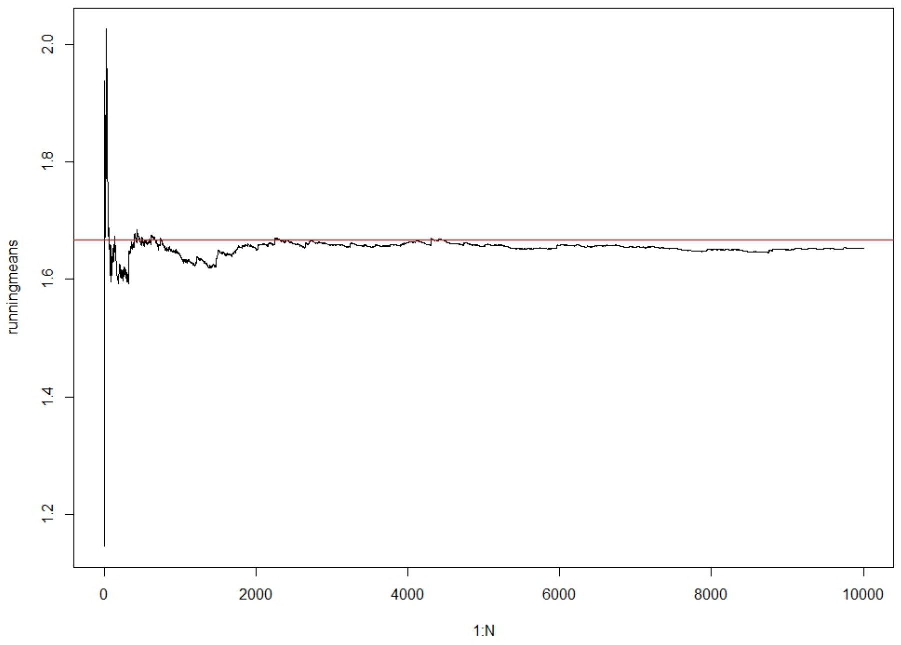
$$I \in [\ \hat{I} - z_{\frac{\alpha}{2}} \frac{\hat{\sigma}}{\sqrt{n}}, \hat{I} + z_{\frac{\alpha}{2}} \frac{\hat{\sigma}}{\sqrt{n}}\ ]$$

where our approximate error must be:

$$z_{\frac{\alpha}{2}} \frac{\hat{\sigma}}{\sqrt{n}} = 10^{-2}$$

Since everything is fixed except n, after choosing a significance level of α, we can increase n until our error doesn't decrease to 10^{-2}. Indeed to see how increasing the number of simulations improves the error, we can compute the running mean and see how it approaches to the expected value I.

```
approxI = mean(X)
I=2.5/1.5
runningmeans=cumsum(X)/(1:N)
plot(1:N,runningmeans,type="l")
abline(h=expval, col='red')
```



6 Exercise 6

Let's compute the inverse function such that:

$$F_x^{*-1}(u) = inf\,\{x : F_x(x) > u\}$$

$$F_X(x) = \begin{cases} 0 & for\ \ x < 0 \\ 0.25 & for\ \ 0 \leq x < 1 \\ 0.6 & for\ \ 1 \leq x < 2 \\ 1 & for\ \ x \geq 2 \end{cases}$$

$$F_x^{*-1} : (0,1) \Rightarrow \mathbb{R}$$

$$F_x^{*-1}(u) = \begin{cases} 0 & for\ \ 0 \leq u \leq 0.25 \\ 1 & for\ \ 0.25 < u \leq 0.6 \\ 2 & for\ \ 0.6 < u \leq 1 \end{cases}$$

We can simulate now 100 i.i.d. random deviates from the discrete distribution $F_X(x)$ using 100 pseudo-random deviates from a Uniform distribution and performing the generalized version of the Integral transform method.

```
genNsim = function(N){
  set.seed(123)
  U = runif(N)
  X = numeric(length = N)
  for(i in 1:N){
    if(U[i]>=0 && U[i]<=0.25){
      X[i]=0 }
    else if(U[i]> 0.25 && U[i]<=0.6){
      X[i]=1 }
    else if(U[i]> 0.6 && U[i]<=1){
      X[i]=2 } }
  return(X) }
```

To perform this directly without using pseudo-random deviates we can use the *sample*() command and compare the mean of the two sets.

```
> N = 100
> X = genNsim(N)
> Xdirect = sample(c(0,1,2),prob=c(0.25,0.35,0.4),size=N,replace=T)
> mean(X)
[1] 1.14
> mean(Xdirect)
[1] 1.18
```

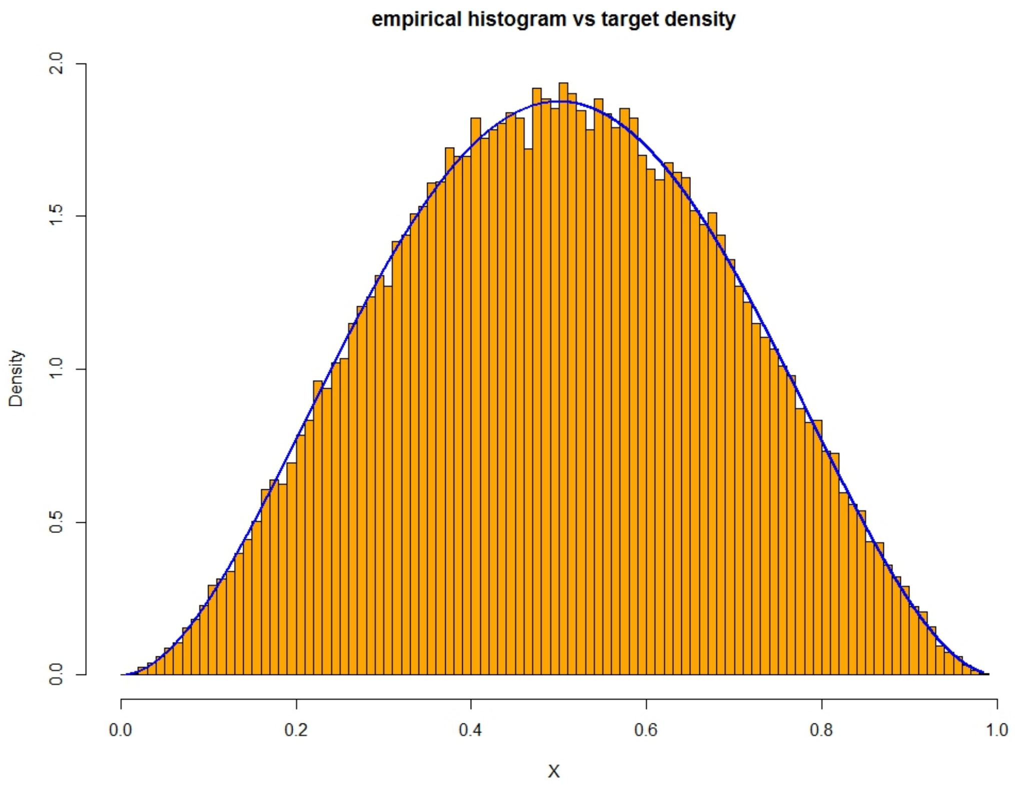
$\mathbb{E}(X) = \sum_k kProb(X = k) = 0 \cdot 0.25 + 1 \cdot 0.35 + 2 \cdot 0.4 = 1.15$

7 Exercise 7

To simulate n deviates from a $Beta(3, 3)$ using the A-R method we code in R as follows:

```
#target density f(x) : Beta(a1,b1)
#instrumental density g(y) : Beta(a2,b2)
#U is Uniform(0,1)
#acceptance condition: k*U < Beta(Y,a1,b1)/Beta(Y,a2,b2)
#parameter k: max(Beta(x,a1,b1)/Beta(x,a2,b2))

generateNBetafromBeta = function(N,a1,b1,a2,b2) {
  set.seed(123)
  #defining function to optimize
  funzBeta2 = function(x){
    dbeta(x,a1,b1)/dbeta(x,a2,b2) }
  #optimization starts from E(x)
  expf = c(a1/(a1+b1))
  k = optim(expf,funzBeta2,method="BFGS",control=list(fnscale=-1))$value

  X=NULL

  #We will create samples U and Y of length N*k,
  #until X reaches length N.
  #Usually this 'while' will iterate just once.
  #This because to create N xi with p of accept. = 1/k,
  #we need approximately N*k ui and yi.
  #If we don't select enough xi in the first iteration,
  #we should definitely get done in the second.

  while ( length(X) < N ){
    U =runif(N*k,min = 0,max = 1)
    Y= rbeta(N*k,a2,b2)
    X=c(X,Y[U<dbeta(Y,a1,b1)/(k*dbeta(Y,a2,b2))]) }
  #let's take away any extra xi
  X=X[1:N]
  return(X) }

#I chose a Beta(2,2) as instrumental or candidate distribution.
```

```
#It is better than a Beta(1,1)=Unif(0,1) because the resulting k is higher.
#This gives a smaller p of acceptance which brings more efficiency.

X = generateNBetafromBeta(100000,3,3,2,2)

#but we can use a Beta(1,1) = Unif(0,1) as well for our instrumental density.

X = generateNBetafromBeta(100000,3,3,1,1)

#as we can see it fits the target density
hist(X, freq = F, breaks = 75, col = 'orange',
     main = 'empirical histogram vs target density')
xfit<-seq(min(X),max(X),length=length(X))
yfit<-dbeta(xfit,3,3)
lines(xfit, yfit, col="blue", lwd=2)
```



We can compute now analytically the acceptance probability.

$$Y \Rightarrow \begin{cases} Y^A & if\ \ U \leq \frac{f_X(Y)}{k\, g_Y(Y)} \\ Y^R & if\ \ U \geq \frac{f_X(Y)}{k\, g_Y(Y)} \end{cases}$$

The target density:

$$X \sim Beta(3,3) \Rightarrow f_X(x) = \frac{x^2(1-x)^2}{\int_0^1 x^2(1-x)^2} =$$

$$= \frac{x^2(1-x)^2}{|\frac{x^5}{5} - \frac{x^4}{2} + \frac{x^3}{3}|_0^1} = 30x^2(1-x)^2$$

The candidate density:

$$Y \sim Beta(1,1) = U(0,1) \Rightarrow f_Y(y) = 1$$

$$X \in [0,1] = D_x \Rightarrow \int_0^1 f_X(x)\, dx = 1$$

$$Y \in [0,1] = D_y \Rightarrow \int_0^1 f_Y(y)\, dy = 1$$

$$D_x = D_y \qquad \exists\ k \geq \frac{f_X(x)}{g_Y(x)}\ \ \forall x$$

The acceptance probability is then:

$$Prob(Y = Y^A) =$$

$$= \int_{D_y} Prob(Y = Y^A | Y = y)\, g_Y(y)\, dy =$$

$$= \int_0^1 Prob(U \leq \frac{f_X(y)}{k\, g_Y(y)})\, g_Y(y)\, dy =$$

$$= \int_0^1 [\int_0^{\frac{f_X(y)}{k\, g_Y(y)}} du]\, g_Y(y)\, dy =$$

$$= \int_0^1 \frac{f_X(y)}{k\, g_Y(y)}\, g_Y(y)\, dy =$$

$$= \int_0^1 \frac{30\, y^2(1-y)^2}{k}\, dy = \frac{30}{30\, k} = \frac{1}{k}$$

$\frac{f_X(x)}{g_Y(x)} = 30\, x^2(1-x)^2$

$x^* = argmax(30\, x^2(1-x)^2) = 0.5 \Rightarrow k = 30 \cdot 0.5^2 \cdot (1-0.5)^2 = 1.875$

$Prob(Y = Y^A) = \frac{1}{1.875} = 0.533$

It can be shown that by choosing a candidate density Beta(2,2) instead of a uniform, the relative k will be equeal to 1.25 giving an acceptance probability of 0.8, which being higher gives a more efficient and faster system.
We can now think each attempt to simulate the target distribution as a Bernoulli random variable.

$\tilde{X}_i \sim Bern(\frac{1}{k})$

$$Y \Rightarrow \begin{cases} Y^A & if\ \ U \leq \frac{f_X(Y)}{k\, g_Y(Y)} \\ Y^R & if\ \ U \geq \frac{f_X(Y)}{k\, g_Y(Y)} \end{cases} \Rightarrow \tilde{X} \Rightarrow \begin{cases} 1 & if\ \ U \leq \frac{f_X(Y)}{k\, g_Y(Y)} \\ 0 & if\ \ U \geq \frac{f_X(Y)}{k\, g_Y(Y)} \end{cases}$$

$$Prob(\tilde{X}_i = \tilde{x}) = \begin{cases} \frac{1}{k} & if\ \ for\ \ \tilde{x} = 1 \\ 1 - \frac{1}{k} & if\ \ for\ \ \tilde{x} = 0 \end{cases}$$

$I = \mathbb{E}(\tilde{X}_i) = \frac{1}{k} = 0.5333$

$\hat{I} = \frac{1}{n} \sum_i^n \tilde{X}_i = 0.5376$

It follows the R code to generate $\tilde{X}$ and compute the approximation of Monte Carlo $\hat{I}$.

```
generateBernFromNTrials = function(N, a1, b1, a2, b2){
  set.seed(123)
  funzBeta2 = function(x){
    dbeta(x,a1,b1)/dbeta(x,a2,b2) }
  k = optim(c(a1/(a1+b1)),funzBeta2,
              method="BFGS",control=list(fnscale=-1))$value
  U =runif(N,min = 0,max = 1)
  Y= rbeta(N,a2,b2)
  X = Y[U<=dbeta(Y,a1,b1)/(k*dbeta(Y,a2,b2))]
  succ = rep(1,length(X))
  unsuc = rep(0,N - length(X))
  X = c(succ,unsuc)
  print('I=')
  print(1/k)
  print('I_hat=')
  print(mean(X))
  return(X) }

  > Xtilde = generateBernFromNTrials(10000,3,3,1,1)
[1] "I="
[1] 0.5333
[1] "I_hat="
[1] 0.5376

 #the beta(2,2) candidate density performs better
 #with an higher acceptance probability
 #making the process more efficient
 > Xtilde = generateBernFromNTrials(10000,3,3,2,2)
[1] "I="
[1] 0.8
[1] "I_hat="
[1] 0.8024
```

8 Exercise 8

General analytic expression of the acceptance probability:

$Prob(Y = Y^A) =$

$= \int_{D_y} Prob(Y = Y^A | Y = y)\, g_Y(y)\, dy =$

$= \int_{-\infty}^{+\infty} Prob(U \leq \frac{f_X(y)}{k\, g_Y(y)})\, g_Y(y)\, dy =$

$= \int_{-\infty}^{+\infty} [\int_0^{\frac{f_X(y)}{k\, g_Y(y)}} du]\, g_Y(y)\, dy =$

$= \int_{-\infty}^{+\infty} \frac{f_X(y)}{k\, g_Y(y)}\, g_Y(y)\, dy =$

$= \int_{-\infty}^{+\infty} \frac{f_X(y)}{k}\, dy = \frac{1}{k}$

$x^* = argmax(\frac{f_X(x)}{g_Y(x)}) \Rightarrow k = \frac{f_X(x^*)}{g_Y(x^*)}$

Bayesian inference wiht A-R method:

The candidate density ($g_Y(y)$) is prior density:

$\theta \sim \pi(\theta)$

The target density ($f_X(x)$) is posterior density:

$\theta/X \sim \pi(\theta/x)$

We need to simulate from a target distribution whose density is known up to an unknown finite positive constant c.

$\pi(\theta/x) = c \cdot \pi(\theta)\ L_x(\theta) \propto \pi(\theta)\ L_x(\theta)$

$\theta^*_{MLE} = argmax_\theta(L_x(\theta)) = argmax_\theta(\frac{\pi(\theta/x)}{\pi(\theta)})$

$f_X(x) \leq k\ g_Y(x) \Rightarrow \pi(\theta/x) \leq k\ \pi(\theta)\ \ \forall\ \theta \in \Theta$

$\frac{\pi(\theta/x)}{\pi(\theta)} \leq k = \frac{\pi(\theta^*_{MLE}/x)}{\pi(\theta^*_{MLE})} = c \cdot L_x(\theta^*_{MLE})\ \ \forall\ \theta \in \Theta$

The algorithm is:

Repeat:

generate θ_i from $\pi(\theta)$

generate U_i from $Unif(0,1)$

if $\quad U_i \leq \frac{\pi(\theta_i/x)}{k\ \pi(\theta_i)} = \frac{L_x(\theta_i)}{L_x(\theta^*_{MLE})} \quad :$

accept $\theta_{POST\ i} = \theta_i$

Indeed we can perform this on the conjugate model of Exercise 1.

$$\pi(\theta) \Rightarrow Beta(1,1)$$

$$\pi(\theta/x) \Rightarrow Beta(4,8)$$

$$c \cdot L_x(\theta^*_{MLE}) \Rightarrow max_\theta(\frac{\pi(\theta/x)}{\pi(\theta)}) = k$$

The acceptance probability will be then $\frac{1}{k}$. Some possible difficulties of this kind of approach might be related to such value. Indeed usually when you implement a general A-R method, you worry to find a candidate density g able to minimize the acceptance probability and make the algorithm more efficient. Unfortunately our candidate is the prior and we are not completely free to find what more suits us. This will mean that if $\frac{1}{k}$ is too big for any decent prior $\pi(\theta)$, the method will accept and create θ_{POST} at a really slow pace, resulting in a really inefficient system.

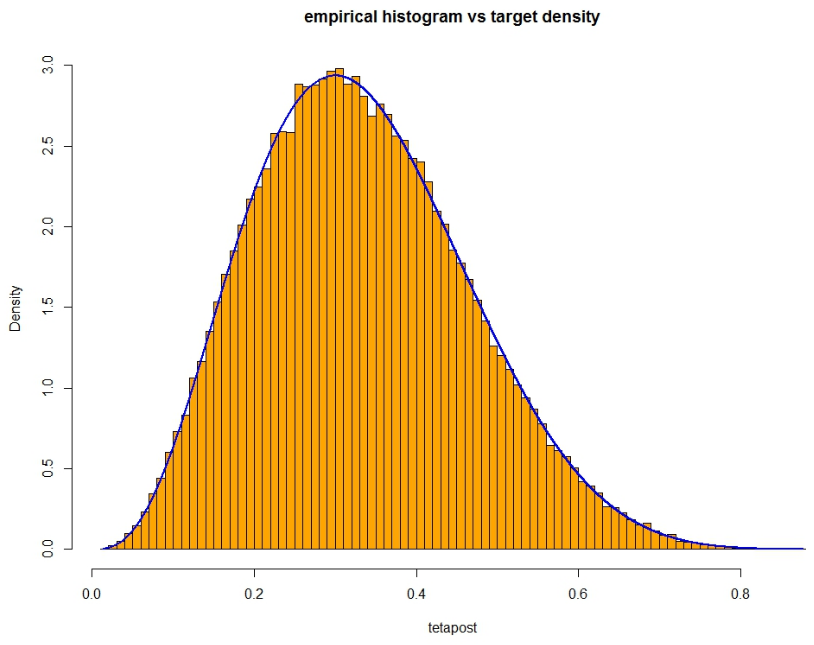
Let's implement our A-R approach with this conjugate model:

```
generateTetaPost = function(N,a_post,b_post) {
  set.seed(123)
  a_prior = 1
  b_prior = 1
  fk = function(x){
    dbeta(x,a_post,b_post)/dbeta(x,a_prior,b_prior) }
  k = optim(c(a_post/(a_post+b_post)),fk,
                      method="BFGS",control=list(fnscale=-1))$value

  tetaPost=NULL
  while (length(tetaPost)<N){

    U =runif(N*k)
    teta= rbeta(N*k,a_prior,b_prior)
    tetaPost=c(tetaPost,teta[U<dbeta(teta,a_post,b_post)/
                              (k*dbeta(teta,a_prior,b_prior))])}

  tetaPost=tetaPost[1:N]
  return(tetaPost) }

a_post = 4
b_post = 8

tetapost = generateTetaPost(100000,4,8)

>mean(tetapost)
[1] 0.3327541
```

```
> a_post/(a_post+b_post)
[1] 0.3333333
```

$$\mathbb{E}(\theta/x) = \frac{\alpha_{POST}}{\alpha_{POST}+\beta_{POST}}$$

```
hist(tetapost,freq = F,breaks = 75, col = 'orange',
    main = 'empirical histogram vs target density')
xfit<-seq(min(tetapost),max(tetapost),length=length(tetapost))
yfit<-dbeta(xfit,4,8)
lines(xfit , yfit , col="blue", lwd=2)
```

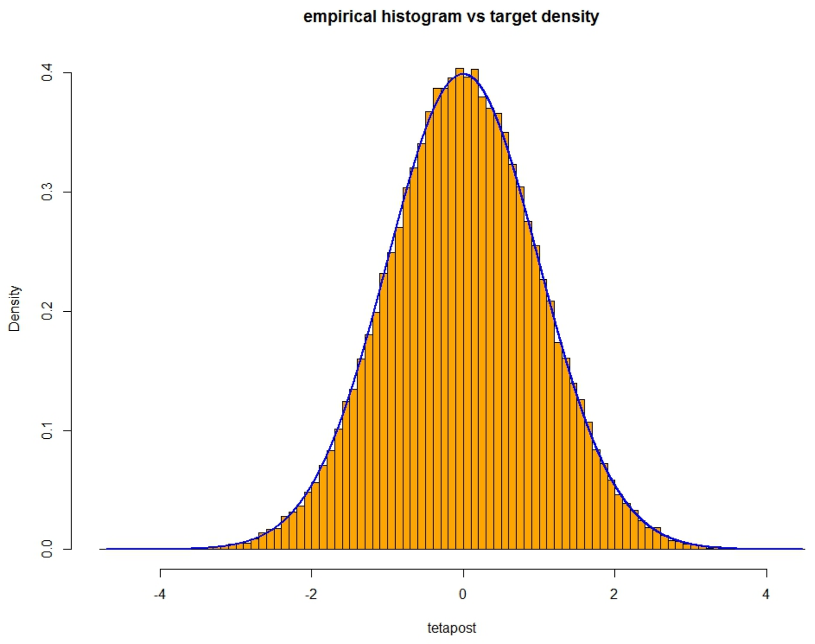


9 Exercise 9

To simulate n deviates from a $N(0,1)$ using the A-R method we code in R as follows:

```
generateTetaPost = function(N) {
  set.seed(123)
  fk = function(x){
    dnorm(x,mean=0,sd=1)/dcauchy(x,location = 0, scale = 1)
  }
  k = optim(c(1),fk,method="BFGS",control=list(fnscale=-1))$value
```

```
    tetaPost=NULL
    while (length(tetaPost)<N){
      U =runif(N*k)
      teta= rcauchy(N*k, location = 0, scale = 1)
      tetaPost=c(tetaPost, teta[U<=dnorm(teta,mean=0,sd=1)
                                    /(k*dcauchy(teta, location = 0, scale = 1))])}
    tetaPost=tetaPost[1:N]
    return(tetaPost)
}
tetapost = generateTetaPost(100000)

hist(tetapost, freq = F, breaks = 75, col = 'orange',
     main = 'empirical␣histogram␣vs␣target␣density')
xfit<-seq(min(tetapost),max(tetapost),length=length(tetapost))
yfit<-dnorm(xfit,0,1)
lines(xfit, yfit, col="blue", lwd=2)
```



We can now think each attempt to simulate the target distribution as a Bernoulli random variable.

$$\tilde{X}_i \sim Bern(\tfrac{1}{k})$$

$$Y \Rightarrow \begin{cases} Y^A & if\ \ U \leq \frac{f_X(Y)}{k\,g_Y(Y)} \\ Y^R & if\ \ U \geq \frac{f_X(Y)}{k\,g_Y(Y)} \end{cases} \Rightarrow \tilde{X} \Rightarrow \begin{cases} 1 & if\ \ U \leq \frac{f_X(Y)}{k\,g_Y(Y)} \\ 0 & if\ \ U \geq \frac{f_X(Y)}{k\,g_Y(Y)} \end{cases}$$

$$Prob(\tilde{X}_i = \tilde{x}) = \begin{cases} \frac{1}{k} & if \;\; for \;\; \tilde{x} = 1 \\ 1 - \frac{1}{k} & if \;\; for \;\; \tilde{x} = 0 \end{cases}$$

$$I = \mathbb{E}(\tilde{X}_i) = \frac{1}{k} = 0.6577446$$

$$\hat{I} = \frac{1}{n}\sum_i^n \tilde{X}_i = 0.65935$$

It follows the R code to generate $\tilde{X}$ and compute the approximation of Monte Carlo $\hat{I}$.

```
 generateBernFromNTrials = function(N){
  set.seed(123)
  fk = function(x){
    dnorm(x,mean=0,sd=1)/dcauchy(x, location = 0, scale = 1) }
  k = optim(c(1),fk,method="BFGS",control=list(fnscale=-1))$value
  U =runif(N,min = 0,max = 1)
  teta= rcauchy(N,location = 0, scale = 1)
  tetapost = teta[U<=dnorm(teta,mean=0,sd=1)/
                   (k*dcauchy(teta,location = 0, scale = 1))]
  succ = rep(1,length(tetapost))
  unsuc = rep(0,N - length(tetapost))
  X = c(succ,unsuc)
  print('approx.␣accept.␣prob.')
  print(mean(X))
  print('theoretical␣accept.␣prob.')
  print(1/k)
  print('#␣of␣successes')
  print(sum(X))
  print('#␣of␣trials')
  print(N)
  return(X) }

Xtilde = generateBernFromNTrials(100000)
[1] "approx.␣accept.␣prob."
[1] 0.65935
[1] "theoretical␣accept.␣prob."
[1] 0.6577446
[1] "#␣of␣successes"
[1] 65935
[1] "#␣of␣trials"
[1] 1e+05
```