TMA4300 - Exercise 1

Elena Dami, Marco Pozzetto & Marijan Soric

PROBLEM A: Stochastic simulation by the probability integral transform and bivariate techniques

1.

The following function, that we called `sample_exponential`, generates samples from an exponential distribution with rate λ. This function takes as input the rate parameter λ and the number of samples to generate, n, and it returns a vector with the generated random numbers. We used the inversion method, with the following formula for the inverse

$$F^{-1}(u) = x = -\frac{\log(1-u)}{\lambda}.$$

```r
sample_exponential = function(lambda, n) {
  u  = runif(n)
  x = -log(u)/lambda
  return(x)
}
```

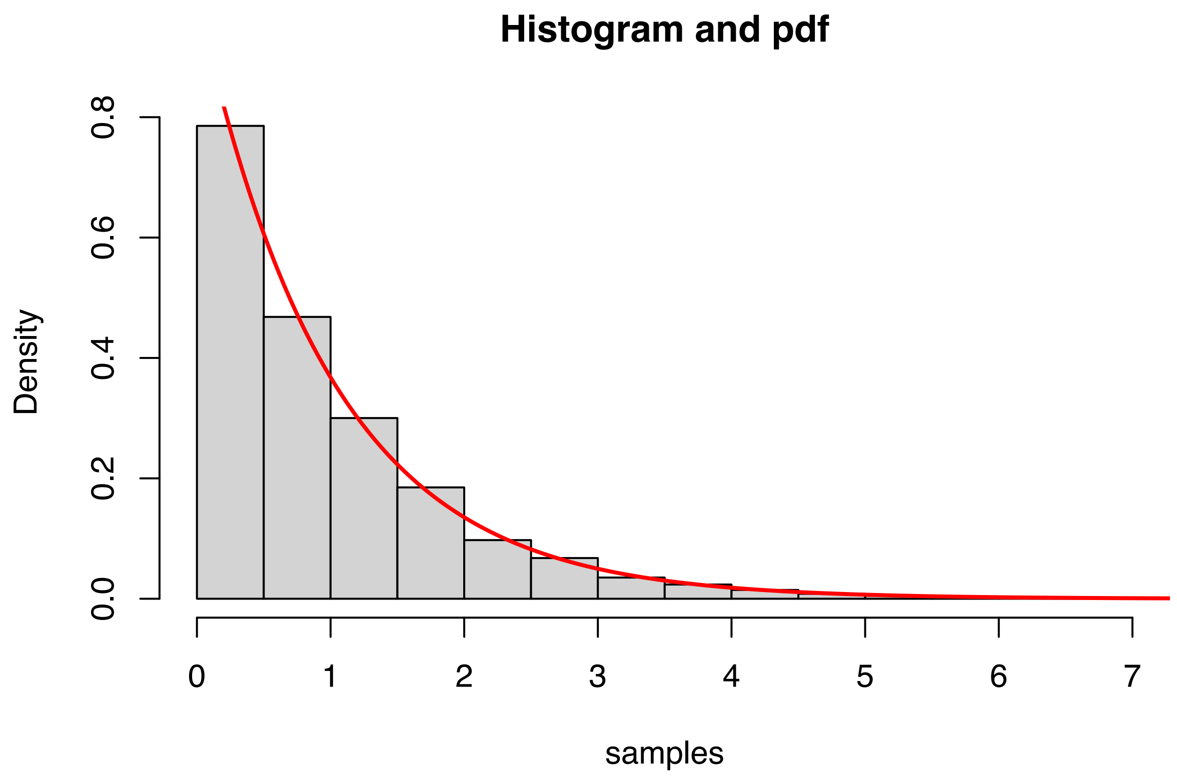
We used this function to generate 10000 samples, setting $\lambda = 1$. To check that the function works properly, we made a histogram to visualize the distribution, and we computed the empirical mean and the empirical variance, to compare them with the theoretical mean, $\frac{1}{\lambda}$, and theoretical variance, $\frac{1}{\lambda^2}$.

In the same plot, we also put the probability density function of the exponential. For this purpose we wrote the function `density_exp`.

```r
density_exp <- function(lambda, x) {
  if(x>0) {return (lambda*exp(-lambda*x ))}
  else {return (0)}
}
```

```r
lambda <-1
y = c()
x <- seq(1e-10,8,0.01)
for(i in x) {
  y= c(y,density_exp(lambda,i))
}
```

```r
samples = sample_exponential(lambda, 10000)
hist(samples, freq=FALSE, main="Histogram and pdf", xlim=c(0,7))
lines(x,y, col="red", lwd=2)
```



```
true_mean <- 1/lambda
estimated_mean <- mean(samples)
true_variance <- 1/lambda^2
estimated_variance <- var(samples)
```

True Mean: 1 Estimated Mean: 1.001092 True Variance: 1 Estimated Variance: 0.9945507

As we can see, the estimated mean is similar to the true mean, and the estimated variance is similar to the true variance, meaning that the function works properly.

2.

In this exercise we considered the following probability density function

$$g(x) = \begin{cases} cx^{\alpha-1}, & \text{for } 0 < x < 1, \\ ce^{-x}, & \text{for } 1 \leq x, \\ 0, & \text{otherwise.} \end{cases}$$

where c is a normalising constant and $\alpha \in (0,1)$.

We computed its cumulative distribution function $G(x)$ as:

$$G(x) = \int_{-\infty}^{x} g(s)ds = \begin{cases} \int_0^x cs^{\alpha-1}\,ds, & 0 < x < 1, \\ \int_0^1 cs^{\alpha-1}\,ds + \int_1^x ce^{-s}\,ds, & x \geq 1, \\ 0, & \text{otherwise.} \end{cases} = \begin{cases} \frac{cx^\alpha}{\alpha}, & 0 < x < 1, \\ \frac{c}{\alpha} + c(e^{-1} - e^{-x}), & x \geq 1, \\ 0, & \text{otherwise.} \end{cases}$$

To find the value of c, we computed $\int_{-\infty}^{\infty} g(x)dx$ and set it equal to 1:

$$\int_{-\infty}^{\infty} g(x)dx = \int_0^1 cx^{\alpha-1}dx + \int_1^{\infty} ce^{-x}dx = c\left(\frac{1}{\alpha}\right) - c(-e^{-1}) = c\left(\frac{1}{\alpha}+\frac{1}{e}\right) = 1.$$

Thus, we obtained

$$\Rightarrow c = \frac{1}{\frac{1}{\alpha}+\frac{1}{e}} = \frac{\alpha e}{\alpha+e}.$$

Substituting this value into the expression for $G(x)$, we get

$$G(x) = \begin{cases} \frac{e}{\alpha+e}x^{\alpha}, & 0<x<1,\\ 1-\frac{\alpha}{\alpha+e}e^{1-x}, & x\geq 1,\\ 0, & \text{otherwise.}\end{cases}$$

Then we computed the inverse $G^{-1}(u)$

$$G^{-1}(u) = \begin{cases} (u\frac{\alpha+e}{e})^{\frac{1}{\alpha}}, & \text{for } 0<u<\frac{e}{\alpha+e},\\ -\ln\left[\frac{\alpha+e}{\alpha e}(1-u)\right], & \text{for } u\geq\frac{e}{\alpha+e},\\ 0, & \text{otherwise.}\end{cases}$$

After that, we wrote a function that generates samples from $g(x)$ using the inversion method, called `sample_g`:

```r
sample_g <- function(n, alpha) {
u <- runif(n)
x <- ifelse(u >= 0 & u < exp(1)/(alpha + exp(1)), (u*(alpha + exp(1))/ exp(1))^(1/alpha),
            ifelse(u >= exp(1)/(alpha + exp(1)),
                   -log((alpha + exp(1))/(alpha * exp(1))*(1 - u)), 0))
  return(x)
}
```

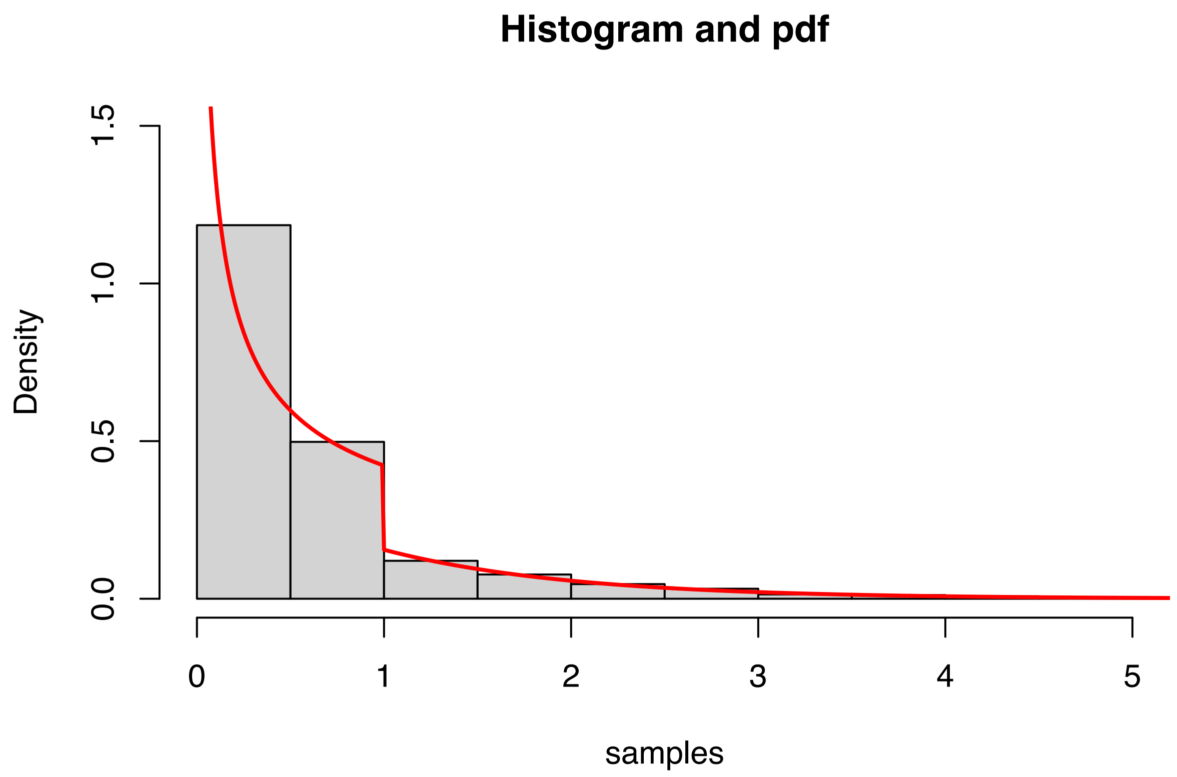
To check that the function is correct, we made a histogram, and we compared the empirical mean with the true mean.

The true mean is given by

$$\mathbb{E}[X] = \int_0^1 x\cdot cx^{\alpha-1}\,dx + \int_1^{\infty} x\cdot ce^{-x}\,dx = c\int_0^1 x^{\alpha}\,dx + c\int_1^{\infty} xe^{-x}\,dx =$$

$$c\left[\frac{x^{\alpha+1}}{\alpha+1}\right]_0^1 + ce^{-1} + ce^{-1} = c\left(\frac{1}{\alpha+1}\right) + 2ce^{-1} = \frac{\alpha e}{\alpha+e}\left(\frac{1}{\alpha+1}\right) + \frac{2\alpha}{\alpha+e}.$$

```r
density_g <- function(alpha,x) {
  c <- alpha*exp(1)/(alpha+exp(1))
  ifelse(x>0 & x<1, c*x^(alpha-1), ifelse(x>=1, c*exp(-x),0))
}

alpha <- 0.5
x <- seq(1e-10, 8, 0.01)
y <- density_g(alpha,x)

samples=sample_g(10000,alpha)
hist(samples, freq=FALSE, main="Histogram and pdf", xlim=c(0,5), ylim=c(0,1.5))
lines(x,y, col="red", lwd=2)
```



```
estimated_mean <- mean(samples)
true_mean <- (alpha*exp(1))/((alpha+exp(1))*(alpha+1)) + 2*alpha/(alpha+exp(1))
```

True Mean: 0.5922707 Estimated Mean: 0.6001572 We note that these values are similar, so the function works correctly.

3.

We considered the probability density function

$$f(x) = \frac{ce^{\alpha x}}{(1+e^{\alpha x})^2}, \quad -\infty < x < \infty,\ \alpha > 0,$$

where c is a normalising constant.

First we found the value of c using the following property of a density function

$$\int_{-\infty}^{\infty} f(x)dx = 1.$$

$$\int_{-\infty}^{\infty} f(x)dx = \int_{-\infty}^{\infty} \frac{ce^{\alpha x}}{(1+e^{\alpha x})^2}dx = \frac{1}{\alpha}c\int_{-\infty}^{\infty} \alpha e^{\alpha x}(1+e^{\alpha x})^{-2}dx = \frac{c}{\alpha}\left[-(1+e^{\alpha x})^{-1}\right]_{-\infty}^{\infty} = \frac{c}{\alpha} = 1$$
$$\Rightarrow c = \alpha.$$

Then, we computed formulas for the cumulative distribution function $F(x)$ and its inverse $F^{-1}(u)$.

$$F(x) = \int_{-\infty}^{x} f(t)dt = \int_{-\infty}^{x} \frac{\alpha e^{\alpha t}}{(1+e^{\alpha t})^2}dt = \int_{-\infty}^{x} \alpha e^{\alpha t}(1+e^{\alpha t})^{-2}dt = [-(1+e^{\alpha t})^{-1}]_{-\infty}^{x} = -\frac{1}{1+e^{\alpha x}} + 1.$$

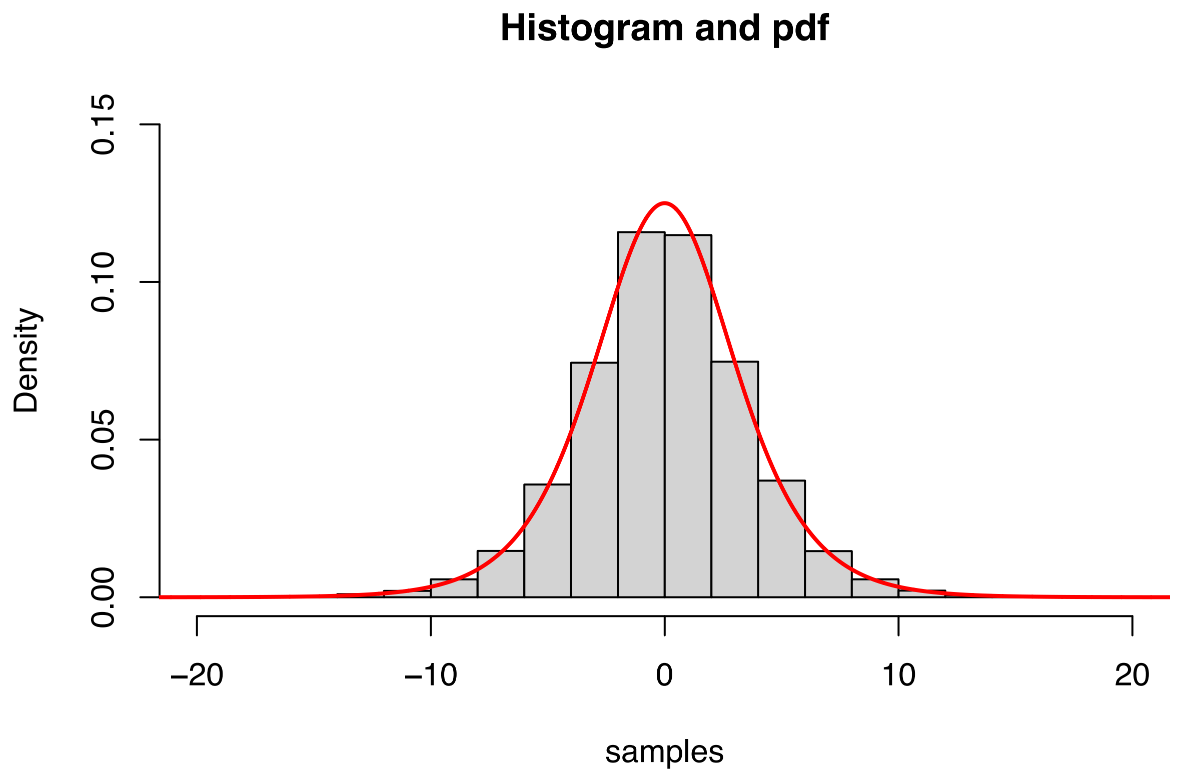
$$x = F^{-1}(u) = \frac{1}{\alpha}\ln\left(\frac{u}{1-u}\right).$$

We wrote a new function called `sample_f` that generates samples from $f(x)$. This function takes two input arguments, α and n, and it returns a vector with the generated random numbers.

```
sample_f = function(alpha, n) {
  u  = runif(n)
  x = log(u/(1-u))/alpha
  return(x)
}
```

We made a histogram to check that the function is working properly.

```
f = function(x, alpha) {
  return(alpha*exp(alpha*x)/(1+exp(alpha*x))^2)
}

samples = sample_f(.5, 100000)
hist(samples, freq = FALSE, main="Histogram and pdf", ylim=c(0,0.15),
     xlim=c(-20,20))
x=seq(-30,30,0.01)
y=f(x,.5)
lines(x,y,col='red', lwd=2)
```



We checked the empirical variance against the true variance, which give similar results.

```
var(samples)
```

```
## [1] 13.23418
```

```
pi^2/3 *(1/alpha)^2 # true variance
```

```
## [1] 13.15947
```

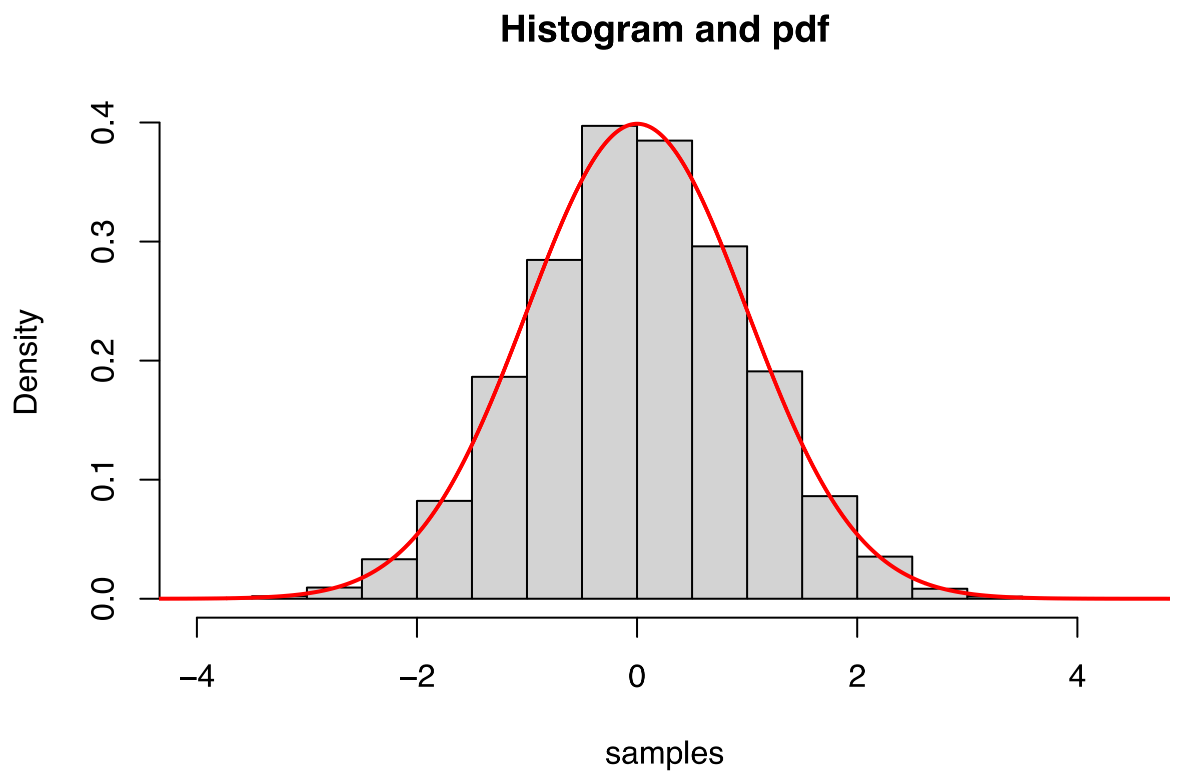
4.

We wrote a new function, `sample_normal_Box_Muller` that uses the Box-Muller algorithm to generate a vector of n independent samples from the standard normal distribution.

```
sample_normal_Box_Muller=function(n) {
  theta = runif(n, 0, 2*pi)
  u = runif(n)
  r = sqrt(-2*log(u))
  y1 = r*cos(theta)
  y2 = r*sin(theta)
  return(cbind(y1, y2))
}
```

We checked that the function is working properly by making an histogram and computing the empirical mean and variance, that should be respectively equal to 0 and 1.

```
density_normst <- function(x) {
  return(exp((-x^2)/2)/sqrt(2*pi))
}

x <- seq(-5,5,0.01)
y <- density_normst(x)
samples <- sample_normal_Box_Muller(10000)[,1]
hist(samples, freq=FALSE, main= "Histogram and pdf")
lines(x,y, col="red", lwd=2)
```



```
mean(samples)
```

```
## [1] 0.01204746
```

```
var(samples)
```

```
## [1] 0.9928171
```

5.

We wrote a new function, `sample_d_variate_normal_distribution`, that generates realisations from a d-variate normal distribution with given mean vector μ and covariance matrix Σ.

```
generate_d_variate_normal_distribution <- function(n, mu, sigma) {
  d <- length(mu)
  realisations <- matrix(NA, nrow = n, ncol = d)
  # Initialize dxd matrix to store realisations
  transformed_values <- numeric(d)

  for (i in 1:n) {
    # Generate one standard normal random variable using Box-Muller algorithm
    z <- sample_normal_Box_Muller(d)[,1]

    # Transform standard normal random variable
    transformed_values <- mu + t(z) %*% chol(sigma)

    # Store realizations
    realisations[i, ] <- transformed_values
  }

  return(realisations)
}
```

We checked that the function is working properly, by comparing μ and Σ with the corresponding estimated values.

```
mu <- c(1:3)
sigma <- matrix(c(1, 0.5, 0.3,
                  0.5, 1, 0.2,
                  0.3, 0.2, 1), nrow = length(mu), ncol = length(mu))

sample_matrix <- generate_d_variate_normal_distribution(10000, mu, sigma)

true_mean <- mu
estimated_mean <- colMeans(sample_matrix)
true_covariance <- sigma
estimated_covariance <- matrix(cov(sample_matrix),length(mu), length(mu))

true_mean
```

```
## [1] 1 2 3
```

```
estimated_mean
```

```
## [1] 0.9824544 1.9969253 2.9841342
```

```
true_covariance
```

```
##      [,1] [,2] [,3]
## [1,]  1.0  0.5  0.3
## [2,]  0.5  1.0  0.2
## [3,]  0.3  0.2  1.0
```

```
estimated_covariance
```

```
##           [,1]      [,2]      [,3]
## [1,] 0.9843104 0.4978563 0.2826544
## [2,] 0.4978563 1.0153852 0.2046461
## [3,] 0.2826544 0.2046461 0.9824264
```

PROBLEM B: The gamma distribution

1.

In this exercise, we consider a gamma distribution with parameter $\alpha \in (0,1)$ and $\beta = 1$,

$$f(x) = \begin{cases} \frac{1}{\Gamma(\alpha)} x^{\alpha-1} e^{-x}, & x > 0, \\ 0, & \text{otherwise.} \end{cases}$$

We used rejection sampling to generate samples from this distribution by proposing samples from the distribution

$$g(x) = \begin{cases} \frac{\alpha e}{\alpha+e} x^{\alpha-1}, & \text{for } 0 < x < 1, \\ \frac{\alpha e}{\alpha+e} e^{-x}, & \text{for } 1 \leq x, \\ 0, & \text{otherwise.} \end{cases}$$

To find an expression for the acceptance probability, we first computed $\alpha' = \frac{f(x)}{cg(x)}$.

$$\alpha' = \frac{f(x)}{cg(x)} = \begin{cases} \frac{(\alpha+e)x^{\alpha-1}e^{-x}}{c\ \alpha\ e\ x^{\alpha-1}\ \Gamma(\alpha)} & 0 < x < 1, \\ \frac{x^{\alpha-1}(\alpha+e)e^{-x}}{c\ \alpha\ e\ e^{-x}\ \Gamma(\alpha)} & x \geq 1, \\ 0 & \text{otherwise.} \end{cases} = \begin{cases} \frac{(\alpha+e)e^{-x}}{c\ \alpha\ e\ \Gamma(\alpha)} & 0 < x < 1, \\ \frac{x^{\alpha-1}(\alpha+e)}{c\ \alpha\ e\ \Gamma(\alpha)} & x \geq 1, \\ 0 & \text{otherwise.} \end{cases}$$

We want $\alpha' \leq 1, \forall x \in \mathbb{R}$, which means:

$$0 < x < 1: \quad \alpha' \leq 1 \Leftrightarrow \frac{\alpha+e}{c\ \alpha\ e\ \Gamma(\alpha)} \leq 1 \Rightarrow c \geq \frac{\alpha+e}{\alpha\ e\ \Gamma(\alpha)},$$

$$x \geq 1: \quad \alpha' \leq 1 \Leftrightarrow \frac{\alpha+e}{c\ \alpha\ e\ \Gamma(\alpha)} \leq 1 \Rightarrow c \geq \frac{\alpha+e}{\alpha\ e\ \Gamma(\alpha)}.$$

We choose:

$$c = \frac{\alpha+e}{\alpha\ e\ \Gamma(\alpha)},$$

$$\Rightarrow \alpha' = \begin{cases} e^{-x}, & 0 < x < 1, \\ x^{\alpha-1}, & x \geq 1. \end{cases}$$

Then we wrote the function `rejection_sampling` that generates a vector of n independent samples from f. Before doing that, we wrote two functions, `f` and `g`, for the two densities $f(x)$ and $g(x)$.

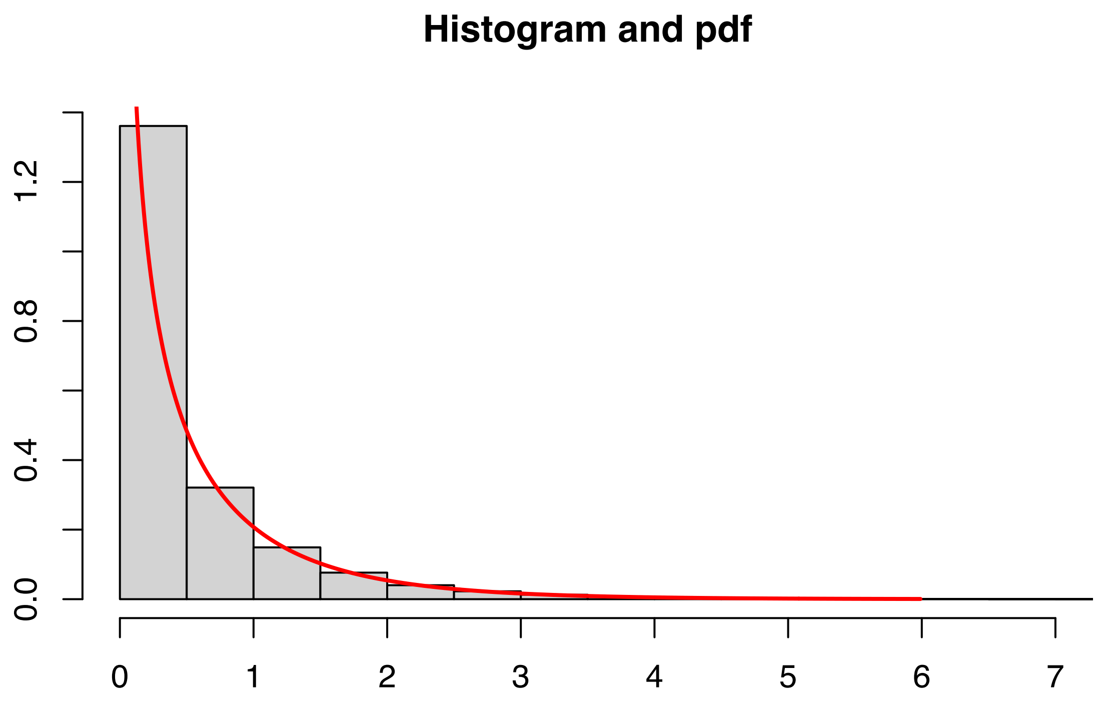
```r
f = function(x,alpha){
  ifelse(x>0, (x**(alpha-1)*exp(-x))/gamma(alpha),0)
}
```

```r
g = function(x,alpha){
  c = alpha*exp(1)/(alpha+exp(1))
  if (x<=0){return(0)}
  if ((0<x)&(x<1)){
    return(c*x**(alpha-1))
  }
  if (1<=x){return(c*exp(-x))}
}
```

```r
rejection_sampling <- function(alpha_fg, n) {

sampled_values <- numeric(n)
count <- 0
c <- (alpha_fg + exp(1)) / (alpha_fg * exp(1) * gamma(alpha_fg))

  while (count < n) {
    x <- sample_g(1, alpha_fg)
    u <- runif(1)
    alpha <- f(x, alpha_fg) / (c * g(x, alpha_fg))

    if (alpha > u) {
      count <- count + 1
      sampled_values[count] <- x
    }
  }

  return(sampled_values)
}
```

We checked that the function is working properly by making a histogram of the sampled values, and plotting the probability density function of $f(x)$ in the same graph.

```r
alpha_fg <- 0.5
x <- seq(1e-10,6,0.01)
y <- c()
for(i in x){
  y = c(y, f(i,alpha_fg))
}

sample <- rejection_sampling(alpha_fg,100000)
hist(sample, freq=FALSE, xlim=c(0,7), main="Histogram and pdf",
     ylab="", xlab="")
lines(x,y, col="red", lwd=2)
```



2.

We considered a gamma distribution with parameters $\alpha > 1$ and $\beta = 1$. We used the ratio of uniforms method to simulate from this distribution. We defined

$$f^*(x) = \begin{cases} x^{\alpha-1}e^{-x}, & x > 0, \\ 0, & \text{otherwise,} \end{cases}$$

$$C_f = \left\{(x_1, x_2) : 0 \leq x_1 \leq \sqrt{f^*(\frac{x_2}{x_1})}\right\} = \left\{(x_1, x_2) : 0 \leq x_1 \leq \sqrt{\left(\frac{x_2}{x_1}\right)^{\alpha-1} e^{-\frac{x_2}{x_1}}}\right\}$$

, and $a = \sqrt{\sup_x f^*(x)}$, $b_+ = \sqrt{\sup_{x\geq 0}(x^2 f^*(x))}$ and $b_- = -\sqrt{\sup_{x\leq 0}(x^2 f^*(x))}$,

so that $C_f \subset [0, a]\text{X}[b_-, b_+]$.

First we found the values of a, b_- and b_+ :

$$\frac{d \ln f^*(x)}{dx} = \frac{d}{dx}[(\alpha - 1)\ln x - x] = \frac{\alpha - 1}{x} - 1 = 0 \;\Rightarrow\; x = \alpha - 1,$$

$$a = \sqrt{\sup_x f^*(x)} = \sqrt{(\alpha-1)^{\alpha-1}e^{-(\alpha-1)}} = \left(\frac{\alpha-1}{e}\right)^{\frac{\alpha-1}{2}}.$$

$$\frac{d}{dx}\ln(x^2x^{\alpha-1}e^{-x}) = \frac{d}{dx}[(\alpha+1)\ln x - x] = \frac{\alpha+1}{x} - 1 = 0 \ \Rightarrow x = \alpha+1.$$

$$b_+ = \sqrt{\sup_{x\geq 0}(x^2 f^*(x))} = \sqrt{(\alpha+1)^{\alpha+1}e^{-\alpha-1}} = \left(\frac{\alpha+1}{e}\right)^{\frac{\alpha+1}{2}},$$

$$b_- = \sqrt{\inf_{x\leq 0}(x^2 f^*(x))} = 0.$$

Then we wrote the function `sample_f1` that generates a vector of n independent samples from f. We implemented the algorithm on log-scale, so we used the following formulas:

$$\log a = \frac{\alpha-1}{2}\log\left(\frac{\alpha-1}{e}\right), \log b_+ = \frac{\alpha+1}{e}\log\left(\frac{\alpha+1}{e}\right).$$

Since $X_1 \sim U(0,a)$ and $X_2 \sim U(0,b_*)$, we defined the two random variables, $X_1 = ar_1$ and $X_2 = ar_2$, where $r_1 \sim U(0,1)$ and $r_2 \sim U(0,\frac{b_+}{a})$. We can then write C_f as

$$C_f = \left\{(x_1,x_2) : 0 \leq \log(x_1) \leq \left(\frac{\alpha-1}{2}\right)\log\left(\frac{x_2}{x_1}\right) - \frac{x_2}{2x_1}\right\} = \left\{(r_1,r_2) : \log(ar_1) \leq \frac{1}{2}\left((\alpha-1)\log\left(\frac{r_2}{r_1}\right) - \frac{r_2}{r_1}\right)\right\}.$$

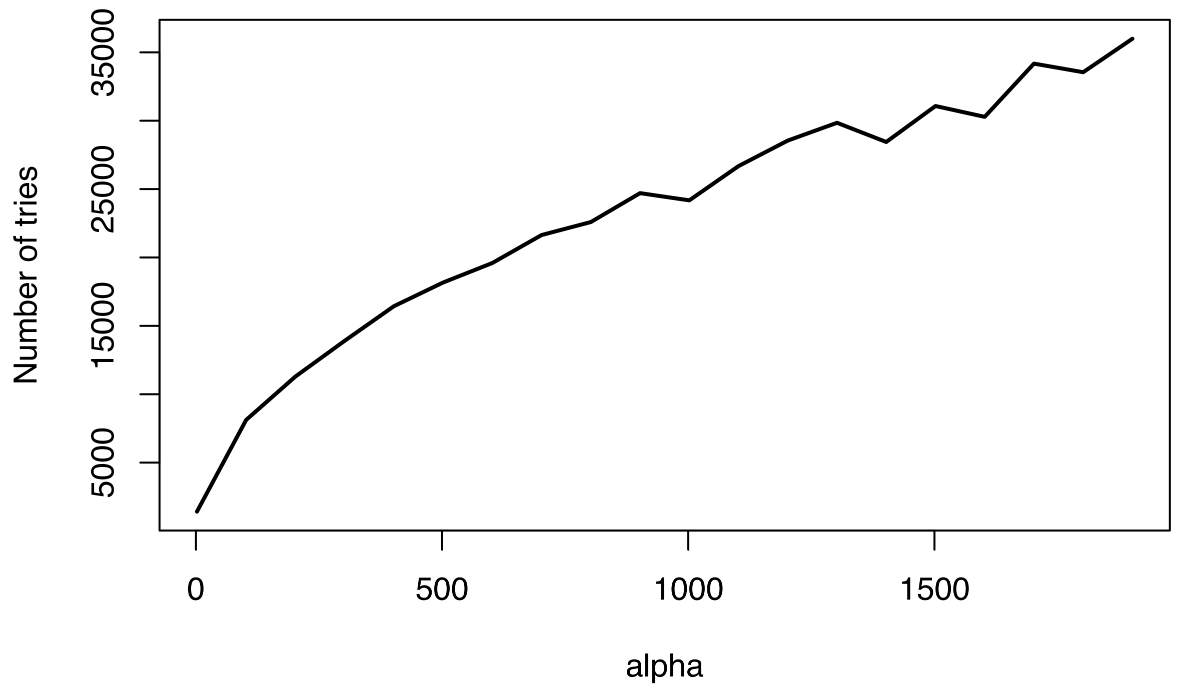
In the code, the value `b_first` refers to the ratio $\frac{b_+}{a} = \frac{e^{\log b_+}}{e^{\log a}} = e^{\log b_+ - loga}$.

```
sample_f1 = function(alpha, n){
  lna=(alpha-1)/2*log((alpha-1)/exp(1)) #a = ((alpha-1)/exp(1))((alpha-1)/2)
  b_min = 0
  lnb_max=(alpha+1)/2*log((alpha+1)/exp(1)) #b_max = ((alpha+1)/exp(1))((alpha+1)/2)
  b_first=exp(lnb_max-lna)
  i = 0
  x = c()
  repeat {
    r1 = runif(1, 0, 1)
    r2 = runif(1, 0, b_first)
    i = i + 1
    if((lna+log(r1))<= (0.5*(alpha-1)*log(r2/r1)-0.5*r2/r1)) {
      x=c(x, r2/r1)
    }
    if (length(x)==n){
      break}
  }
  return(list(x=x, i=i))
}
```

The function returns a vector of n independent samples, and the number of tries that the algorithm needs to generate n realisations, depending on the value of $\alpha \in (1, 2000]$.

We generated a plot with values of α on the x-axis and the number of tries used on the y-axis.

```
n=1000
try = c()
alpha_values <- seq(2,2000,100)
for (alpha in alpha_values){
  try = c(try, sample_f1(alpha, n)$i)
}
plot(alpha_values,try, ylab="Number of tries", xlab="alpha", type="l", lwd=2)
```



Looking at the plot, we can see that the number of iterations required for generating n samples, for a fixed n, grows when α grows. In particular, it has a logarithmic trend.

3.

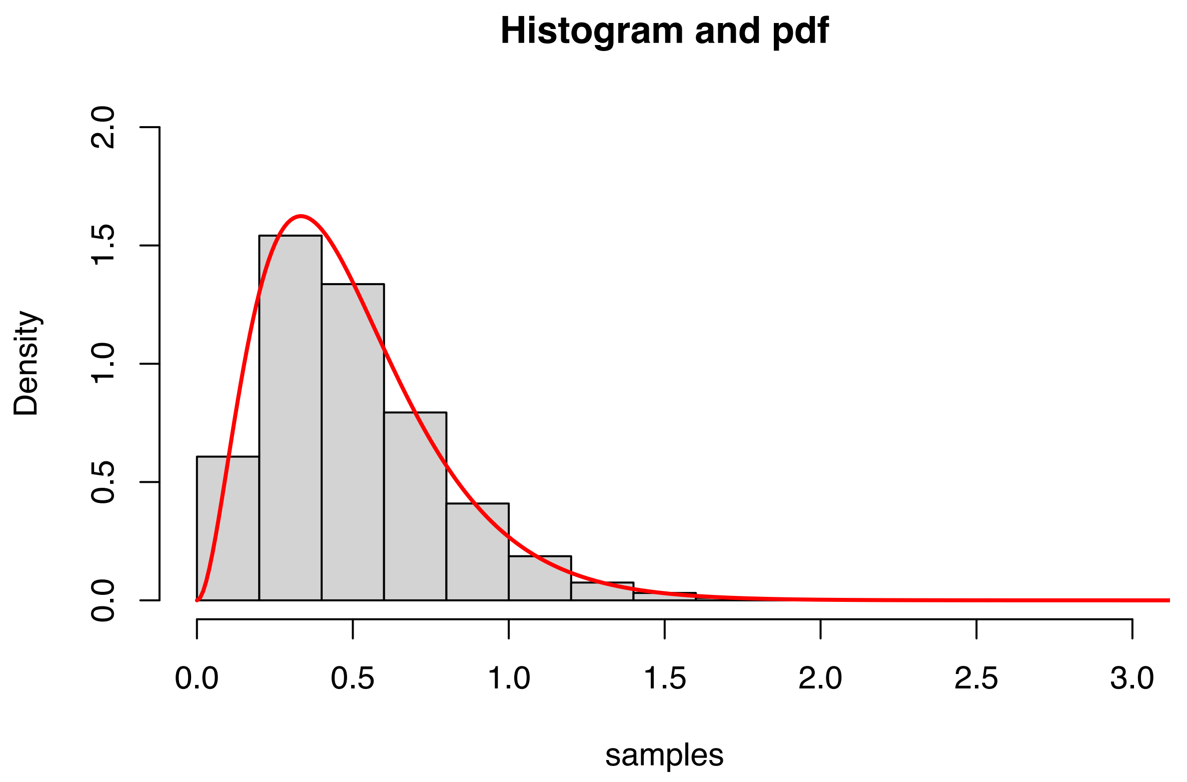
The function `generate_gamma_samples` generates a vector of n independent samples from a gamma distribution with arbitrary parameters $\alpha, \beta > 0$. We wrote this function using three of the functions created before, `rejection_sampling`, `sample_exponential` and `sample_f1`.

```
generate_gamma_samples <- function(n, alpha, beta) {
  if(alpha<1) {
    x = rejection_sampling(alpha,n)
  }
  if(alpha==1) {
    x = sample_exponential(alpha,n)
  }
  if(alpha>1) {
    x = sample_f1(alpha,n)$x
  }
  return(x/beta) # beta is an inverse scale parameter
}
```

We sampled from a $Gamma(3, 6)$ and made a histogram of the sampled values to check if the functions is correct. We also plotted the density function of the gamma distribution.

```r
density_gamma= function(alpha,beta,x) {
  ifelse(x>0, beta^alpha * x^(alpha-1) * exp(-beta*x) / (gamma(alpha)), 0)
}
```

```r
beta = 6
alpha= 3
x <- seq(1e-10,6,0.01)
y <- c()
for (i in x) {
  y <- c(y, density_gamma(alpha,beta,i))
}
samples <- generate_gamma_samples(100000, alpha, beta)
hist(samples, freq=FALSE, xlim=c(0,3), ylim=c(0,2), main="Histogram and pdf")
lines(x,y, lwd=2, col="red")
```



4.

To show that $Z \sim Beta(\alpha, \beta)$, we need to derive the probability density function (pdf) of Z and show that it matches the pdf of a Beta distribution:

$$f(z) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} z^{\alpha-1}(1-z)^{\beta-1}, \quad z \in [0,1].$$

Given that $X \sim \text{Gamma}(\alpha, 1)$ and $Y \sim \text{Gamma}(\beta, 1)$, independently, we know that their probability density functions are:

$$f_X(x) = \begin{cases} \frac{1}{\Gamma(\alpha)} x^{\alpha-1} e^{-x}, & \text{for } x > 0 \\ 0, & \text{otherwise} \end{cases}$$

$$f_Y(y) = \begin{cases} \frac{1}{\Gamma(\beta)} y^{\beta-1} e^{-y}, & \text{for } y > 0 \\ 0, & \text{otherwise} \end{cases}$$

We also know that the joint density function is given by

$$f_{X,Y}(x,y) = f_X(x) f_Y(y) = \frac{1}{\Gamma(\alpha)} x^{\alpha-1} e^{-x} \frac{1}{\Gamma(\beta)} y^{\beta-1} e^{-y}.$$

We define $z = \frac{x}{x+y}$ and $w = x + y$, so we can write $z = \frac{x}{w}$, $x = zw$ and $y = w - x = w - zw = w(1-z)$.

We compute the Jacobian matrix, in order to use the transformation formula for joint densities.

$$J = \begin{bmatrix} \frac{\partial x}{\partial z} & \frac{\partial y}{\partial z} \\ \frac{\partial x}{\partial w} & \frac{\partial y}{\partial w} \end{bmatrix} = \begin{bmatrix} w & -w \\ z & 1-z \end{bmatrix}$$

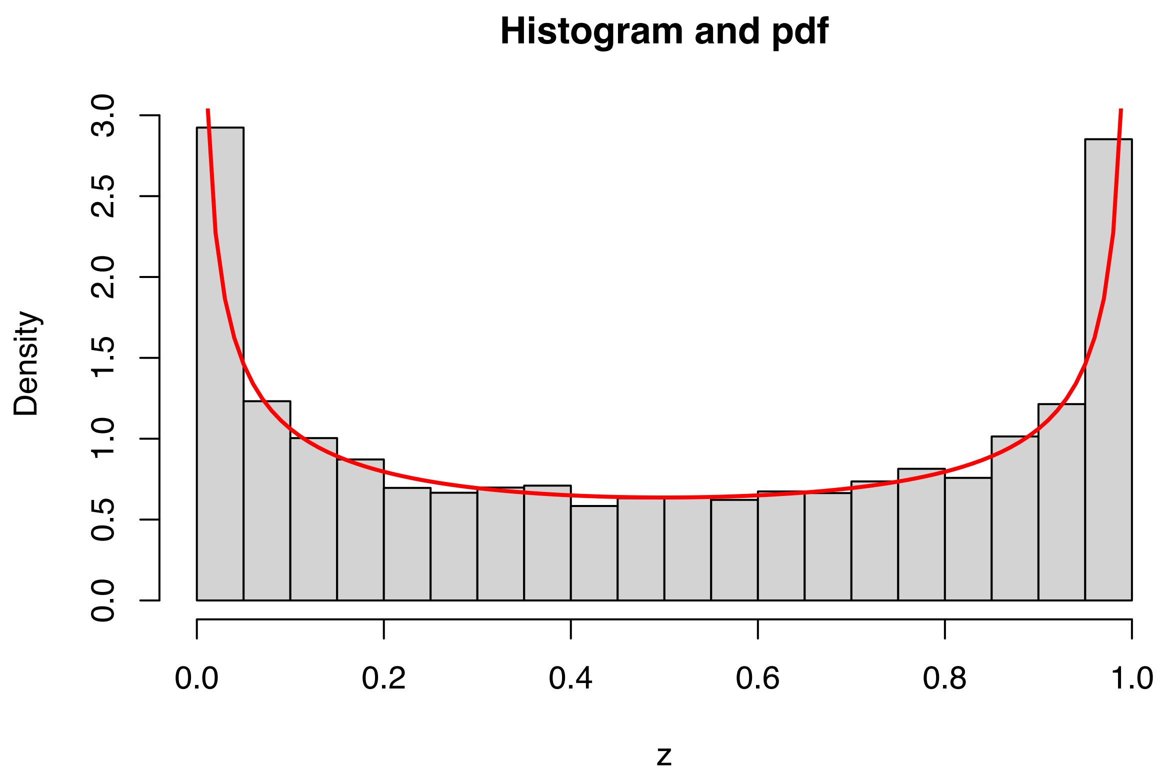
$$|J| = w(1-z) + wz = w.$$

$$f_{Z,W}(z,w) = f_{X,Y}(zw, w(1-z))|J| =$$
$$= \frac{1}{\Gamma(\alpha)\Gamma(\beta)} (zw)^{\alpha-1} [w(1-z)]^{\beta-1} e^{-w} w = \frac{1}{\Gamma(\alpha)\Gamma(\beta)} z^{\alpha-1} w^{\alpha-1} w^{\beta-1} (1-z)^{\beta-1} e^{-w} w =$$
$$= \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \frac{1}{\Gamma(\alpha+\beta)} z^{\alpha-1} (1-z)^{\beta-1} w^{\alpha+\beta-1} e^{-w} = \frac{1}{B(\alpha,\beta)} z^{\alpha-1} (1-z)^{\beta-1} \frac{1}{\Gamma(\alpha+\beta)} w^{\alpha+\beta-1} e^{-w} = f_Z(z) f_W(w)$$
$$\Rightarrow Z \sim Beta(\alpha,\beta),\ W \sim Gamma(\alpha+\beta, 1).$$

The function `sample_beta` generates a vector of n independent samples from a beta distribution with parameters α and β.

```r
sample_beta=function(n,alpha,beta){
  x=generate_gamma_samples(n,alpha,1)
  y=generate_gamma_samples(n,beta,1)
  z=x/(x+y)
  return(z)
}

density_beta=function(x,alpha,beta){
  ifelse(x>=0 && x<=1,x^(alpha-1)*(1-x)^(beta-1)/beta(alpha,beta),0)
}

n=10000
alpha=0.5
beta=0.5

x<- seq(0,1,0.01)
y <- c()
for(i in x) {
  y = c(y, density_beta(i,alpha,beta))
}
```

```r
z=sample_beta(n,alpha,beta)
hist(z,freq=FALSE, main="Histogram and pdf")
lines(x,y, lwd=2, col="red")
```



PROBLEM C: Monte Carlo integration and variance reduction

The aim of this problem is to use Monte Carlo integration to find $\theta = \mathbb{P}[X > 4]$ when $X \sim N(0,1) =: f(x)$.

1.

First we estimated θ by Monte Carlo integration using $n = 100000$ samples from the standard normal distribution. To estimate $\theta = \mathbb{E}[h(X)] = \mathbb{E}[I_{[X>4]}]$, we use the Monte Carlo estimator

$$\hat{\theta} = \mathbb{E}[\hat{h(X)}] = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}(x_i > 4),$$

where $x_i \sim N(0,1)$.

```r
N = 100000
X = sample_normal_Box_Muller(N)[,1]
Ind = as.integer(X>4)
```

We also computed a 95% confidence interval for θ based on these n samples. As the variance is estimated, we used t-student distribution with $N-1$ degree of freedom.

```r
mean1 = mean(Ind)
sd1 = sqrt(var(Ind))

# Confidence Interval (t-student N-1)
par = qt(.975, df = N-1)
CI_min1 = mean1 - par*sd1/sqrt(N)
CI_max1 = mean1 + par*sd1/sqrt(N)
cat("Estimate theta:", mean1, "\n")
```

```
## Estimate theta: 1e-05
```

```r
cat("Confidence Interval:", c(CI_min1, CI_max1), "\n")
```

```
## Confidence Interval: -9.599877e-06 2.959988e-05
```

2.

Now, we use importance sampling to estimated θ via sampling from

$$g(x) = I_{[x>4]} \cdot cxe^{-\frac{1}{2}x^2}$$

We can compute the normalising constant c as:

$$\int_{\mathbb{R}} g(x)dx = c\int_{4}^{+\infty} cxe^{-\frac{1}{2}x^2}dx = c\left[-e^{-\frac{1}{2}x^2}\right]_4^{+\infty} = ce^{-8} = 1 \Rightarrow c = e^8.$$

Recall: importance sampling

$$\widehat{\mathbb{E}[h(X)]} = \frac{1}{n}\sum_{i=1}^{n} h(X_i)\frac{f(X_i)}{g(X_i)}.$$

In order to sample from $g(x)$, we use inversion sampling. We have $G(x) = \int_{-\infty}^{x} g(t)dt = 1 - e^8 e^{-\frac{1}{2}x^2}$. And $G^{-1}(u) = \sqrt{-2\log(1-u)+16}$ for $0 < u < 1$.

```r
simulta_g = function(n){
  u = runif(n)
  x = sqrt(-2*log(u)+16)
  return(x)
}

function_g = function(x){
  c = exp(8)
  return((x>4)*c*x*exp(-x**2/2))
}

function_gaussian = function(x){
  return(exp(-x**2/2)/sqrt(2*pi))
}


importance_sampling_g = function(n){
  X = simulta_g(n)
  hX = as.integer(X >4)
  theta = hX * function_gaussian(X)/function_g(X)
  # We make sure to replace NaN value to 0
  theta[X <= 4] = 0
  return(theta)
}

sampling = importance_sampling_g(N)

mean2 = mean(sampling)
sd2 = sqrt(var(sampling))
# Confidence Interval
par = qt(.975, df = N-1)

CI_min2 = mean2 - par*sd2/sqrt(N)
CI_max2 = mean2 + par*sd2/sqrt(N)
cat("Estimate theta:", mean2, "\n")
```

```
## Estimate theta: 3.167886e-05
```

```
cat("Confidence Interval:", c(CI_min2, CI_max2), "\n")
```

```
## Confidence Interval: 3.166926e-05 3.168847e-05
```

We compare the precisions of the two estimator, which is defined by the inverse of the variance. We want to

$$\frac{1}{\widehat{\mathbb{V}_1[h(X)]}} = \frac{1}{\widehat{\mathbb{V}_2[h(X)]}}$$

$$\widehat{\mathbb{V}_1[h(X)]} = \mathbb{V}\left[\frac{1}{N_1}\sum_{i=1}^{N_1} h(X_i)\right] = \frac{1}{N_1^2}\mathbb{V}\left[\sum_{i=1}^{N_1} h(X_i)\right] \underbrace{=}_{i.i.d} \frac{1}{N_1^2}\sum_{i=1}^{N_1}\mathbb{V}\left[h(X_i)\right] = \frac{1}{N_1}\underbrace{\widehat{\mathbb{V}[h(X)]}}_{known}$$

Similary,

$$\widehat{\mathbb{V}_2[h(X)]} = \mathbb{V}\left[\frac{1}{N_2}\sum_{i=1}^{N_2} h(X_i)\frac{f(X_i)}{g(X_i)}\right] = \frac{1}{N_2}\underbrace{\widehat{\mathbb{V}[h(X)]}}_{known}$$

That is why $N_1 = \frac{\widehat{\mathbb{V}_1[h(X)]}}{\widehat{\mathbb{V}_2[h(X)]}} N_2$.

```r
N_1 = N*sd1**2/sd2**2
cat("Samples needed:", N_1, "\n")
```

```
## Samples needed: 416253782244
```

```r
cat("By comparison N_1/N:", N_1/N, "\n")
```

```
## By comparison N_1/N: 4162538
```

This result implies that the importance sampling is closer to the real value, and it has a smaller variance.

3.

Now we combine the importance sampling in the previous item with the use of antithetic variates. This method should reduce the variance.

$$\widehat{\mathbb{E}[h(X)]} = \frac{1}{2n}\sum_{i=1}^{n}\left(h(X_i)\frac{f(X_i)}{g(X_i)} + h(X_i^*)\frac{f(X_i^*)}{g(X_i^*)}\right)$$

```r
importance_sampling_antithetic_g = function(n){
  u = runif(n)
  x = sqrt(-2*log(u)+16)
  theta = function_gaussian(x)/function_g(x)
  theta[x <= 4] = 0

  x_star = sqrt(-2*log(1-u)+16)
  theta_star = function_gaussian(x_star)/function_g(x_star)
  theta_star[x_star <= 4] = 0

  return(c(theta,theta_star))
}
```

In order to get a comparison, we generated $n = 50000$ pairs of samples from $g(x)$. It is a fair comparison because the antithetic sampling generates two samples $x = G^{-1}(u)$ and $x^* = G^{-1}(1-u)$ for each $u \sim U(0,1)$.

```r
N_3 = N/2
sampling_anth = importance_sampling_antithetic_g(N_3)
mean3 = mean(sampling_anth)
sd3 = sqrt(var(sampling_anth))
# Confidence Interval
par = qt(.975, df = N_3-1)

CI_min3 = mean3 - par*sd3/sqrt(N_3)
CI_max3 = mean3 + par*sd3/sqrt(N_3)
cat("Estimate theta:", mean3, "\n")
```

```
## Estimate theta: 3.166758e-05
```

```r
cat("Confidence Interval:", c(CI_min3, CI_max3), "\n")
```

```
## Confidence Interval: 3.165383e-05 3.168132e-05
```

```r
cat("Compare estimate theta:", c(mean1,mean2, mean3), "\n")
```

```
## Compare estimate theta: 1e-05 3.167886e-05 3.166758e-05
```

```r
cat("Compare precision:", 1/c(sd1**2, sd2**2, sd3**2), "\n")
```

```
## Compare precision: 1e+05 416253782244 406641427542
```

Digression on $Cov(h(F^{-1}(u)), h(F^{-1}(1-u))) \leq 0$.

Let's compute the correlation p value defined as:

$$\begin{aligned} p := \mathbb{C}\text{ov}\left[h(G^{-1}(U)), h(G^{-1}(1-U))\right] &= \mathbb{C}\text{ov}\left[h\left(\sqrt{16-2\log(U)}\right), h\left(\sqrt{16-2\log(1-U)}\right)\right] \\ &= \mathbb{C}\text{ov}\left[I_{[\sqrt{16-2\log(U)}>4]}, I_{[\sqrt{16-2\log(1-U)}>4]}\right] \end{aligned}$$

But as, $U \in [0,1]$, we have $\sqrt{16-2\log(U)} > 4 \Leftrightarrow U < 1$ and $\sqrt{16-2\log(1-U)} > 4 \Leftrightarrow U > 0$, which are always true. Thus

$$\mathbb{C}\text{ov}\left[h(G^{-1}(U)), h(G^{-1}(1-U))\right] = \mathbb{C}\text{ov}\left[1,1\right] = 0 \leq 0$$

.

But if we want to compare another correlation, there exists $k > 0$ since $h(X)\frac{f(X)}{g(X)} = \frac{h(X)}{c\sqrt{2\pi}X} = k \cdot \frac{h(X)}{X}$.

$$\begin{aligned} p' &:= \mathbb{C}\text{ov}\left[h(G^{-1}(U))\frac{f(G^{-1}(U))}{g(G^{-1}(U))}, h(G^{-1}(1-U))\frac{f(G^{-1}(1-U))}{g(G^{-1}(1-U))}\right] \\ &= k \cdot \mathbb{C}\text{ov}\left[\frac{1}{\sqrt{16-2\log(U)}}, \frac{1}{\sqrt{16-2\log(1-U)}}\right] \\ &= \mathbb{E}\left[\frac{1}{\sqrt{16-2\log(U)}\sqrt{16-2\log(1-U)}}\right] - \mathbb{E}\left[\frac{1}{\sqrt{16-2\log(U)}}\right]^2 \\ &< 0 \end{aligned}$$

According to Wolfram Alpha $p' \leq 0$.

$$\begin{aligned} \mathbb{V}\left[\mathbb{E}\left[\widehat{h(X)\frac{f(X)}{g(X)}}\right]\right] &= \frac{1}{2n}\mathbb{V}\left[h(X)\frac{f(X)}{g(X)}\right] + \frac{1}{4n^2}2n \cdot \mathbb{C}\text{ov}\left[h(X)\frac{f(X)}{g(X)}, h(X^*)\frac{f(X^*)}{g(X^*)}\right] \\ &\leq \frac{1}{2n}\mathbb{V}\left[h(X)\frac{f(X)}{g(X)}\right]\underbrace{(1+p)}_{\leq 1} \end{aligned}$$

PROBLEM D: Rejection sampling and importance sampling

1.

In this problem we considered the data of Rao, that contains 197 counts, classified into four categories, and assumed to be multinomial distributed.

The multinomial mass function is given by

$$f(\mathbf{y}|\theta) \propto (2+\theta)^{y_1}(1-\theta)^{y_2+y_3}\theta^{y_4}.$$

Using a uniform prior on $(0,1)$ for θ, the observed posterior density is:

$$f(\theta|\mathbf{y}) \propto (2+\theta)^{y_1}(1-\theta)^{y_2+y_3}\theta^{y_4}, \text{ for } \theta \in (0,1).$$

We are interested in the posterior mean

$$\mathbb{E}(\theta|\mathbf{y}).$$

First, we use rejection sampling to simulate from

$$f(\theta|\mathbf{y})$$

using a $U(0,1)$ density as the proposal density $g(\theta)$.

We start by writing the function `posterior`, that takes as input θ, and returns the value of

$$(2+\theta)^{y_1}(1-\theta)^{y_2+y_3}\theta^{y_4}$$

```
y=c(125, 18, 20, 34)
posterior=function(theta){
  return(exp(log(2+theta)*y[1]+log(1-theta)*(y[2]+y[3])+log(theta)*y[4]))
}
```

In order to apply the rejection method, we have found the value c such that $f(\theta|y) \leq cg(\theta),\ \forall\theta \in (0,1)$ by computing the derivative of $\ln f$.

$$\ln f = y_1\ln(2+\theta) + (y_2+y_3)\ln(1-\theta) + y_4\ln\theta.$$

$$\frac{d\ln f}{d\theta} = \frac{y_1}{2+\theta} - \frac{y_2+y_3}{1-\theta} + \frac{y_4}{\theta} = \frac{\theta(1-\theta)y_1 - \theta(2+\theta)(y_2+y_3) + y_4(2+\theta)(1-\theta)}{\theta(1-\theta)(2+\theta)} =$$

$$= \frac{y_1\theta - y_1\theta^2 - 2(y_2+y_3)\theta - (y_2+y_3)\theta^2 + 2y_4 - y_4\theta - y_4\theta^2}{\theta(1-\theta)(2+\theta)} =$$

$$= \frac{-(y_1+y_2+y_3+y_4)\theta^2 + (y_1-2y_2-2y_3-y_4)\theta + 2y_4}{\theta(1-\theta)(2+\theta)},$$

$$\theta_{1,2} = \frac{-y_1+2y_2+2y_3+y_4 \pm \sqrt{(y_1-2y_2-2y_3-y_4)^2 + 8(y_1+y_2+y_3+y_4)y_4}}{-2(y_1+y_2+y_3+y_4)}.$$

We defined

$$a = -(y_1+y_2+y_3+y_4),\ b = y_1-2y_2-2y_3-y_4,\ c = 2y_4.$$

Finally, we wrote the rejection sampling function `rejection`.

```
rejection=function(y){
  a=-sum(y)
  b=y[1]-2*y[2]-2*y[3]-y[4]
  c=2*y[4]
  theta_max=(-b-sqrt(b**2-4*a*c))/(2*a)
  c=posterior(theta_max)
  repeat{
    u=runif(1,0,1)
    theta=runif(1,0,1)
    alpha=posterior(theta)/c
    if(alpha>u)
      {break()}
  }
  return(theta)
}
```

2.

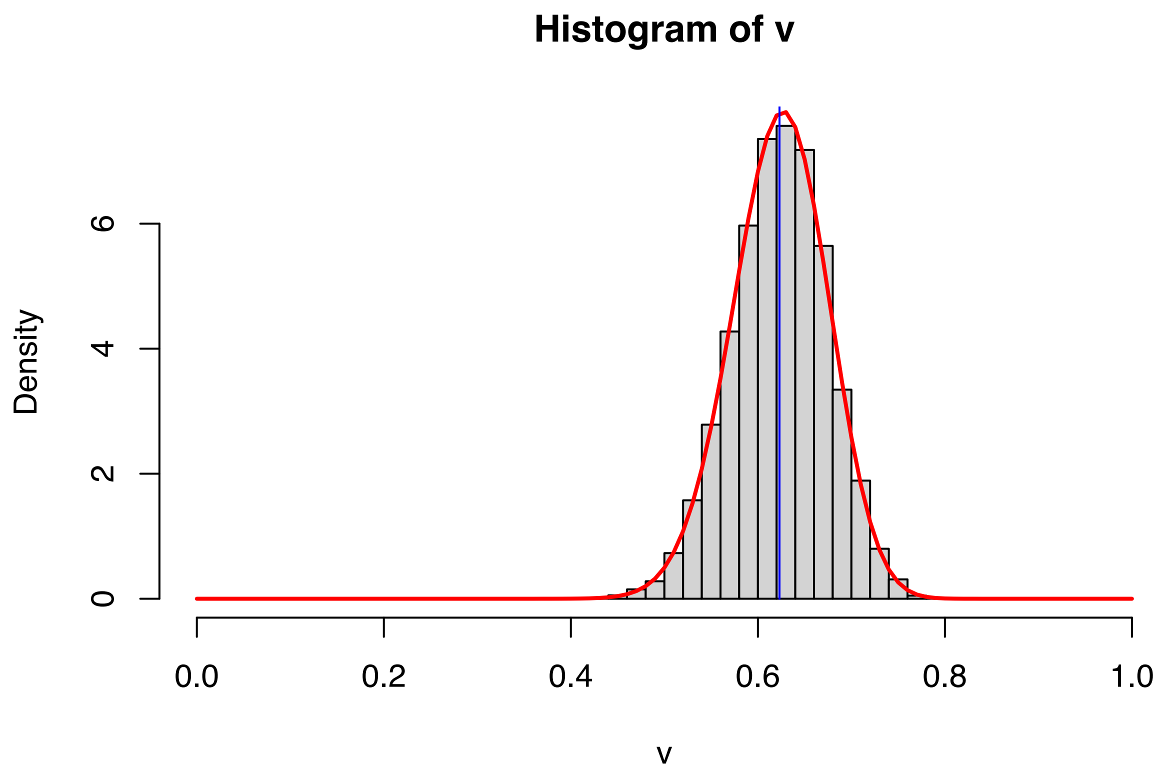
We generated M samples from $f(\theta|y)$ and we estimated the posterior mean by Monte-Carlo integration, using the following formula

$$\hat{\theta} = \frac{1}{M}\sum_{i=1}^{M} f(\theta_i|y).$$

```
M=10000 # number of samples
v=c()
for (i in 1:M){
  v=c(v, rejection(y)) # vector of generated samples
}
theta_hat=mean(v)
theta_hat # estimated posterior mean
```

```
## [1] 0.6230921
```

```
cost=1/integrate(posterior,0,1)$value # normalising constant
x=seq(0,1,0.01)
z=c()
for(i in x){
  z=c(z,posterior(i)*cost)
}
hist(v,freq=FALSE,xlim=c(0,1)) # histogram of the samples
lines(x,z,col="red",lwd=2) # theoretical posterior density distribution
lines(c(theta_hat,theta_hat),c(0,10),col="blue") # estimated posterior mean
```



```
integrand=function(theta){
  return(cost*exp(log(theta)+log(2+theta)*y[1]+log(1-theta)*(y[2]+y[3])+log(theta)*y[4]))
}
mean=integrate(integrand,0,1)
mean$value # mean computed using the numerical integration
```

```
## [1] 0.6228061
```

3.

We derived practically the average of the generated random numbers, in order to obtain one sample of $f(\theta|y)$, by computing the mean of the number of iterations of $M = 10000$ samples.

```
rejection1=function(y){
  a=-sum(y)
  iter=0
  b=y[1]-2*y[2]-2*y[3]-y[4]
  c=2*y[4]
  theta_max=(-b-sqrt(b**2-4*a*c))/(2*a)
  c=posterior(theta_max)
  repeat{
    u=runif(1,0,1)
    theta=runif(1,0,1)
    alpha=posterior(theta)/c
    iter=iter+1
    if(alpha>u)
    {break()}
  }
  return(c(theta,iter))
}

M=10000
v=c()
for (i in 1:M){
  x=rejection1(y)
  v=c(v, x[2]) # vector of number of iterations
}
mean(v) #average of iteration for one sample
```

```
## [1] 7.8045
```

In the following code, we derived theoretically the average of the generated random numbers, by computing the overall acceptance probability

$$P(U < \alpha) = \frac{1}{c} \int f(\theta|y) d\theta.$$

```
a=-sum(y)
iter=0
b=y[1]-2*y[2]-2*y[3]-y[4]
c=2*y[4]
theta_max=(-b-sqrt(b**2-4*a*c))/(2*a)
c=posterior(theta_max)
over_acc_prob=integrate(posterior,0,1)$value/c
num = 1/over_acc_prob
num
```

```
## [1] 7.799308
```

4.

Now we consider the $Beta(1,5)$ prior for θ. Thus, the observed posterior density is:

$$f'(\theta|\mathbf{y}) \propto (2+\theta)^{y_1}(1-\theta)^{y_2+y_3+4}\theta^{y_4}, \text{ for } \theta \in (0,1).$$

In order to estimate the posterior mean under the new prior, we used the importance sampling weights, by generating from $f(\theta|y)$. We used the following formula

$$\hat{\theta} = \frac{1}{M}\sum_{i=1}^{M} \theta_i \frac{f'(\theta_i|y)}{f(\theta_i|y)}.$$

```
M=10000
a=1
b=5

posterior_new=function(theta){
  return(exp(log(2+theta)*y[1]+log(1-theta)*(y[2]+y[3]+4)+log(theta)*(y[4])))
}

costante_2=1/integrate(posterior_new,0,1)$value
```

```r
media=function(M){
  sum=0
  for(i in 1:M){
    v=rejection(y)
    sum=sum+v*costante_2*posterior_new(v)/(cost*posterior(v))
  }
  return(sum/M)
}

theta_first_hat=media(M)
theta_first_hat
```

```
## [1] 0.5916327
```

We can compare this result with the following theoretical result, computed using numerical integration.

```r
integrand_2=function(theta){
  return(exp(log(theta)+log(2+theta)*y[1]+log(1-theta)*(y[2]+y[3]+b-1)+log(theta)*(y[4]+a-1)))
}
media_2=integrate(integrand_2,0,1)$value*costante_2
media_2
```

```
## [1] 0.5959316
```

TMA4300 Project 2

Elena Dami, Marco Pozzetto & Marijan Soric

PROBLEM 1

In this problem, we will look at a portion of the Tokyo rainfall dataset, a famous dataset with daily rainfall data from 1951–1989. We will consider the response to be whether the amount of rainfall exceeded 1 *mm* over the given time period:

$$y_t|x_t \sim \text{Bin}(n_t, \pi(x_t)), \quad \pi(x_t) = \frac{e^{x_t}}{1+e^{x_t}} = \frac{1}{1+e^{-x_t}},$$

for n_t being 10 for $t = 60$ (February 29th) and 39 for all other days in the year, and $\pi(x_t)$ being the probability of rainfall exceeding 1 *mm* for days $t = 1, ..., 366$.
We can note that x_t is the logit probability of rainfall exceeding 1 *mm* and can be obtained from $\pi(x_t)$ via the logit function: $x_t = \log\left(\frac{\pi(x_t)}{1-\pi(x_t)}\right)$.
We assume conditional independence among the $y_t|x_t$ for all $t = 1, ..., 366$.

a)

We begin by exploring the dataset. We check for any missing data, assess the dimensions of the dataset, and examine some summary statistics.

```r
head(rain)
```

```
##   day n.years n.rain
## 1   1      39      8
## 2   2      39      7
## 3   3      39      8
## 4   4      39     11
## 5   5      39      8
## 6   6      39      6
```

```r
sum(is.na(rain))
```

```
## [1] 0
```

```r
dim(rain)
```

```
## [1] 366   3
```

```r
summary(rain)
```

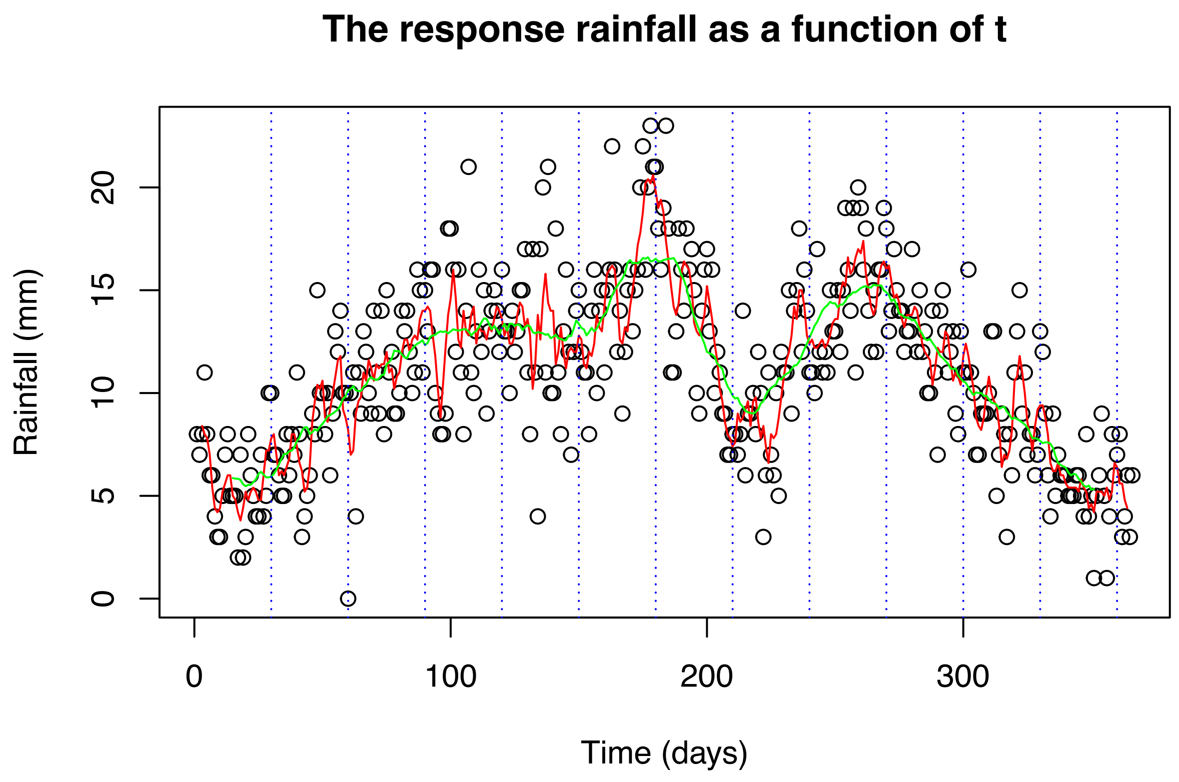
```
##       day            n.years          n.rain
##  Min.   :  1.00   Min.   :10.00   Min.   : 0.00
##  1st Qu.: 92.25   1st Qu.:39.00   1st Qu.: 8.00
##  Median :183.50   Median :39.00   Median :11.00
##  Mean   :183.50   Mean   :38.92   Mean   :10.98
##  3rd Qu.:274.75   3rd Qu.:39.00   3rd Qu.:14.00
##  Max.   :366.00   Max.   :39.00   Max.   :23.00
```

We proceed by making a plot of the rainfall data against time (days). This code calculates and plots the rainfall data along with the 5-day and 30-day rolling means, and adds vertical dotted lines at 30-day intervals to mark the months on the x-axis.

```r
library("zoo")
```

```
##
## Caricamento pacchetto: 'zoo'

## I seguenti oggetti sono mascherati da 'package:base':
##
##     as.Date, as.Date.numeric
```

```r
rain_5 <- rollmean(rain$n.rain, k = 5, fill = NA)
rain_30 <- rollmean(rain$n.rain, k = 30, fill = NA)
plot(rain$day, rain$n.rain,
     main = "The response rainfall as a function of t",
     xlab = "Time (days)",
     ylab = "Rainfall (mm)")
lines(rain$day, rain_5, type = 'l', col = "red")
lines(rain$day, rain_30, type = 'l', col = "green")
abline(v = seq(30, max(rain$day), by = 30), lty = "dotted", col = "blue")
```



Rainfall is lower at the beginning and at the end of the year, with a distinct rainy season occurring during the summer months.
Additionally, there appears to be greater variation in rainfall during the middle months of the year compared to the beginning and end.

b)

Now we look for the likelihood of y_t depending on parameters $\pi(x_t)$ for $t = 1, ..., 366$.

We know that $(y_t|x_t) \sim \text{Bin}(n_t, \pi(x_t))$, so we have

$$p(y_t|x_t) = \binom{n_t}{y_t} \pi(x_t)^{y_t}(1-\pi(x_t))^{n_t-y_t},$$

which we can also write as

$$p(y_t|x_t) = \binom{n_t}{y_t} \left(\frac{e^{x_t}}{1+e^{x_t}}\right)^{y_t} \left(\frac{1}{1+e^{x_t}}\right)^{n_t-y_t} = \binom{n_t}{y_t} \frac{(e^{x_t})^{y_t}}{(1+e^{x_t})^{n_t}}.$$

Thus, the likelihood is

$$p(\mathbf{y}|\mathbf{x}) = \prod_{t=1}^{T} \binom{n_t}{y_t} \frac{(e^{x_t})^{y_t}}{(1+e^{x_t})^{n_t}}.$$

We will apply a Bayesian hierarchical model to the dataset, using a random walk of order 1 (RW(1)) to model the trend on a logit scale,

$$x_t = x_{t-1} + u_t,$$

for $u_t \overset{iid}{\sim} \mathcal{N}(0, \sigma_u^2)$, so that,

$$p(\mathbf{x}|\sigma_u^2) \propto \prod_{t=2}^{T} \frac{1}{\sigma_u} \exp\{-\frac{1}{2\sigma_u^2}(x_t - x_{t-1})^2\}. \tag{1}$$

We will place the following inverse gamma prior on σ_u^2,

$$p(\sigma_u^2) = \frac{\beta^\alpha}{\Gamma(\alpha)} (1/\sigma_u^2)^{\alpha+1} \exp\{-\beta/\sigma_u^2\},$$

for shape α and scale β. Let $\mathbf{y} = (y_1, y_2, ...y_T)^T,\ \mathbf{x} = (x_1, ..., x_T)^T$ and $\pi = (\pi(x_1), ..., \pi(x_T))^T$.

c)

We now look for the conditional $p(\sigma_u^2|\mathbf{x}, \mathbf{y})$.

We know that

$$p(\sigma_u^2|\mathbf{x}, \mathbf{y}) = \frac{p(\sigma_u^2, \mathbf{x}, \mathbf{y})}{p(\mathbf{x}, \mathbf{y})} \propto p(\sigma_u^2, \mathbf{x}, \mathbf{y}),$$

since $p(\mathbf{x}, \mathbf{y})$ is constant with respect to σ_u^2.

We have that

$$p(\sigma_u^2, \mathbf{x}, \mathbf{y}) = p(\mathbf{y}|\sigma_u^2, \mathbf{x})p(\mathbf{x}|\sigma_u^2)p(\sigma_u^2).$$

Since we know all terms on the right of the equation, we can substitute their expressions

$$p(\sigma_u^2, \mathbf{x}, \mathbf{y}) = \prod_{t=1}^{T} \binom{n_t}{y_t} \frac{\exp{(x_t)}^{y_t}}{(1+\exp{(x_t)})^{n_t}} \prod_{t=2}^{T} \frac{1}{\sigma_u} \exp\left\{-\frac{1}{2\sigma_u^2}(x_t - x_{t-1})^2\right\} \frac{\beta^\alpha}{\Gamma(\alpha)} \frac{\exp\{-\beta/\sigma_u^2\}}{(\sigma_u^2)^{\alpha+1}}.$$

Considering only the terms that depend on σ_u^2 we obtain

$$p(\sigma_u^2|\mathbf{x},\mathbf{y}) \propto \left(\frac{1}{\sigma_u^2}\right)^{\frac{1}{2}(T-1)} e^{-\frac{1}{2\sigma_u^2}\sum_{t=2}^{T}(x_t-x_{t-1})^2} \left(\frac{1}{\sigma_u^2}\right)^{\alpha+1} e^{-\frac{\beta}{\sigma_u^2}}.$$

Reformulating this expression we get

$$p(\sigma_u^2|\mathbf{x},\mathbf{y}) \propto \left(\frac{1}{\sigma_u^2}\right)^{\alpha+\frac{1}{2}(T-1)+1} e^{-\frac{1}{\sigma_u^2}(\beta+\frac{1}{2}\sum_{t=2}^{T}(x_t-x_{t-1})^2)}.$$

Thus, we obtained that $\sigma_u^2|\mathbf{x},\mathbf{y}$ follows a Inverse Gamma distribution for shape $\left(\alpha+\frac{T-1}{2}\right)$ and scale $\left(\beta+\frac{1}{2}\sum_{t=2}^{T}(x_t-x_{t-1})^2\right)$.

d)

We consider the conditional prior proposal distribution $Q(\mathbf{x}'_{\mathcal{I}}|\mathbf{x}_{-\mathcal{I}},\sigma_u^2,\mathbf{y}) = p(\mathbf{x}'_{\mathcal{I}}|\mathbf{x}_{-\mathcal{I}},\sigma_u^2)$, where $\mathbf{x}'_{\mathcal{I}}$ is the proposed values for $\mathbf{x}_{\mathcal{I}}$, $\mathcal{I}\subseteq\{1,...,366\}$ is a set of time indices and $\mathbf{x}_{-\mathcal{I}}=\mathbf{x}_{\{1,...,366\}\setminus\mathcal{I}}$ is $\mathbf{x}$ subset to include all indices other than those in $\mathcal{I}$.

We will now compute the resulting acceptance probability $\alpha(\mathbf{x}'_{\mathcal{I}}|\mathbf{x}_{-\mathcal{I}},\sigma_u^2,\mathbf{y})$.

We begin with the definition of the acceptance probability with iterative conditioning.

$$\alpha(\mathbf{x}'_{\mathcal{I}}|\mathbf{x}_{-\mathcal{I}},\sigma_u^2,\mathbf{y}) = \min\left(1, \frac{Q(\mathbf{x}_{\mathcal{I}}|\mathbf{x}_{-\mathcal{I}},\sigma_u^2,\mathbf{y})p(\mathbf{x}'_{\mathcal{I}}|\mathbf{x}_{-\mathcal{I}},\sigma_u^2,\mathbf{y})}{Q(\mathbf{x}'_{\mathcal{I}}|\mathbf{x}_{-\mathcal{I}},\sigma_u^2,\mathbf{y})p(\mathbf{x}_{\mathcal{I}}|\mathbf{x}_{-\mathcal{I}},\sigma_u^2,\mathbf{y})}\right). \tag{2}$$

Then, we compute $p(\mathbf{x}'_{\mathcal{I}}|\mathbf{x}_{-\mathcal{I}},\sigma_u^2,\mathbf{y})$,

$$\begin{aligned} p(\mathbf{x}'_{\mathcal{I}}|\mathbf{x}_{-\mathcal{I}},\sigma_u^2,\mathbf{y}) &= \frac{p(\mathbf{x}'_{\mathcal{I}},\mathbf{x}_{-\mathcal{I}},\sigma_u^2,\mathbf{y})}{p(\mathbf{x}_{-\mathcal{I}},\sigma_u^2,\mathbf{y})} \\ &= \frac{p(\mathbf{y}|\mathbf{x}'_{\mathcal{I}},\mathbf{x}_{-\mathcal{I}},\sigma_u^2)p(\mathbf{x}'_{\mathcal{I}}|\mathbf{x}_{-\mathcal{I}},\sigma_u^2)p(\mathbf{x}_{-\mathcal{I}},\sigma_u^2)}{p(\mathbf{y}|\mathbf{x}_{-\mathcal{I}},\sigma_u^2)p(\mathbf{x}_{-\mathcal{I}},\sigma_u^2)} \\ &= \frac{p(\mathbf{y}|\mathbf{x}'_{\mathcal{I}},\mathbf{x}_{-\mathcal{I}})p(\mathbf{x}'_{\mathcal{I}}|\mathbf{x}_{-\mathcal{I}},\sigma_u^2)}{p(\mathbf{y}|\mathbf{x}_{-\mathcal{I}},\sigma_u^2)} \\ &= \frac{p(\mathbf{y}_{\mathcal{I}}|\mathbf{x}'_{\mathcal{I}})p(\mathbf{y}_{-\mathcal{I}}|\mathbf{x}_{-\mathcal{I}})p(\mathbf{x}'_{\mathcal{I}}|\mathbf{x}_{-\mathcal{I}},\sigma_u^2)}{p(\mathbf{y}|\mathbf{x}_{-\mathcal{I}},\sigma_u^2)}. \end{aligned} \tag{3}$$

Using the assumption on conditional independence among the $y_t|x_t,\ t=1,...,366$ and that it does not depend upon σ_u^2.

Similarly,

$$p(\mathbf{x}_{\mathcal{I}}|\mathbf{x}_{-\mathcal{I}},\sigma_u^2,\mathbf{y}) = \frac{p(\mathbf{y}_{\mathcal{I}}|\mathbf{x}_{\mathcal{I}})p(\mathbf{y}_{-\mathcal{I}}|\mathbf{x}_{-\mathcal{I}})p(\mathbf{x}_{\mathcal{I}}|\mathbf{x}_{-\mathcal{I}},\sigma_u^2)}{p(\mathbf{y}|\mathbf{x}_{-\mathcal{I}},\sigma_u^2)}. \tag{4}$$

Dividing (3) by (4), we get

$$\frac{p(\mathbf{y}_{\mathcal{I}}|\mathbf{x}'_{\mathcal{I}})p(\mathbf{x}'_{\mathcal{I}}|\mathbf{x}_{-\mathcal{I}},\sigma_u^2)}{p(\mathbf{y}_{\mathcal{I}}|\mathbf{x}_{\mathcal{I}})p(\mathbf{x}_{\mathcal{I}}|\mathbf{x}_{-\mathcal{I}},\sigma_u^2)}.$$

Substituting the results into (2), we get

$$\begin{aligned} \alpha(\mathbf{x}'_{\mathcal{I}}|\mathbf{x}_{-\mathcal{I}},\sigma_u^2,\mathbf{y}) &= \min\left(1, \frac{p(\mathbf{x}_{\mathcal{I}}|\mathbf{x}_{-\mathcal{I}},\sigma_u^2)p(\mathbf{y}_{\mathcal{I}}|\mathbf{x}'_{\mathcal{I}})p(\mathbf{x}'_{\mathcal{I}}|\mathbf{x}_{-\mathcal{I}},\sigma_u^2)}{p(\mathbf{x}'_{\mathcal{I}}|\mathbf{x}_{-\mathcal{I}},\sigma_u^2)p(\mathbf{y}_{\mathcal{I}}|\mathbf{x}_{\mathcal{I}})p(\mathbf{x}_{\mathcal{I}}|\mathbf{x}_{-\mathcal{I}},\sigma_u^2)}\right) \\ &= \min\left(1, \frac{p(\mathbf{y}_{\mathcal{I}}|\mathbf{x}'_{\mathcal{I}})}{p(\mathbf{y}_{\mathcal{I}}|\mathbf{x}_{\mathcal{I}})}\right). \end{aligned} \tag{5}$$

e)

We can note that the density specified by (1) is improper, it follows that it can be rewritten as

$$p(\mathbf{x}|\sigma_u^2) \propto \exp\left\{-\frac{1}{2}\mathbf{x}^T Q\mathbf{x}\right\}$$

resembling a multivariate normal density but where the impropriety of the density translates into the precision matrix

$$Q = \frac{1}{\sigma_u^2}\begin{pmatrix} 1 & -1 & & & \\ -1 & 2 & -1 & & \\ & & \ddots & & \\ & & -1 & 2 & -1 \\ & & & -1 & 1 \end{pmatrix}.$$

Partitioning the components of $\mathbf{x}$ into two subvectors like

$$\mathbf{x} = \begin{pmatrix}\mathbf{x}_A \\ \mathbf{x}_B\end{pmatrix},$$

and partitioning the precision matrix in the same way like

$$\mathbf{Q} = \begin{pmatrix} Q_{AA} & Q_{AB} \\ Q_{BA} & Q_{BB}\end{pmatrix},$$

we now want to show that the $\mathbf{x}_A$ has a proper multivariate normal distribution conditional on $\mathbf{x}_B$.

We know that $(\mathbf{x}_A, \mathbf{x}_B) \sim \mathcal{N}_{a+b}(0, Q^{-1})$ with density probability:

$$f(\mathbf{x}_A, \mathbf{x}_B) = \frac{1}{\sqrt{(2\pi)^{a+b}|Q^{-1}|}} e^{-\frac{1}{2}(\mathbf{x}_A \quad \mathbf{x}_B)^T Q (\mathbf{x}_A \quad \mathbf{x}_B)}.$$

$$f(\mathbf{x}_A|\mathbf{x}_B) = \frac{f(\mathbf{x}_A, \mathbf{x}_B)}{f(\mathbf{x}_B)}$$

$$f(\mathbf{x}_A|\mathbf{x}_B) \propto \frac{e^{-\frac{1}{2}(\mathbf{x}_A \quad \mathbf{x}_B)^T Q (\mathbf{x}_A \quad \mathbf{x}_B)}}{e^{-\frac{1}{2}\mathbf{x}_B^T Q_{BB}\mathbf{x}_B}}$$

$$f(\mathbf{x}_A|\mathbf{x}_B) \propto \exp\left\{-\frac{1}{2}\left((\mathbf{x}_A \quad \mathbf{x}_B)^T Q (\mathbf{x}_A \quad \mathbf{x}_B) - \mathbf{x}_B^T Q_{BB}\mathbf{x}_B\right)\right\}$$

$$f(\mathbf{x}_A|\mathbf{x}_B) \propto \exp\left\{-\frac{1}{2}\left((\mathbf{x}_A \quad \mathbf{x}_B)^T \begin{pmatrix} Q_{AA} & Q_{AB} \\ Q_{BA} & Q_{BB}\end{pmatrix} (\mathbf{x}_A \quad \mathbf{x}_B) - \mathbf{x}_B^T Q_{BB}\mathbf{x}_B\right)\right\}$$

$$f(\mathbf{x}_A|\mathbf{x}_B) \propto \exp\left\{-\frac{1}{2}\left(\mathbf{x}_A^T Q_{AA}\mathbf{x}_A + \mathbf{x}_A^T Q_{AB}\mathbf{x}_B + \mathbf{x}_B^T Q_{BA}\mathbf{x}_A + \mathbf{x}_B^T Q_{BB}\mathbf{x}_B - \mathbf{x}_B^T Q_{BB}\mathbf{x}_B\right)\right\}$$

$$f(\mathbf{x}_A|\mathbf{x}_B) \propto \exp\left\{-\frac{1}{2}\left(\mathbf{x}_A^T Q_{AA}\mathbf{x}_A + \mathbf{x}_A^T Q_{AB}\mathbf{x}_B + \mathbf{x}_B^T Q_{BA}\mathbf{x}_A\right)\right\}.$$

If $x_A|x_B \sim \mathcal{N}_{\dashv}(\mu_{A|B}, Q_{A|B}^{-1})$ holds, it is true that:

$$f(\mathbf{x}_A|\mathbf{x}_B) \propto \exp\left\{-\frac{1}{2}\left((\mathbf{x}_A - \mu_{A|B})^T Q_{A|B}(\mathbf{x}_A - \mu_{A|B})\right)\right\} =$$

$$\exp\left\{-\frac{1}{2}\left(\mathbf{x}_A^T Q_{A|B}\mathbf{x}_A - \mathbf{x}_A^T Q_{A|B}\mu_{A|B} - \mu_{A|B}^T Q_{A|B}\mathbf{x}_A + \mu_{A|B}^T Q_{A|B}\mu_{A|B}\right)\right\}$$

By identification:

$$\begin{cases} Q_{A|B} = Q_{AA} \\ -Q_{AA}\ \mu_{A|B} = Q_{AB}\mathbf{x}_B \end{cases}$$

$$\begin{cases} Q_{A|B} = Q_{AA}, \\ \mu_{A|B} = -Q_{AA}^{-1} Q_{AB}\mathbf{x}_B. \end{cases}$$

Summarizing, we showed that if

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_A \\ \mathbf{x}_B \end{pmatrix} \sim \mathcal{N}_{\dashv+\lfloor}\left(\begin{pmatrix} \mu_A \\ \mu_B \end{pmatrix}, \begin{pmatrix} Q_{AA} & Q_{AB} \\ Q_{BA} & Q_{BB} \end{pmatrix}^{-1}\right),$$

then $\mathbf{x}_A|\mathbf{x}_B$ is a multivariate normal distribution with mean and precision

$$\mu_{A|B} = -Q_{AA}^{-1} Q_{AB}\ x_B \quad \text{and} \quad Q_{A|B} = Q_{AA}.$$

f)

Our goal now is to implement a MCMC sampler for the posterior $p(\pi, \sigma_u^2|\mathbf{y})$ using MH steps for individual x_t parameters using the conditional prior, $p(x_t|\mathbf{x}_{-t}, \sigma_u)$, and Gibbs steps for σ_u^2.

First, we note that we can simplify the fraction in the formula of the acceptance probability in (5):

$$\begin{aligned} \frac{p(\mathbf{y}_t|\mathbf{x}_t')}{p(\mathbf{y}_t|\mathbf{x}_t)} &= \frac{\binom{n_t}{y_t}\pi(x_t')^{y_t}(1-\pi(x_t'))^{n_t-y_t}}{\binom{n_t}{y_t}\pi(x_t)^{y_t}(1-\pi(x_t))^{n_t-y_t}} \\ &= \frac{\exp\{y_t x_t' - n_t \ln(1+e^{x_t'})\}}{\exp\{y_t x_t - n_t \ln(1+e^{x_t}\}} \\ &= \exp\left\{y_t(x_t' - x_t) + n_t \ln\left(\frac{1+e^{x_t}}{1+e^{x_t'}}\right)\right\}. \end{aligned}$$

We can also note that:

$$x_t|\mathbf{x}_{-t}, \sigma_u^2 \sim \begin{cases} \mathcal{N}(x_2, \sigma_u^2), & t = 1, \\ \mathcal{N}(\frac{1}{2}(x_{t+1} + x_{t-1}), \frac{\sigma_u^2}{2}), & t \in \{2, ..., 365\} \\ \mathcal{N}(x_{365}, \sigma_u^2), & t = 366 \end{cases}$$

Proof:
Starting with the conditional probability distribution

$$\begin{aligned} p(x_t|x_{-t}, \sigma_u^2) = \frac{p(x_t, x_{-t}, \sigma_u^2)}{p(x_{-t}, \sigma_u^2)} &\propto p(\mathbf{x}|\sigma_u^2) \\ &= \prod_{t=2}^{T} \frac{1}{\sigma_u} e^{-\frac{1}{2\sigma_u^2}(x_t - x_{t-1})^2}. \end{aligned}$$

For $t = 2, .., 365$ we have:

$$
\begin{aligned}
p(x_t|x_{-t},\sigma_u^2) &\propto e^{-\frac{1}{2\sigma_u^2}(x_t-x_{t-1})^2} e^{-\frac{1}{2\sigma_u^2}(x_{t+1}-x_t)^2} \\
&= e^{-\frac{1}{2\sigma_u^2}(x_t^2+x_{t-1}^2-2x_tx_{t-1}+x_{t+1}^2+x_t^2-2x_tx_{t+1})} \\
&= e^{-\frac{1}{2\sigma_u^2}(2x_t^2-2x_t(x_{t-1}+x_{t+1})+x_{t+1}^2+x_{t-1}^2)} \\
&\propto e^{-\frac{2}{2\sigma_u^2}(x_t-\frac{1}{2}(x_{t-1}+x_{t+1}))^2}.
\end{aligned}
$$

Thus,

$$
x_t|x_{-t},\sigma_u^2 \sim \mathcal{N}\left(\frac{1}{2}(x_{t-1}+x_{t+1}),\frac{\sigma_u^2}{2}\right).
$$

For $t = 1$ we have

$$
p(x_1|x_{-1},\sigma_u^2) \propto e^{-\frac{1}{2\sigma_u^2}(x_2-x_1)^2} = e^{-\frac{1}{2\sigma_u^2}(x_1-x_2)^2}.
$$

Thus,

$$
x_1|x_{-1},\sigma_u^2 \sim \mathcal{N}\left(x_2,\sigma_u^2\right).
$$

For $t = 366$ we have

$$
p(x_{366}|x_{-366},\sigma_u^2) \propto e^{-\frac{1}{2\sigma_u^2}(x_{366}-x_{365})^2}.
$$

Thus,

$$
x_{366}|x_{-366},\sigma_u^2 \sim \mathcal{N}\left(x_{365},\sigma_u^2\right).
$$

We will use these results in the implementation.
We will calculate also the computation time and acceptance rate.

```
mcmc_sampler1 <- function(alpha, beta, k, Tot, y, n){

  burnin=100
  k <- k+burnin
  accept=0 # variable that counts how many proposed values are accepted

  x_matrix = matrix(nrow=k, ncol=Tot) # matrix to store results
  x_matrix[1,] <- (y+1)/(n+2) # initialization of the first row

  sigma2_vector = c() # vector to store values of sigma squared
  sigma2_vector[1] = 1/rgamma(1,alpha, scale = beta) # first value

  start_time <- proc.time()[3] # Start measuring computation time

  alpha_new <- alpha + (Tot-1)/2

  for (j in 2:k){ # for every row

    x = x_matrix[j-1, ] # (j-1)th row of the matrix
    new_x = c() # j-th row of the matrix

    beta_new <- beta + 0.5*sum((x[2:Tot] - x[1:(Tot-1)])^2)
    sigma2_vector[j] <- 1/rgamma(1, alpha_new, scale=beta_new)

    # MH steps
    for (t in 1:Tot) {

      if (t == 1){
        proposed_x <- rnorm(1, x[2], sqrt(sigma2_vector[j]))
      } else if (t == 366) {
        proposed_x <- rnorm(1, x[365], sqrt(sigma2_vector[j]))
      } else {
        proposed_x <- rnorm(1, (x[t+1]+x[t-1])/2, sqrt(sigma2_vector[j])/2)
      }

      u <- runif(1)
      alpha <- min(1,exp(y[t]*(proposed_x-x[t]) +
                           n[t]*log((1+exp(x[t]))/(1+exp(proposed_x)))))

      if(u<alpha){
        new_x[t] <- proposed_x
        accept <- accept + 1
      }
      else {
        new_x[t] <- x[t]
      }
    }

     x_matrix[j, ] <- new_x
  }

  end_time <- proc.time()[3] # End measuring computation time
  computation_time <- end_time - start_time

  acceptance_prob <- accept/(Tot*k)*100

  cat("Acceptance rate:", round(acceptance_prob,2), "%\n")
  cat("Running time:", computation_time, "seconds \n")
  return (list(
    matrix = x_matrix[(burnin+1):k,],
    sigma2 = sigma2_vector[(burnin+1):k]
  ))

}
```

We now implement the function setting $\alpha = 2$, $\beta = 0.05$ and $k = 50000$ (number of iterations). The results of the function include one matrix containing the values of x_t and a vector containing the values of σ^2.

```
set.seed(123)
alpha=2
beta=0.05
k=50000
Tot=366
n=rain$n.years
y=rain$n.rain

results1<- mcmc_sampler1(alpha,beta,k,Tot,y,n)
```

```
## Acceptance rate: 94.21 %
## Running time: 98.03 seconds
```

We consider the quantities σ_u^2, $\pi(x_1)$, $\pi(x_{201})$ and $\pi(x_{366})$. To compute the last three, we need a proper function.

```r
pi <- function(x) {1/(1+exp(-x))}
x_matrix <- results1$matrix
pi_matrix <- pi(x_matrix)
pi1 <-pi_matrix[,1]
pi2 <- pi_matrix[,201]
pi3 <- pi_matrix[,366]
sigma2 <- results1$sigma2
```

We proceed by making traceplots, histograms and estimated autocorrelation functions of these quantities.

```r
par(mfrow=c(2,2), mar = c(5,5,0,0))
plot(pi1, type="l", xlab="", ylab=expression(pi[1]), col="darkgreen")
plot(pi2, type="l", xlab="", ylab=expression(pi[201]), col="darkgreen")
plot(pi3, type="l", xlab="",ylab=expression(pi[366]), col="darkgreen")
plot(sigma2, xlab="", ylab=expression(sigma[u]^2),type="l", col="darkgreen")
```

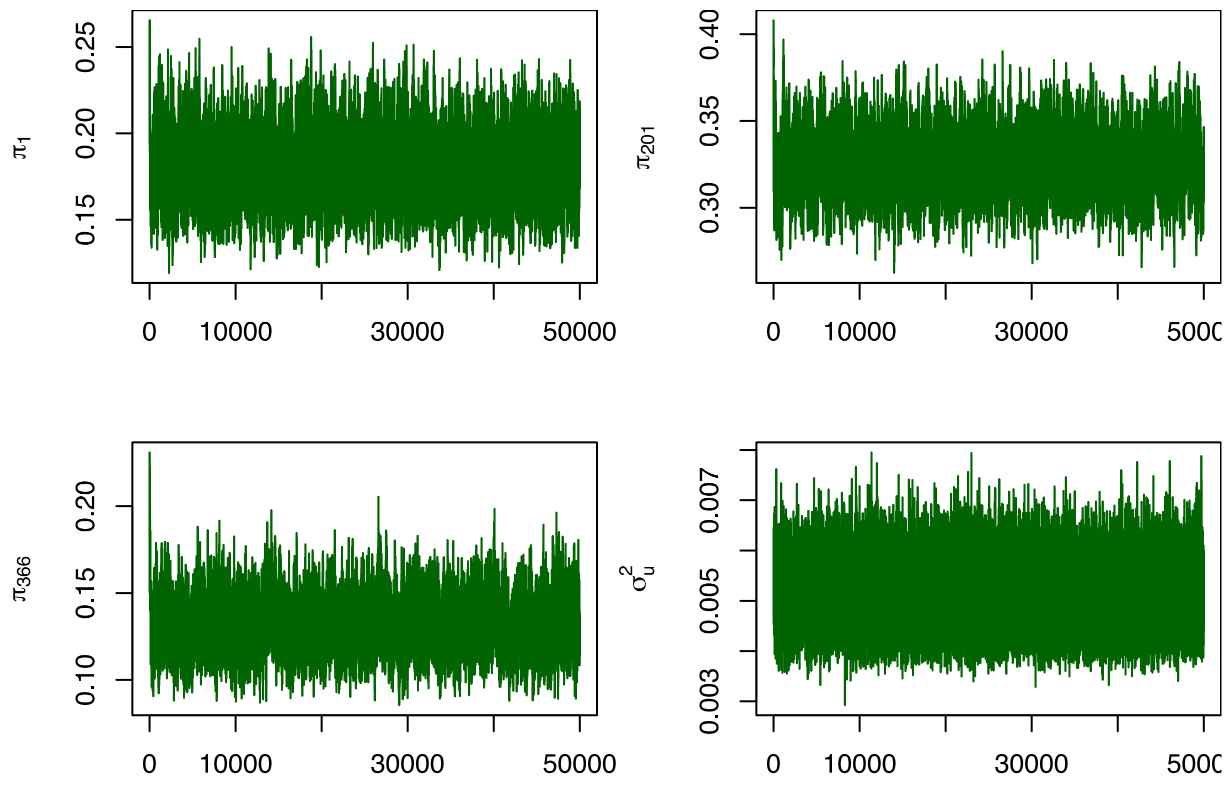


Figure 1: Traceplots of $\pi(x_1)$, $\pi(x_{201})$, $\pi(x_{366})$ and σ_u^2, produced by MCMC algorithm for 50000 iterations, where every element of $\mathbf{x}$ is updated per iteration.

These trace plots show how the values of these variables change over time during the execution of the algorithm.
The Markov chain seems to converge and since it does not take a long time to move around the parameter space, it looks to converge quickly.
However, this diagnostic does not guarantee convergence and there is no overall minimum number of samples to ensure approximation of the target distribution.

```r
par(mfrow=c(2,2), mar=c(5,5,0,0))
hist(pi1, xlab=expression(pi(x[1])), col="orange", main="", breaks=100)
abline(v=mean(pi1), col="blue", lwd=2)
hist(pi2, xlab=expression(pi(x[201])), col="orange", main="", breaks=100)
abline(v=mean(pi2), col="blue", lwd=2)
hist(pi3, xlab=expression(pi(x[366])), col="orange", main="", breaks=100)
abline(v=mean(pi3), col="blue", lwd=2)
hist(sigma2,xlab=expression(sigma^2), col="orange", main ="", breaks=100)
abline(v=mean(sigma2), col="blue", lwd=2)
```

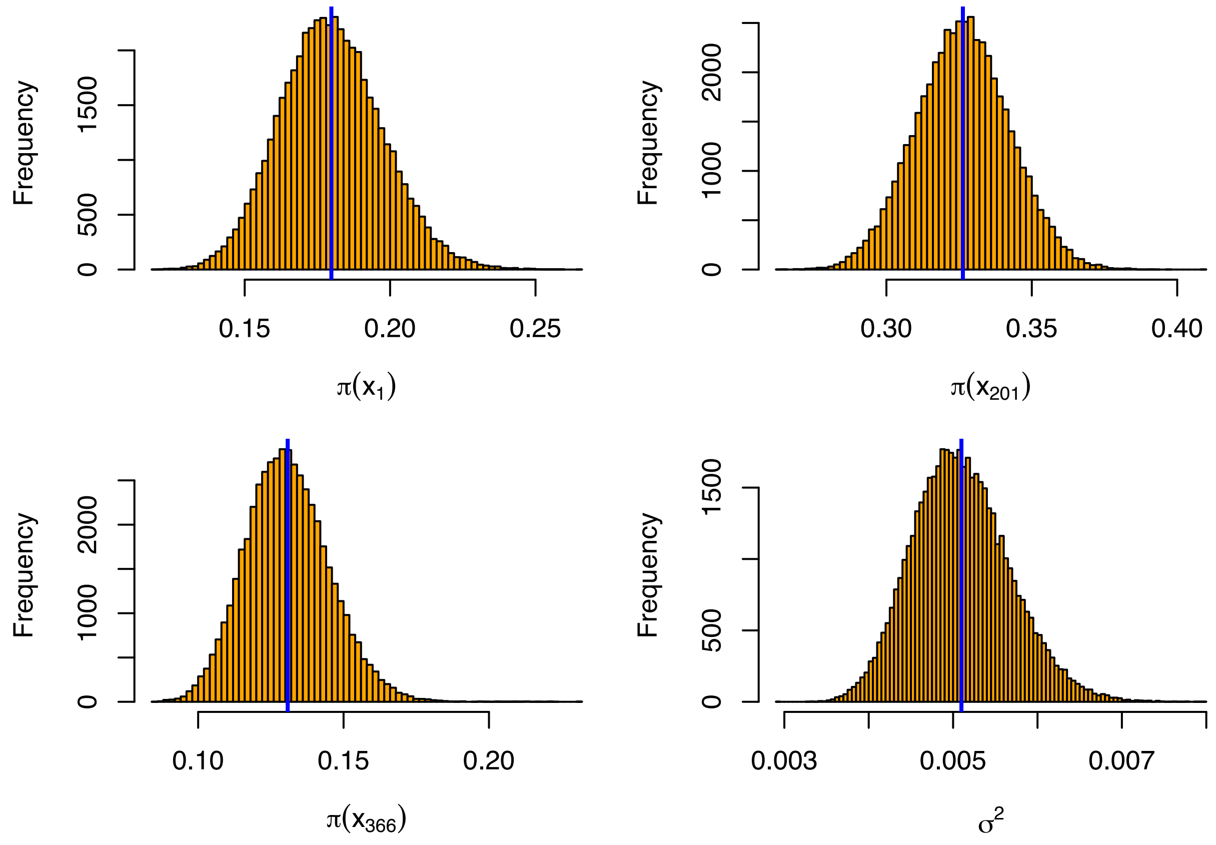


Figure 2: Histograms of $\pi(x_1)$, $\pi(x_{201})$, $\pi(x_{366})$ and σ_u^2, with corresponding means, produced by MCMC algorithm for 50000 iterations, where every element of $\mathbf{x}$ is updated per iteration.

These histograms provide a view of the distribution of sampled values for these variables.

```r
par(mfrow=c(2,2), mar=c(5,5,0,0))
acf(pi1, main="",ylab = expression(paste("ACF ", pi[1])))
acf(pi2, main="", ylab = expression(paste("ACF ", pi[201])))
acf(pi3, main="", ylab = expression(paste("ACF ", pi[366])))
acf(sigma2, main="", ylab = expression(paste("ACF ", sigma^2)))
```

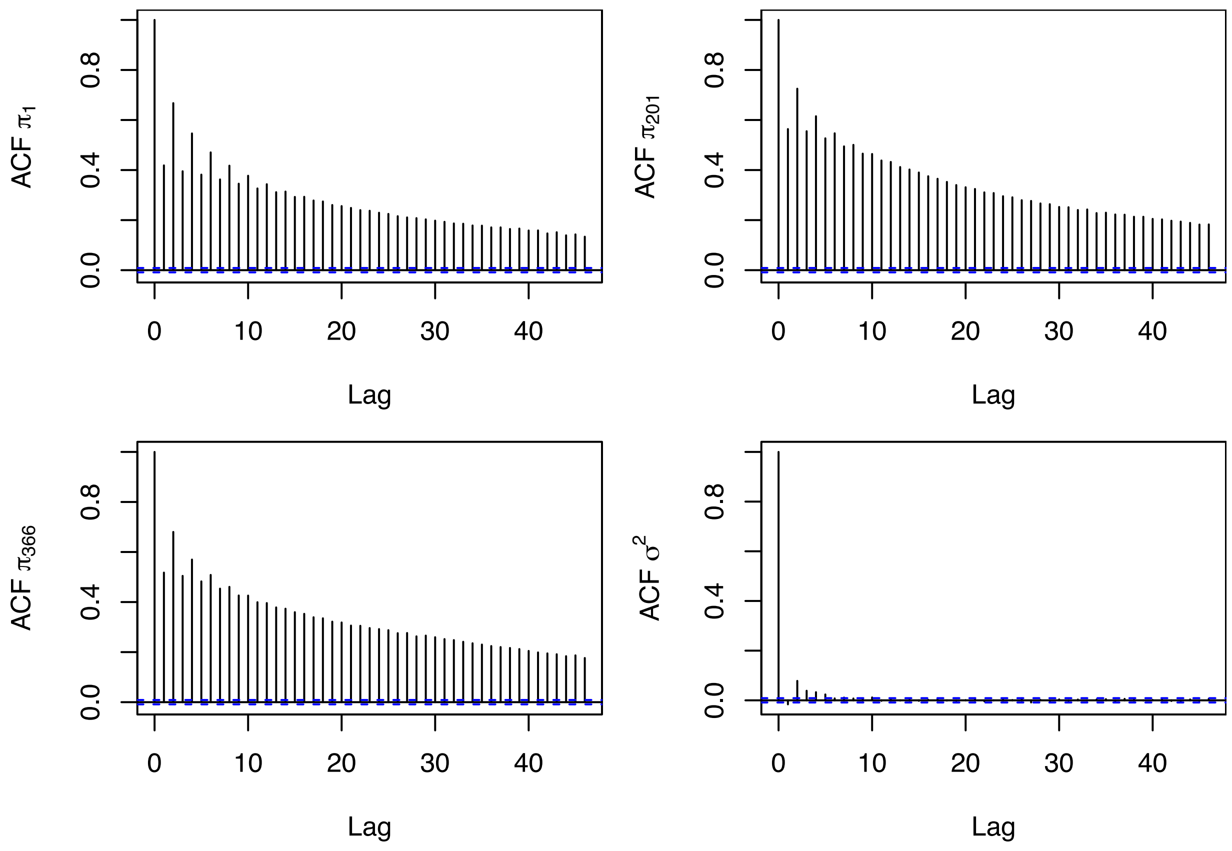


Figure 3: Autocorrelation functions of $\pi(x_1)$, $\pi(x_{201})$, $\pi(x_{366})$ and σ_u^2, produced by a MCMC algorithm for 50000 iterations, where every element of $\mathbf{x}$ is updated per iteration.

The plots shows that there is less autocorrelation between samples for σ_u^2, compared to the others. We could expect this result, because of the nature of a random walk.

Lastly we compare predictions for π_t and associated uncertainties to y_t/n_t as a function of t.

```
mean <-colMeans(pi_matrix)
library(coda)
pi_mcmc <- as.mcmc(pi_matrix)
conf_levels = summary(pi_mcmc)$quantile
lower <- conf_levels[,1]
upper <- conf_levels[,5]
```

```
set.seed(123)
plot(rain$day, y/n, type="l", col="darkgray", xlab="t", ylab="")
lines(rain$day, mean, col="red")
lines(lower, col="blue", lty=2)
lines(upper, col="blue", lty=2)
legend("topleft",
       legend = c(expression(y[t]/n[t]),
                  expression(paste(pi[t]," predictions")),
                  expression(paste("95% credible intervals for ", pi[t]))),
       col = c("darkgray", "red", "blue"),
       lty = c(1, 1,2), cex=0.6)
```

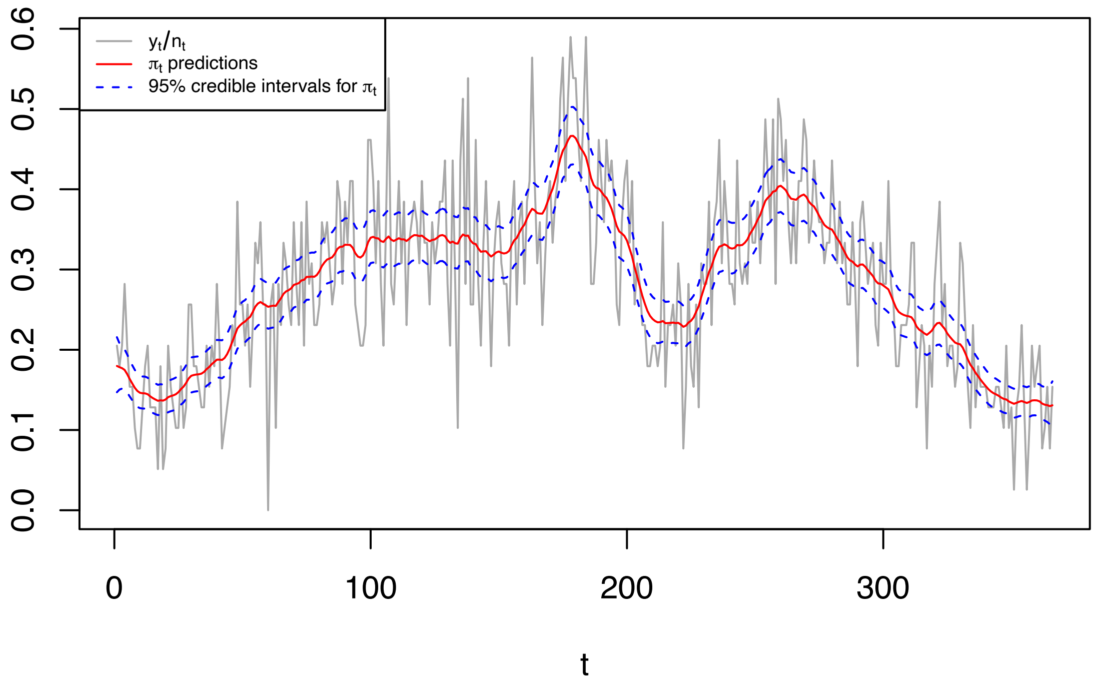


Figure 4: Central estimates and 95% credible intervals for $\pi(x_t)$ as a function of t compared to y_t/n_t.

g)

We will repeat part f), except now rather than updating individual x_t parameters, we will use a conditional prior proposal involving $p(\mathbf{x}_{(a,b)}|\mathbf{x}_{-(a,b)}, \sigma_u^2)$, where $\mathbf{x}_{(a,b)} = (x_a ... x_b)^T$ choosing intervals $[a, b]$ of length M. Thus, in every iteration, we will update all the elements in a block. We will need to choose an appropriate value for M.
The rationale behind this approach is to potentially expedite the process, although it's essential to strike a balance as overly large blocks could lead to computationally intensive matrix operations.

First, we derive a new formula for the acceptance probability, suitable for this case where we work in blocks.

$$\begin{aligned}\alpha(\mathbf{x}'_{\mathcal{I}}|\mathbf{x}_{-\mathcal{I}}, \sigma_u^2, \mathbf{y}) &= \min\left(1, \frac{p(y_{\mathcal{I}}|x'_{\mathcal{I}})}{p(y_{\mathcal{I}}|x_{\mathcal{I}})}\right) \\ &= \min\left(1, \sum_{\mathcal{I}}\left[y_{\mathcal{I}}(x'_{\mathcal{I}} - x_{\mathcal{I}}) + n_{\mathcal{I}} \ln\left(\frac{1+e^{x_{\mathcal{I}}}}{1+e^{x'_{\mathcal{I}}}}\right)\right]\right),\end{aligned}$$

where $\mathcal{I} = (a, b)$.

We know that the conditional distribution of $x_{\mathcal{I}}|x_{-\mathcal{I}}$ follows a multivariate normal distribution, as shown in part e), where $\mu_{\mathcal{I}|-\mathcal{I}} = -Q_{\mathcal{I},\mathcal{I}}^{-1} Q_{\mathcal{I},-\mathcal{I}} x_{-I}$, is the conditional mean and $Q_{\mathcal{I}|-I} = Q_{\mathcal{I},\mathcal{I}}$ is the precision matrix.

To improve the efficiency of our sampling process, we can pre-compute $Q_{\mathcal{I},\mathcal{I}}^{-1} Q_{\mathcal{I},-\mathcal{I}}$ and use the Cholesky decomposition of $Q_{\mathcal{I},\mathcal{I}}^{-1}$ to sample from a multivariate normal.
We will perform these pre-computations three times:
1) First, for $a = 1$ and $b = M$;
2) Second, for $a > 1$ and $b < 366$;
3) Lastly, for $b = 366$.

In the following code, we first write some helpful functions, later we implement the sampler.

```r
# Function to generate Q, tri-diagonal matrix as described in part e)
tridiag <- function(T){
  Q <- matrix(0, nrow = T, ncol = T)
  diag(Q) <- 2
  for (i in 1:(T - 1)) {
    Q[i, i + 1] <- -1
    }
  for (i in 2:T) {
    Q[i, i - 1] <- -1
    }
  Q[1, 1] <- 1
  Q[T, T] <- 1
return(Q)
}

# Function that computes the acceptance probability for this case
acc_prob <- function(int, proposal, x_int) {
  prob = exp(sum(y[int]*(proposal-x_int)+
                   n[int]*log((1+exp(x_int))/(1+exp(proposal)))))
  return(min(1, prob))
}



# Metropolis-Hastings step
MH_step <- function(int, old_x, sigma2, Q_prod, Q_chol) {
  # conditional mean
  cond_mu = Q_prod %*% old_x[-int]
  # sample a proposal from multivariate normal
  proposal = cond_mu + sqrt(sigma2) * Q_chol %*% rnorm(length(old_x[int]))
  # accept/reject
  if (runif(1) < acc_prob(int, proposal, old_x[int])) {
    return(list(x = proposal, accept = length(int)))
    } else {
      return(list(x = old_x[int], accept = 0))
    }
  }
```

```r
mcmc_sampler2 <- function(alpha, beta, k, M) {

  start_time = proc.time()[3] # Start measuring computation time
  Tot      = 366
  accept   = 0
  iter     = ceiling(Tot/M)

  ## Pre-calculate the matrices needed
  Q = tridiag(Tot)
  # sub-matrices of Q
  Q_AA1 = Q[1:M, 1:M]
  Q_AA2 = Q[2:(M + 1), 2:(M + 1)]
  Q_AA3 = Q[(Tot + 1 - M):Tot, (Tot + 1 - M):Tot]
  # inverses
  Q_AA1_inv = solve(Q_AA1)
  Q_AA2_inv = solve(Q_AA2)
  Q_AA3_inv = solve(Q_AA3)
  # matrices needed to compute the product to obtain the conditional mean
  Q_AB1 = Q[1:M, -(1:M)]
  Q_AB3 = Q[(Tot + 1 - M):Tot, -((Tot + 1 - M):Tot)]
  # cholesky decomposition
  Q_chol1 = t(chol(Q_AA1_inv))
  Q_chol2 = t(chol(Q_AA2_inv))
  Q_chol3 = t(chol(Q_AA3_inv))
  # product needed for computation of the conditional mean
  Q_prod1 = -Q_AA1_inv %*% Q_AB1
  Q_prod3 = -Q_AA3_inv %*% Q_AB3

  # still need to compute Q_AB2, Q_prod2, will be computed inside the loops

  # Initialize variables to store results
  x_matrix = matrix(0, nrow = k, ncol = Tot)
  x_matrix[1, ] = (y+1)/(n+2) # initialize first row
  sigma2 = c()
  sigma2[1] = 1/rgamma(1, 2, rate = 0.05) # first value, inverse gamma prior
  alpha_new = alpha+(Tot-1)/2

  # Iterate for all k samples
  for (i in 2:k) {

    # Gibb's sampling for sigma
    sumdif <- sum(diff((x_matrix[i, ]))^2)
    sigma2[i] <- 1/rgamma(1, alpha_new, scale=(beta + 0.5*sumdif))

    # Block 1
    a = 1
    b = M
    int = c(a:b)
    block1 = MH_step(int, x_matrix[i - 1, ], sigma2[i - 1], Q_prod1, Q_chol1)
    x_matrix[i, int] = block1$x
    accept = accept + block1$accept

    for (j in 2:(iter - 1)) {
      # Block 2
      a = (j - 1) * M + 1
      b = j * M
      int = c(a:b)
      Q_AB2 = Q[int, -int]
      Q_prod2 = -Q_AA2_inv %*% Q_AB2
      block2 = MH_step(int, x_matrix[i - 1, ], sigma2[i - 1], Q_prod2, Q_chol2)
      x_matrix[i, int] = block2$x
      accept = accept + block2$accept
      }

    # Block 3
    a = (Tot - M) + 1
    b = Tot
    int = c(a:b)
    block3 = MH_step(int, x_matrix[i - 1, ], sigma2[i - 1], Q_prod3, Q_chol3)
    x_matrix[i, int] = block3$x
    accept = accept + block3$accept

    }

  end_time <- proc.time()[3] # End measuring computation time
  computation_time <- end_time - start_time

  return(list(pi = pi(x_matrix), # matrix of sampled pi(x)
              x = x_matrix, # matrix of sampled x
              sigma2 = sigma2, # vector of sampled sigma^2
              acceptance_rate = accept/(Tot *k),
              computation_time=computation_time))
}
```

To choose the tuning parameter M, we try different values. For each of these values we will compute the computation time and the acceptance rate, and we will plot the results. In this part, we implement the sampler 3000 times.

```
set.seed(123)
alpha=2
beta=0.05
k=3000
M_grid = c(1, 3, 5, 8, 10, 15, 20, 30, 50, 75, 100)
comp_time = c()
accept_rate = c()
for(i in 1:length(M_grid)){
  results <- mcmc_sampler2(alpha, beta, k,M_grid[i])
  comp_time[i]=results$computation_time
  accept_rate[i]=results$acceptance_rate
  }
```

We add the minimum and maximum values in the plots.

```
par(mfrow=c(2,1), mar=c(4,3,2,2))
plot(M_grid, comp_time, type="b", xlab="M", ylab="",main="Computation time")
points(M_grid[which.min(comp_time)], comp_time[which.min(comp_time)],
       col="green", cex=1.5, pch=20)# min
points(M_grid[which.max(comp_time)], comp_time[which.max(comp_time)],
       col="red", cex=1.5, pch=20) #max
plot(M_grid, accept_rate, type="b", xlab="M", ylab="",main="Acceptance rate")
points(M_grid[which.max(accept_rate)],accept_rate[which.max(accept_rate)],
       col="green", cex=1.5, pch=20) # max
points(M_grid[which.min(accept_rate)],accept_rate[which.min(accept_rate)],
       col="red", cex=1.5, pch=20) # max
```

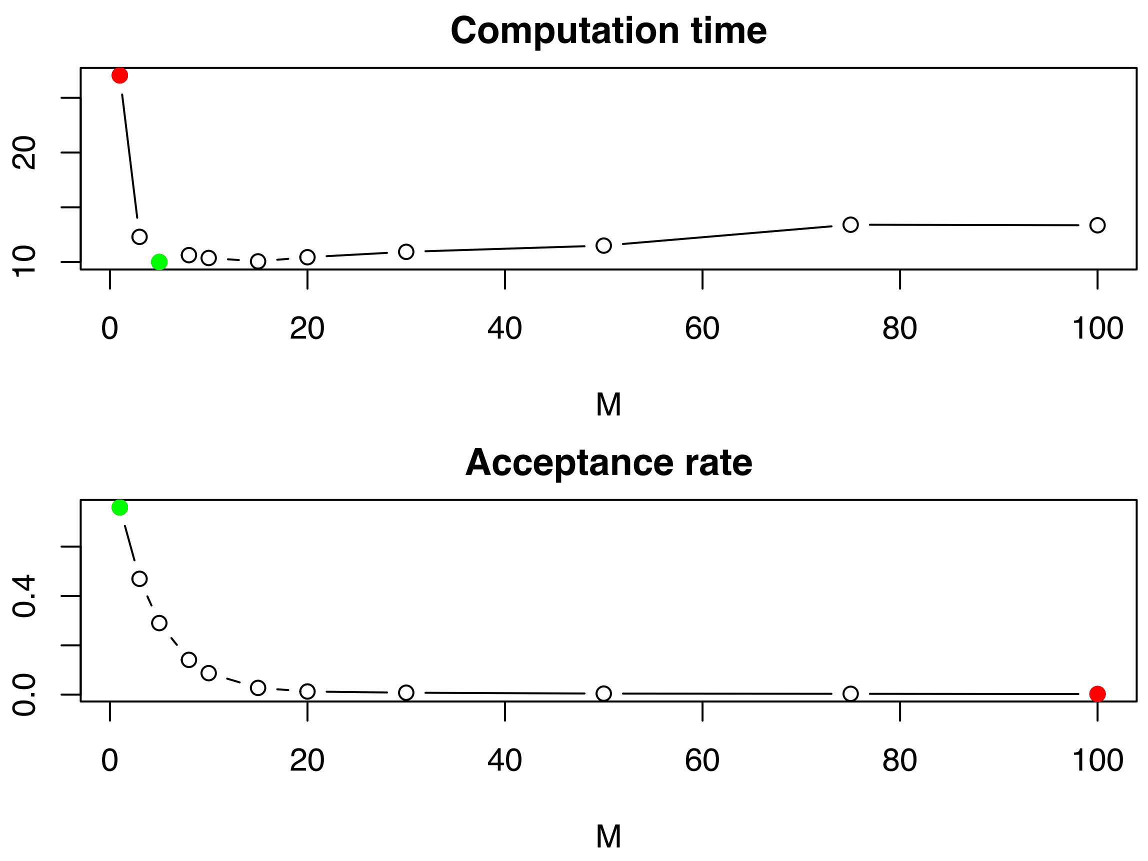


Figure 5: Computation time and acceptance rate for different values of M.

From the left graph, it is evident that the computation time peaks when $M = 1$, and it has a rapid decrease till $M = 3$. After that it doesn't decrease gradually as M increases; instead, it fluctuates a bit.
On the right graph, we observe that the acceptance rate peaks at $M = 1$ and it decreases thereafter.
It seems reasonable to select a value of M that strikes a balance between these two factors. A good choice seems $M = 5$ because the computation time is low and the acceptance rate is quite high.
We will now implement the sampler with 50000 samples and $M = 5$. We will remake the same graphs as in part f).

```
M=5
k=50000
n=rain$n.years
y=rain$n.rain
results2 <- mcmc_sampler2(alpha, beta, k,M)
results2$acceptance_rate
```

```
## [1] 0.2882626
```

```
results2$computation_time
```

```
## elapsed
##  170.12
```

Incorporating a block step over the $\mathbf{x}_{(a,b)}$ parameters may improve the efficiency of the MCMC sampler because the parameters are highly correlated. However, if the block has high dimension, the acceptance rate can be very small and so the computation time might be very high.

```
sigma2_2 <- results2$sigma2
pi1_2 <- results2$pi[,1]
pi2_2 <- results2$pi[,201]
pi3_2 <- results2$pi[,366]
```

```
par(mfrow=c(2,2), mar = c(5,5,0,0))
plot(pi1_2, type="l", xlab="", ylab=expression(pi[1]), col="darkgreen")
plot(pi2_2, type="l", xlab="", ylab=expression(pi[201]), col="darkgreen")
plot(pi3_2, type="l", xlab="",ylab=expression(pi[366]), col="darkgreen")
plot(sigma2_2, xlab="", ylab=expression(sigma^2),type="l", col="darkgreen")
```

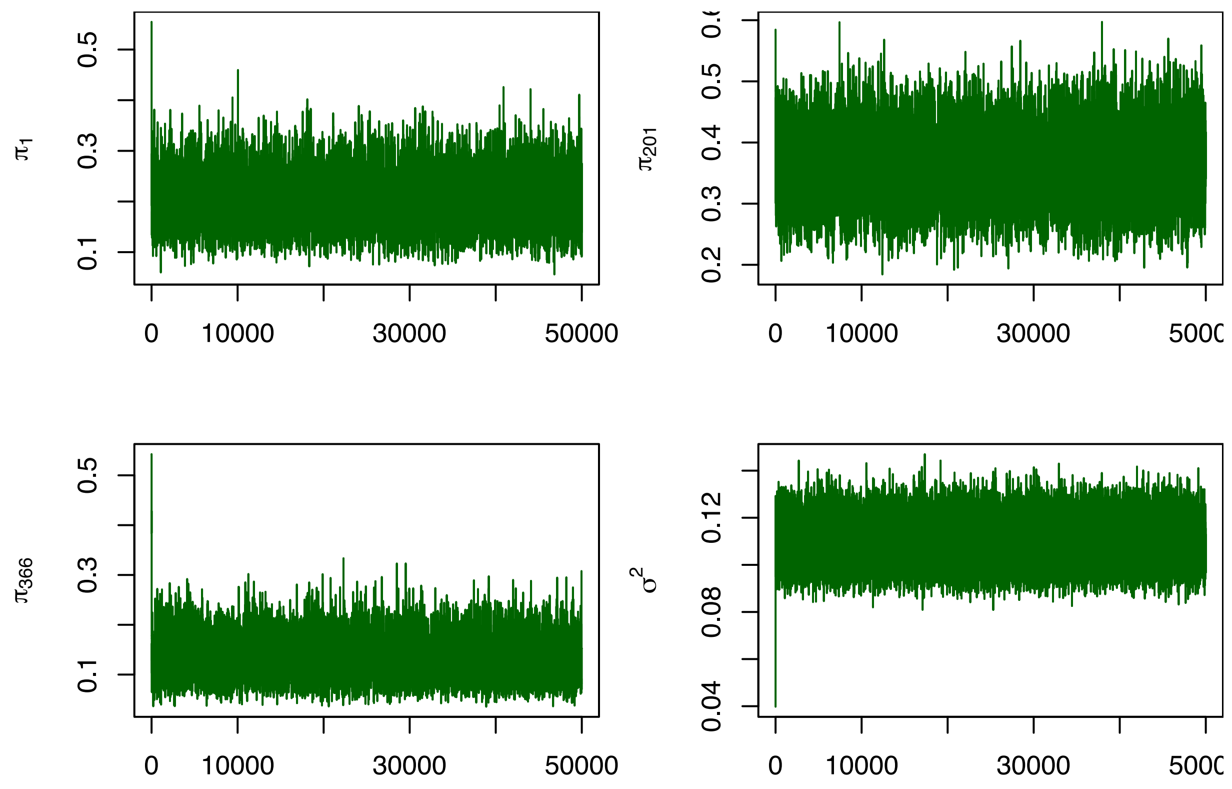


Figure 6: Traceplots of $\pi(x_1)$, $\pi(x_{201})$, $\pi(x_{366})$ and σ_u^2, produced by a MCMC algorithm for 50000 iterations, with block updating.

```
par(mfrow=c(2,2), mar=c(5,5,0,0))
hist(pi1_2, xlab=expression(pi(x[1])), col="orange", main="", breaks=100)
abline(v=mean(pi1_2), col="blue", lwd=2)
hist(pi2_2, xlab=expression(pi(x[201])), col="orange", main="", breaks=100)
abline(v=mean(pi2_2), col="blue", lwd=2)
hist(pi3_2, xlab=expression(pi(x[366])), col="orange", main="", breaks=100)
abline(v=mean(pi3_2), col="blue", lwd=2)
hist(sigma2_2,xlab=expression(sigma^2), col="orange", main ="", breaks=100)
abline(v=mean(sigma2_2), col="blue", lwd=2)
```

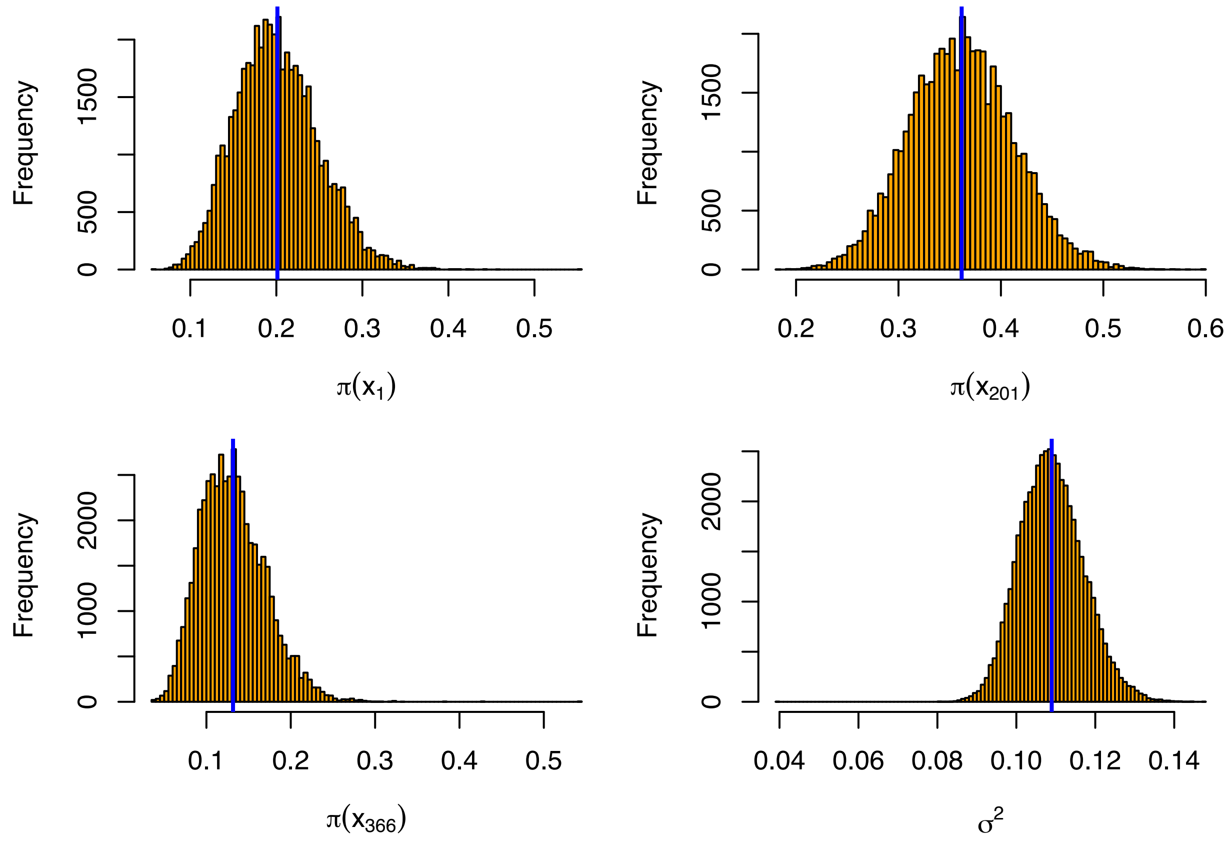


Figure 7: Histograms of $\pi(x_1)$, $\pi(x_{201})$, $\pi(x_{366})$ and σ_u^2, with corresponding means, produced by a MCMC algorithm for 50000 iterations, with block updating.

```
par(mfrow=c(2,2), mar=c(5,5,0,0))
acf(pi1_2, main="",ylab = expression(paste("ACF ", pi[1])))
acf(pi2_2, main="", ylab = expression(paste("ACF ", pi[201])))
acf(pi3_2, main="", ylab = expression(paste("ACF ", pi[366])))
acf(sigma2_2, main="", ylab = expression(paste("ACF ", sigma^2)))
```

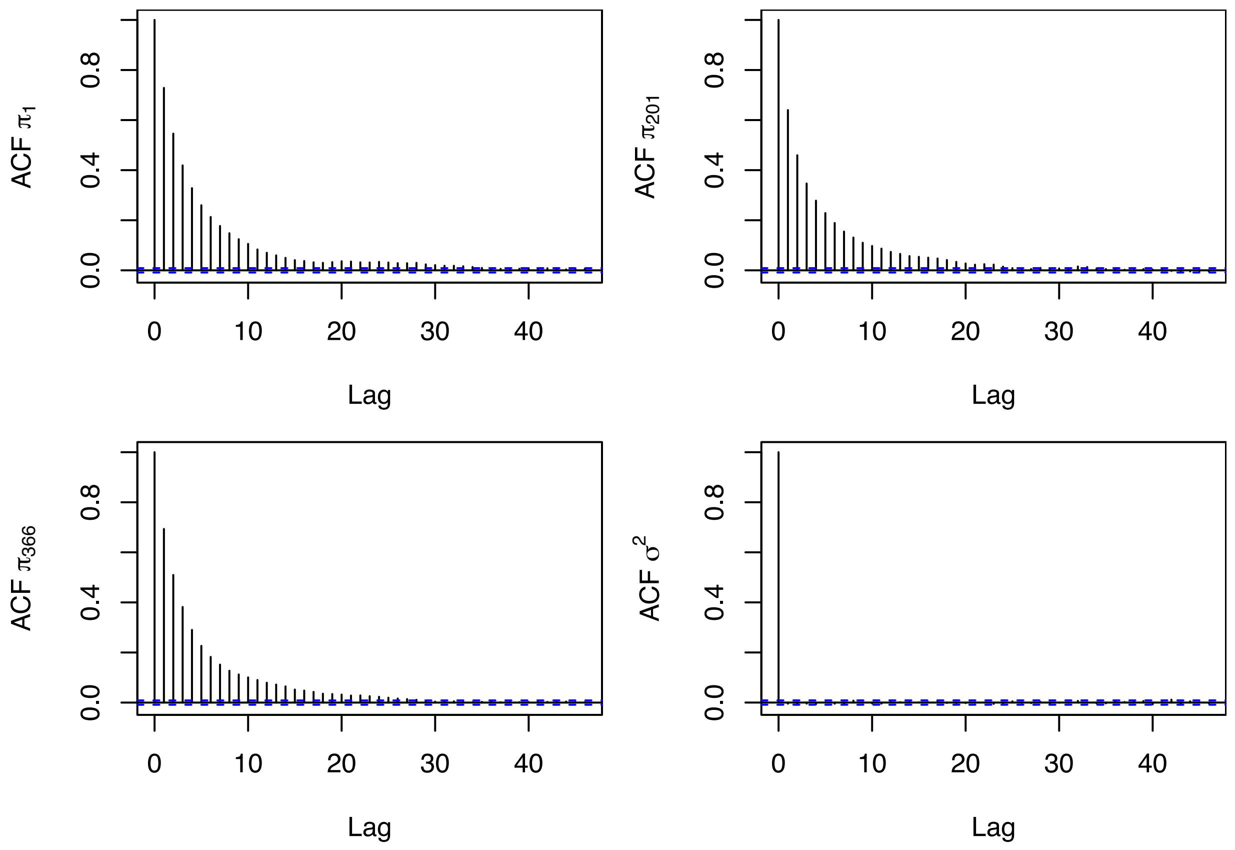


Figure 8: Autocorrelation functions of $\pi(x_1)$, $\pi(x_{201})$, $\pi(x_{366})$ and σ_u^2, produced by a MCMC algorithm for 50000 iterations, with block updating.

```
pi <- results2$pi
mean2 <-colMeans(pi)
pi_mcmc2 <- as.mcmc(pi)
conf_levels2 = summary(pi_mcmc2)$quantile
lower2 <- conf_levels2[,1]
upper2 <- conf_levels2[,5]
```

```
set.seed(123)
plot(rain$day, y/n, type="l", col="darkgray", xlab="t", ylab="")
lines(rain$day, mean2, col="red")
lines(lower2, col="blue", lty=2)
lines(upper2, col="blue", lty=2)
legend("topleft",
       legend = c(expression(y[t]/n[t]),
                  expression(paste(pi[t]," predictions")),
                  expression(paste("95% credible intervals for ", pi[t]))),
       col = c("darkgray", "red", "blue"),
       lty = c(1, 1,2), cex=0.6)
```

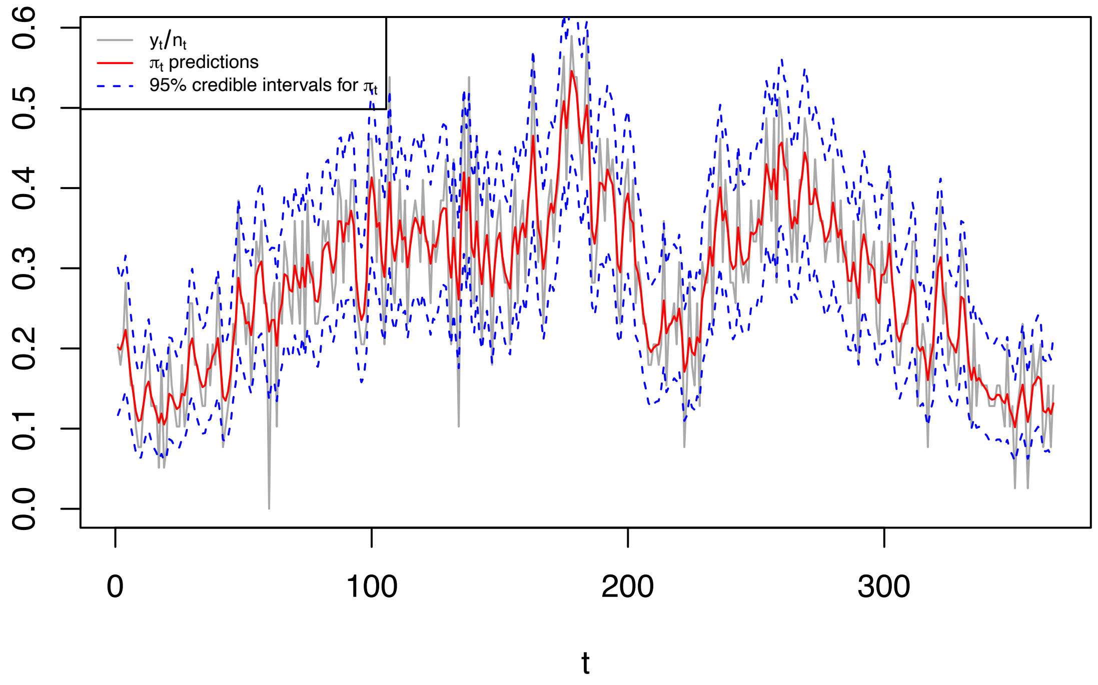


Figure 9: Central estimates and 95% credible intervals for $\pi(x_t)$ as a function of t compared to y_t/n_t.

Here we compare the resulting values from part f) and part g).

```
pi1f <- c(mean[1], as.numeric(lower[1]), as.numeric(upper[1]))
pi2f <- c(mean[201], as.numeric(lower[201]), as.numeric(upper[201]))
pi3f <- c(mean[366], as.numeric(lower[366]), as.numeric(upper[366]))
```

```
pi1g <- c(mean2[1], as.numeric(lower2[1]), as.numeric(upper2[1]))
pi2g <- c(mean2[201], as.numeric(lower2[201]), as.numeric(upper2[201]))
pi3g <- c(mean2[366], as.numeric(lower2[366]), as.numeric(upper2[366]))

rownames <- c("pi_1_f", "pi_1_g", "pi_201_f", "pi_201_g", "pi_366_f", "pi_366_g")
df <- data.frame(
  pi = c(pi1f[1], pi1g[1], pi2f[1], pi2g[1], pi3f[1], pi3g[1]),
  lower = c(pi1f[2], pi1g[2], pi2f[2], pi2g[2], pi3f[2], pi3g[2]),
  upper = c(pi1f[3], pi1g[3], pi2f[3], pi2g[3], pi3f[3], pi3g[3]),
  row.names = rownames
)

print(df)
```

```
##                 pi      lower     upper
## pi_1_f   0.1798080 0.14706311 0.2156600
## pi_1_g   0.2012529 0.11686023 0.3007033
## pi_201_f 0.3263023 0.29520957 0.3578486
## pi_201_g 0.3616735 0.26598918 0.4639962
## pi_366_f 0.1307847 0.10459002 0.1606415
## pi_366_g 0.1315474 0.06740959 0.2163820
```

PROBLEM 2

We want to compare the model INLA with the previous Markov chain, so we need to use the same priors as before. In this model the intercept term is removed so we don't include a prior on this term. As INLA places a prior on the log precision rather than the variance, the following equation holds:

$$\theta = \log\left(\frac{1}{\sigma_u^2}\right) \underset{\sigma_u^2 \sim \Gamma^{-1}(\alpha,\beta)}{\Longrightarrow} \theta \sim \log\Gamma(\alpha, \beta).$$

We use $(\alpha, \beta) = (2, 0.05)$.

```
params = c(2, 0.05)
hypers = list(prec = list(prior = "loggamma", param = params))
```

a)

We want to compare the predictions and uncertainties of INLA with those founded previously with Markov chains.

```
library("INLA")
```

```
## Caricamento del pacchetto richiesto: Matrix
```

```
## Caricamento del pacchetto richiesto: sp
```

```
## This is INLA_24.02.09 built 2024-02-09 03:35:28 UTC.
##  - See www.r-inla.org/contact-us for how to get help.
##  - List available models/likelihoods/etc with inla.list.models()
##  - Use inla.doc(<NAME>) to access documentation
```

```
control.inla = list(strategy="simplified.laplace", int.strategy="ccd")
```

We can fit the same model as in the previous problem using the function INLA and compute the computation time:

```r
t1=proc.time()[3]
mod <- inla(n.rain ~ -1 + f(day, model="rw1", constr=FALSE, hyper=hypers),
            data=rain, Ntrials=n.years, control.compute=list(config = TRUE),
            family="binomial", verbose=FALSE, control.inla=control.inla)
t2=proc.time()[3]

print(paste("The computation time (elapsed) is ", t2-t1))
```

```
## [1] "The computation time (elapsed) is  1.07999999999998"
```

The computation time is much shorter than it was for MCMC because this implementation is easier in *R*.

We can see the predictions and the 95% credible intervals of the hyper parameter and the latent variables:

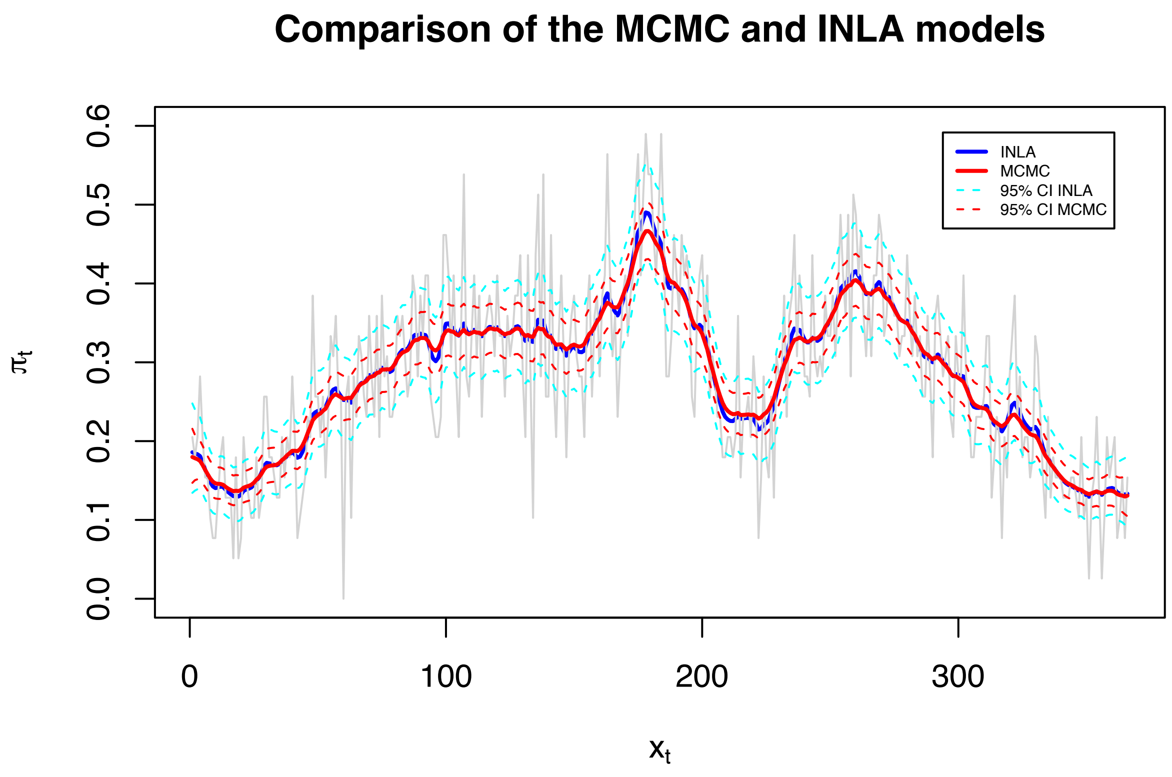
```r
mod$summary.hyperpar
```

```
##                         mean      sd 0.025quant 0.5quant 0.975quant     mode
## Precision for day 106.7317 27.5681   62.13927 103.5107   169.9024 97.51874
```

```r
head(mod$summary.fitted.values)
```

```
##                             mean         sd 0.025quant  0.5quant 0.975quant
## fitted.Predictor.001 0.1858658 0.02899140  0.1343683 0.1840330  0.2477832
## fitted.Predictor.002 0.1844115 0.02636449  0.1371768 0.1828809  0.2403464
## fitted.Predictor.003 0.1833016 0.02466280  0.1388967 0.1819431  0.2354305
## fitted.Predictor.004 0.1809874 0.02344288  0.1386759 0.1797295  0.2304526
## fitted.Predictor.005 0.1731196 0.02200604  0.1333252 0.1719636  0.2194920
## fitted.Predictor.006 0.1638340 0.02072744  0.1262872 0.1627660  0.2074601
##                             mode
## fitted.Predictor.001 0.1803609
## fitted.Predictor.002 0.1798107
## fitted.Predictor.003 0.1792185
## fitted.Predictor.004 0.1772138
## fitted.Predictor.005 0.1696571
## fitted.Predictor.006 0.1606400
```

```r
mean_values <- mod$summary.fitted.values$mean
lower_bound <- mod$summary.fitted.values$"0.025quant"
upper_bound <- mod$summary.fitted.values$"0.975quant"

plot(mean_values, type = "l", col = "blue", lwd = 2, ylab = expression(pi[t]), xlab = expression(x[t]),
#lines(MCM, col='red', lwd = 2, lty = 2)
lines(y/n, col="lightgrey")
lines(lower_bound, col = "cyan", lwd = 1, lty = 2)
lines(upper_bound, col = "cyan", lwd = 1, lty = 2)
lines(rain$day, mean, col="red", lwd = 2)
lines(lower, col="red", lwd = 1, lty=2)
lines(upper, col="red", lwd = 1, lty=2)

legend("topright", legend = c("INLA", 'MCMC', "95% CI INLA", "95% CI MCMC"),
       col = c("blue", 'red', "cyan", "red"), lwd = c(2, 2, 1, 1), lty = c(1, 1, 2, 2),
       inset = 0.05, cex=0.5)
```



As we can see from the plot, the differences in the predictions are very small. The fact that the models do not match exactly can be explained by the random component of the MCMC method.
The INLA method is faster that the MCMC one (against 200 seconds). We know that both of these methods converge (theoretically) but, the INLA method is preferred because it's faster.

b)

To assess the robustness of the results to the two `control.inla` inputs (`strategy` and `int.strategy`), we can fit the INLA model with different values for these options and compare the results.

For example, let's fit the INLA model with the following control options:

- `strategy` available: 'simplified.laplace' or 'laplace'
- `int.strategy` available: 'ccd', 'grid'

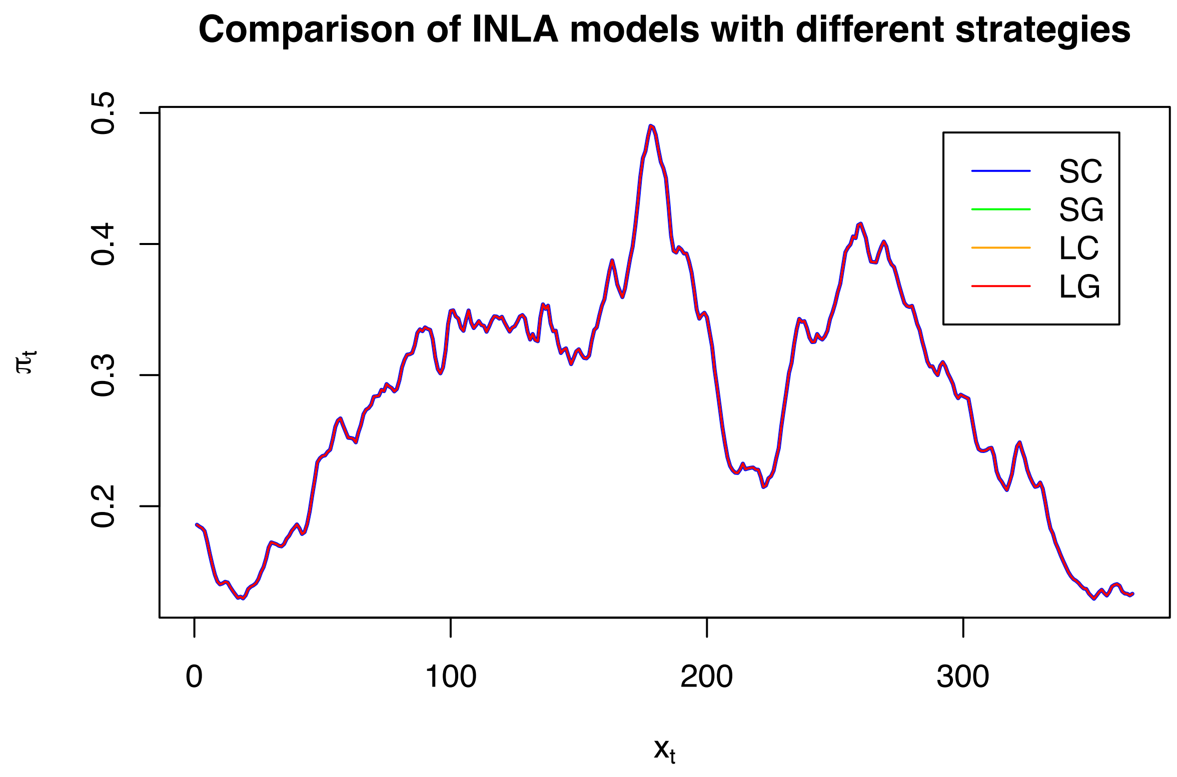
```
strategies = c('simplified.laplace', 'laplace')
int_strategies = c('ccd', 'grid')

INLA_mod = function(control.inla){
  return(inla(n.rain ~ -1 + f(day, model = "rw1", constr = FALSE, hyper=hypers),
               data = rain, Ntrials = n.years, control.inla = control.inla,
             family = "binomial", verbose=FALSE))
}

times = c()

t = proc.time()[3]
mod_sc = INLA_mod(control.inla=list(strategy='simplified.laplace',
                                    int.strategy='ccd'))
times = append(times, proc.time()[3] - t)
t = proc.time()[3]
mod_sg = INLA_mod(control.inla=list(strategy='simplified.laplace',
                                    int.strategy='grid'))
times = append(times, proc.time()[3] - t)
t = proc.time()[3]
mod_lc = INLA_mod(control.inla=list(strategy='laplace',
                                    int.strategy='ccd'))
times = append(times, proc.time()[3] - t)
t = proc.time()[3]
mod_lg = INLA_mod(control.inla=list(strategy='laplace',
                                    int.strategy='grid'))
times = append(times, proc.time()[3] - t)
```

```
times
```

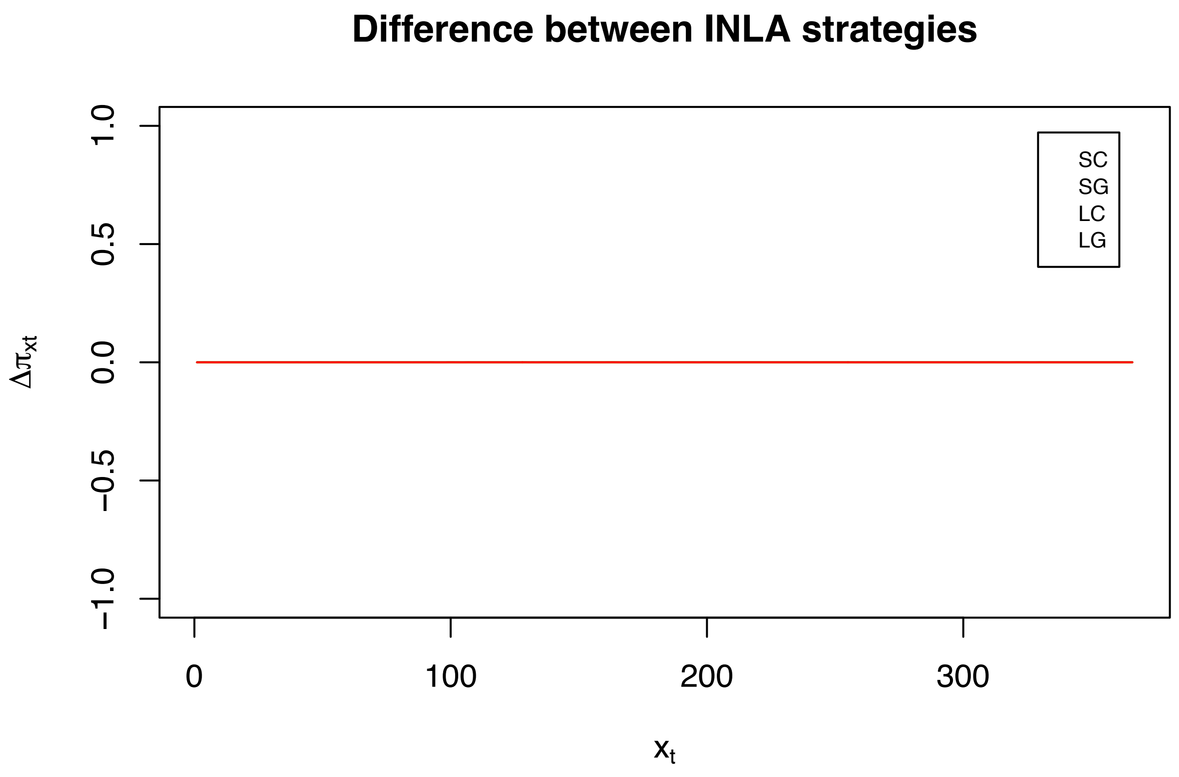
```
## elapsed elapsed elapsed elapsed
##    0.78    0.64    0.78    0.81
```

Computational times are very close to each other.

```
mean_sc = mod_sc$summary.fitted.values$mean
mean_sg = mod_sg$summary.fitted.values$mean
mean_lc = mod_lc$summary.fitted.values$mean
mean_lg = mod_lg$summary.fitted.values$mean

plot(mean_sc, type = "l", col = "blue", lwd = 2, ylab = expression(pi[t]),
     xlab = expression(x[t]),
     main = "Comparison of INLA models with different strategies")
lines(mean_sg,  type = "l", col = "green", lwd = 1 )
lines(mean_lc,  type = "l", col = "orange", lwd = 1)
lines(mean_lg,  type = "l", col = "red", lwd = 1)

legend("topright", legend = c("SC", "SG", "LC", "LG"),
       col = c("blue", "green", "orange", "red"),
       lwd = c(1, 1, 1, 1), lty = c(1, 1, 1, 1), inset = 0.05)
```



```r
plot(mean_sc-mean_sc, type = "l", col = "blue", lwd = 1,
     ylab = expression(Delta* pi[x][t]), xlab = expression(x[t]),
     main = "Difference between INLA strategies")
lines(mean_sg-mean_sc, type = "l", col = "green", lwd = 1)
lines(mean_lc-mean_sc, type = "l", col = "orange", lwd = 1)
lines(mean_lg-mean_sc, type = "l", col = "red", lwd = 1)

legend("topright", legend = c("SC", "SG", "LC", "LG"),
       col = c("blue", "green", "orange", "red"), inset = 0.05, cex=0.7)
```



These plots show that there is almost no differences between the INLA method with different hyper-parameters. Hence, the INLA method is very robust based on the `control.inla` parameters.

c)

We consider a new model in INLA.

```r
mod_c <- inla(n.rain ~ f(day, model="rw1", constr=TRUE, hyper=hypers),
              data=rain, Ntrials=n.years, control.compute=list(config = TRUE),
              family="binomial", verbose=FALSE, control.inla=control.inla)
```

The main differences with the previous model are:

- there is no `-1` that stands for the intercept (in this case there is an intercept). Now $Y_c = \beta_0 + \beta_1 \times X + ...$ and before $Y_a = \beta_1 \times X + ...$.
- `constr` is now true, which indicates whether to set a sum to 0 constraint on the term.

So it's mathematically different from the previous model. In the problem one, we had:

$$y_t|x_t \sim \mathcal{B}(n_t, \pi(x_t))$$

But now, we want to introduce an intercept term, called β_0.

$$y_t|x_t \sim \mathcal{B}(n_t, \pi(\beta_0 + x_t)).$$

If we set a sum to 0 constraint on the term, the following equation holds:

$$\sum_i x_i = 0.$$

And we know that:

$$\beta_0 + x_t = \log\left(\frac{\pi(\beta_0 + x_t)}{1 - \pi(\beta_0 + x_t)}\right).$$

We can analyze the two intercepts.

```
mod$summary.fixed
```

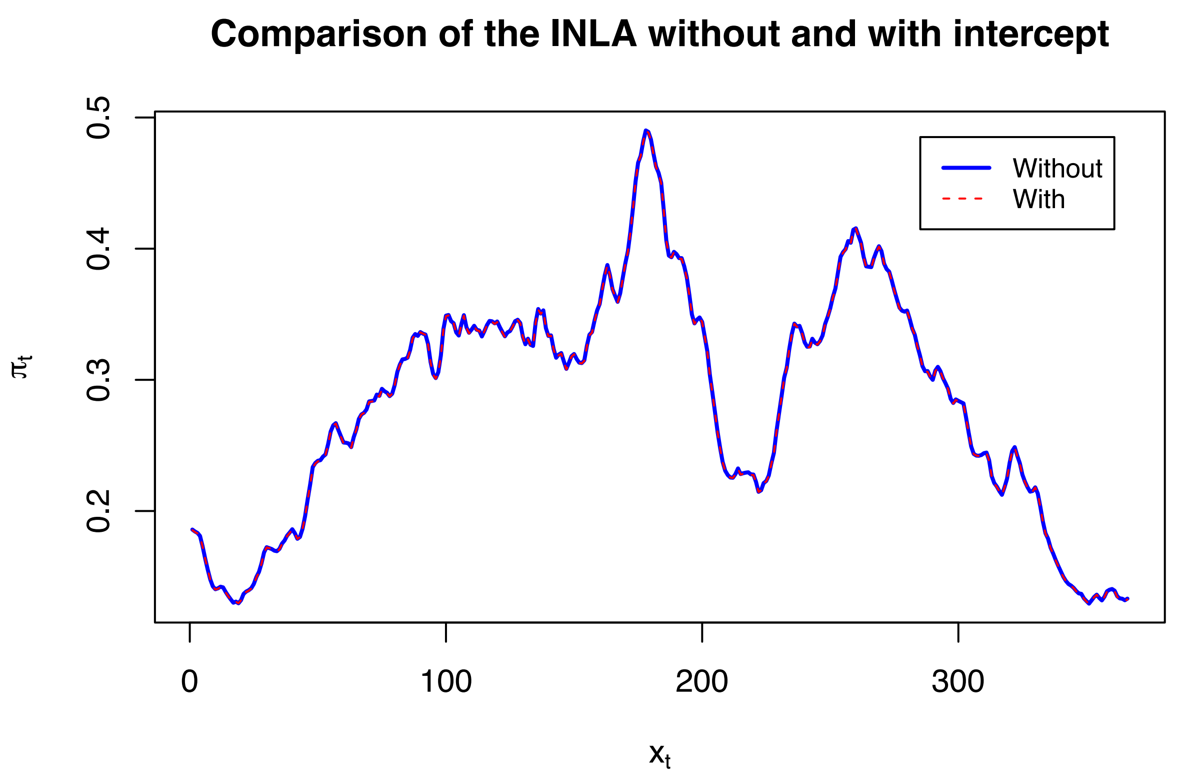
```
## data frame con 0 colonne e 0 righe
```

```
mod_c$summary.fixed
```

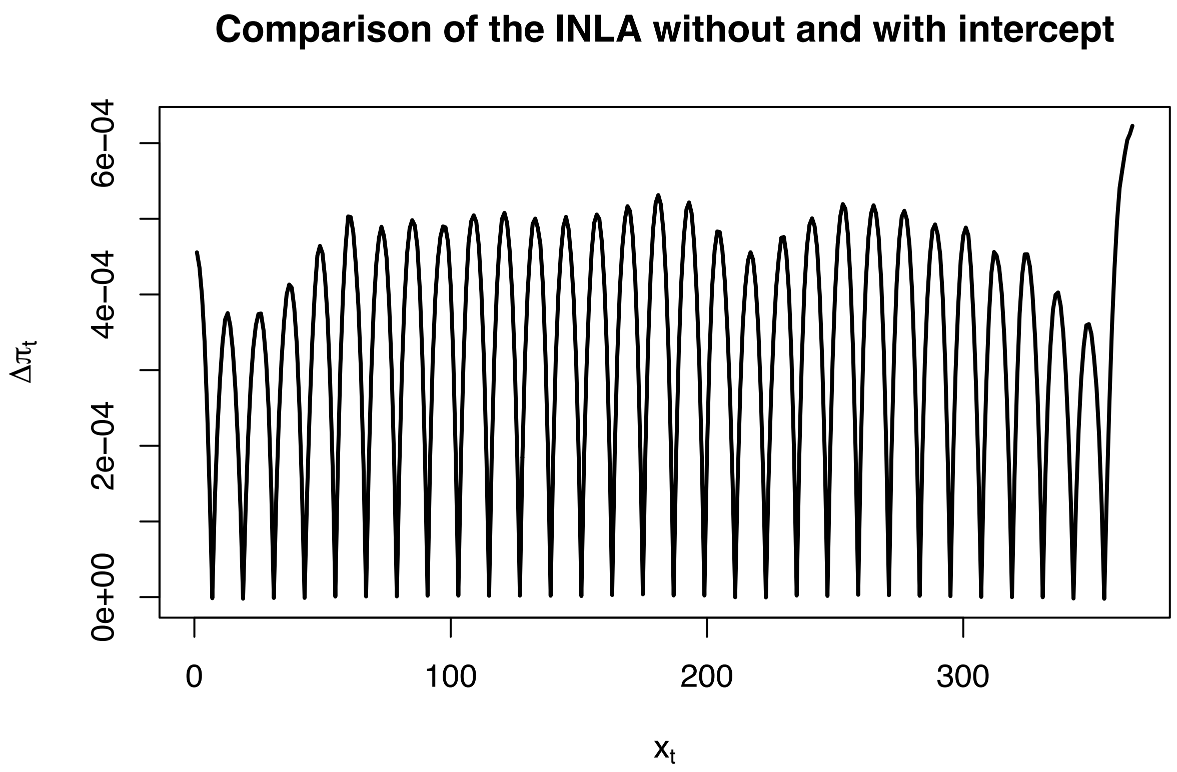
```
##                   mean        sd 0.025quant   0.5quant 0.975quant       mode
## (Intercept) -0.9877868 0.0195123   -1.02605 -0.9877861  -0.949527 -0.9877861
##                    kld
## (Intercept) 5.5623e-11
```

In the first INLA model `mod` there is no fixed value (no intercept) while in `mod_c` we found $\beta_0 \approx -0.987$ for the y-values.

```
mean_c <- mod_c$summary.fitted.values$mean

plot(mean_values, type = "l", col = "blue", lwd = 2, ylab = expression(pi[t]),
     xlab = expression(x[t]),
     main = "Comparison of the INLA without and with intercept")
lines(mean_c, col = "red", lty=2, lwd=1)

legend("topright", legend = c("Without", "With"), col = c("blue", "red"),
       lwd = c(2, 1), lty = c(1, 2), inset = 0.05, cex=0.8)
```



```r
plot(mean_values-mean_c, type = "l", lwd = 2, ylab = expression(Delta* pi[t]),
     xlab = expression(x[t]), main = "Comparison of the INLA without and with intercept")
```



Predictions are very similar so we can consider them as equivalent. However there are slightly different.

PROBLEM 3

Now we implement the same model using the R package `RTMB` that computes the estimations of the parameters via numerical optimization and automatic differentiation.

```r
#install.packages("RTMB")
library(RTMB)
```

The joint likelihood $p(\mathbf{y},\mathbf{x}|\theta)$ is:

$$p(\mathbf{y},\mathbf{x}|\theta) = \frac{p(\mathbf{y},\mathbf{x},\theta)}{p(\theta)} = \frac{p(\mathbf{y}|\mathbf{x},\theta)\ p(\mathbf{x},\theta)}{p(\theta)} = \frac{p(\mathbf{y}|\mathbf{x},\theta)\ p(\mathbf{x}|\theta)\ p(\theta)}{p(\theta)} = p(\mathbf{y}|\mathbf{x})\ p(\mathbf{x}|\theta)$$

$$= \prod_{t=1}^{T} \binom{n_t}{y_t} \frac{\exp{(x_t)}^{y_t}}{(1+\exp{(x_t)})^{n_t}} \prod_{t=2}^{T} \frac{1}{\sigma_u} \exp\left\{-\frac{1}{2\sigma_u^2}(x_t - x_{t-1})^2\right\}.$$

The log of the joint likelihood is:

$$\log(p(\mathbf{y},\mathbf{x}|\theta)) = \sum_{t=1}^{T} \log\left(\binom{n_t}{y_t} \frac{\exp{(x_t)}^{y_t}}{(1+\exp{(x_t)})^{n_t}}\right) + \sum_{t=2}^{T} \log\left(\frac{1}{\sigma_u} \exp\left\{-\frac{1}{2\sigma_u^2}(x_t - x_{t-1})^2\right\}\right).$$

Finally, the negative log of the joint likelihood is:

$$-\log(p(\mathbf{y},\mathbf{x}|\theta)) = -\sum_{t=1}^{T}\log\left(\binom{n_t}{y_t}\frac{\exp{(x_t)}^{y_t}}{(1+\exp{(x_t)})^{n_t}}\right) + \sum_{t=2}^{T}\left(\log(\sigma_u) + \frac{1}{2\sigma_u^2}(x_t - x_{t-1})^2\right).$$

We first write the function that computes the negative log of the likelihood $p(\mathbf{y},\mathbf{x}|\theta)$

```
T = 366
x_0 = rep(0,T)
sigma_0 = 0.5
li <- list(sigma=sigma_0, x=x_0)

f <- function(parms) {

  y <- rain$n.rain
  n <- rain$n.years
  T <- 366

  nll <- 0
  for(t in 1:T)
  {
    nll <- nll - dbinom(x=y[t],
                        size=n[t],
                        prob=exp(parms$x[t])/(1+exp(parms$x[t])),
                        log=TRUE)
  }
  for (t in 2:T){
    nll <- nll + log(parms$sigma) +(parms$x[t]-parms$x[t-1])^2/(2*parms$sigma^2)

  }
  return(nll)
}
```

Then, we pass this function to MakeADFun that computes the negative log of the Laplace approximation of the marginal likelihood $p(\mathbf{y}|\theta) = \int p(\mathbf{y},\mathbf{x}|\theta)\mathrm{d}\mathbf{x}$ using automatic differentiation:

```
obj <- MakeADFun(func = f,
                 parameters = li,
                 random = "x",
                 silent = TRUE)
```

Finally, we use nlminb on the approximate marginal likelihood to obtain the maximum likelihood estimates of the model parameters:

```
t = proc.time()[3]
opt <- nlminb(obj$par, obj$fn, obj$gr)
sdr <- sdreport(obj)
```

```
parameters <- sdreport(obj)
print(proc.time()[3]-t)
```

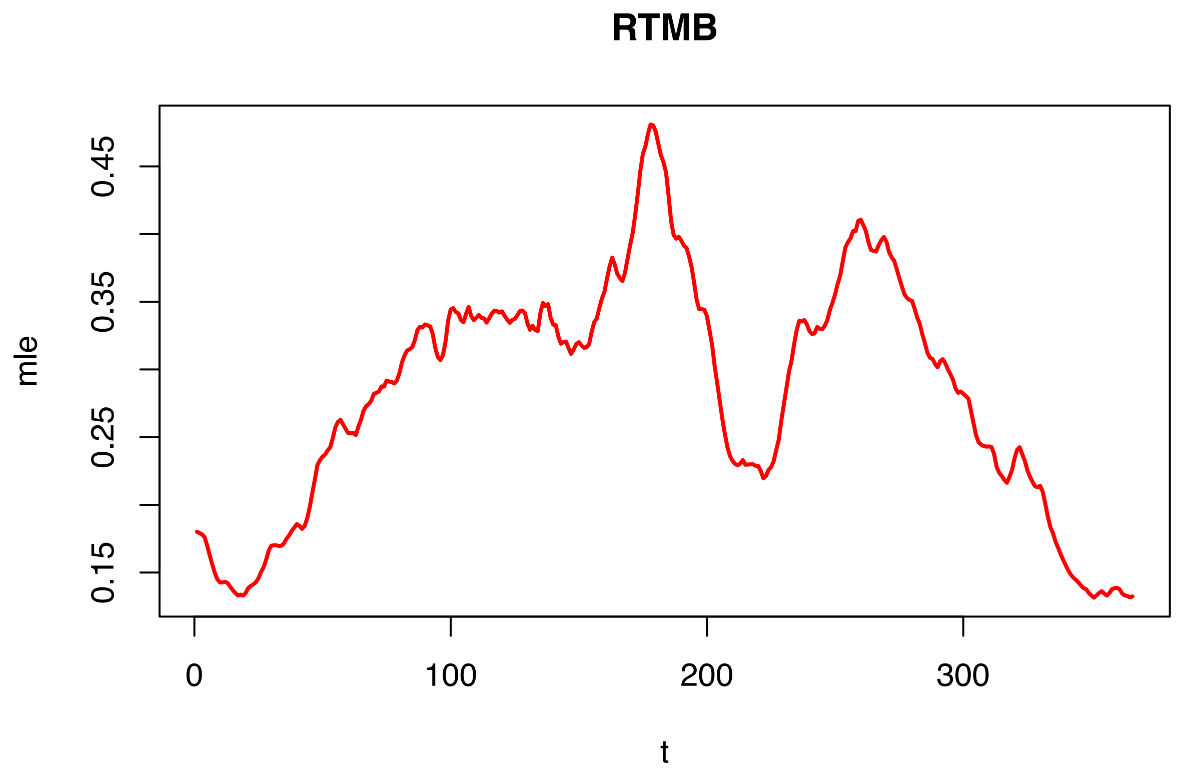
```
## elapsed
##    0.62
```

```
logit <- function(x) {1/(1+exp(-x))}

t_values <- 1:366
```

```r
mle_probs <- logit(parameters$par.random[1:366])

# creates a dataframe with time-values, mle x-values and confidence levels
df <- data.frame(t = t_values, mle=mle_probs)
```

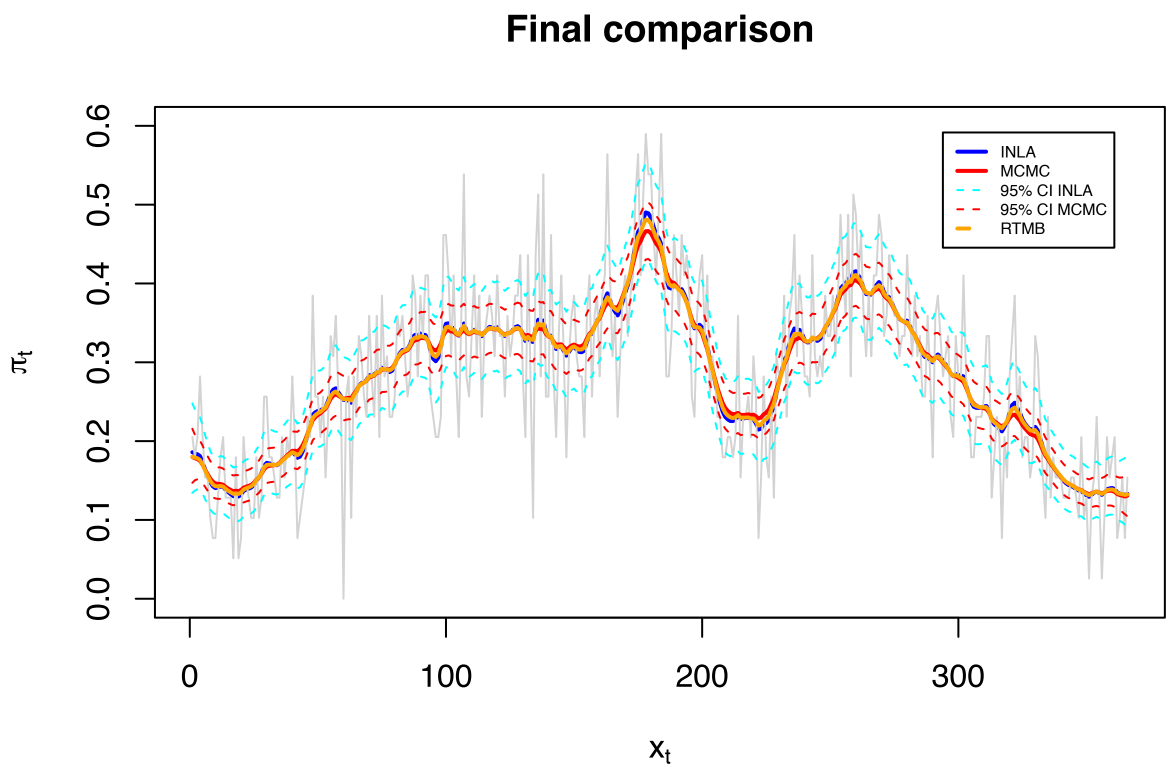
This method is also converging, faster than MCMC but a bit slower than INLA.

```r
# plots the mle x-values with y-values from the dataset
plot(df, type="l", lwd=2, col="red", main="RTMB")
```

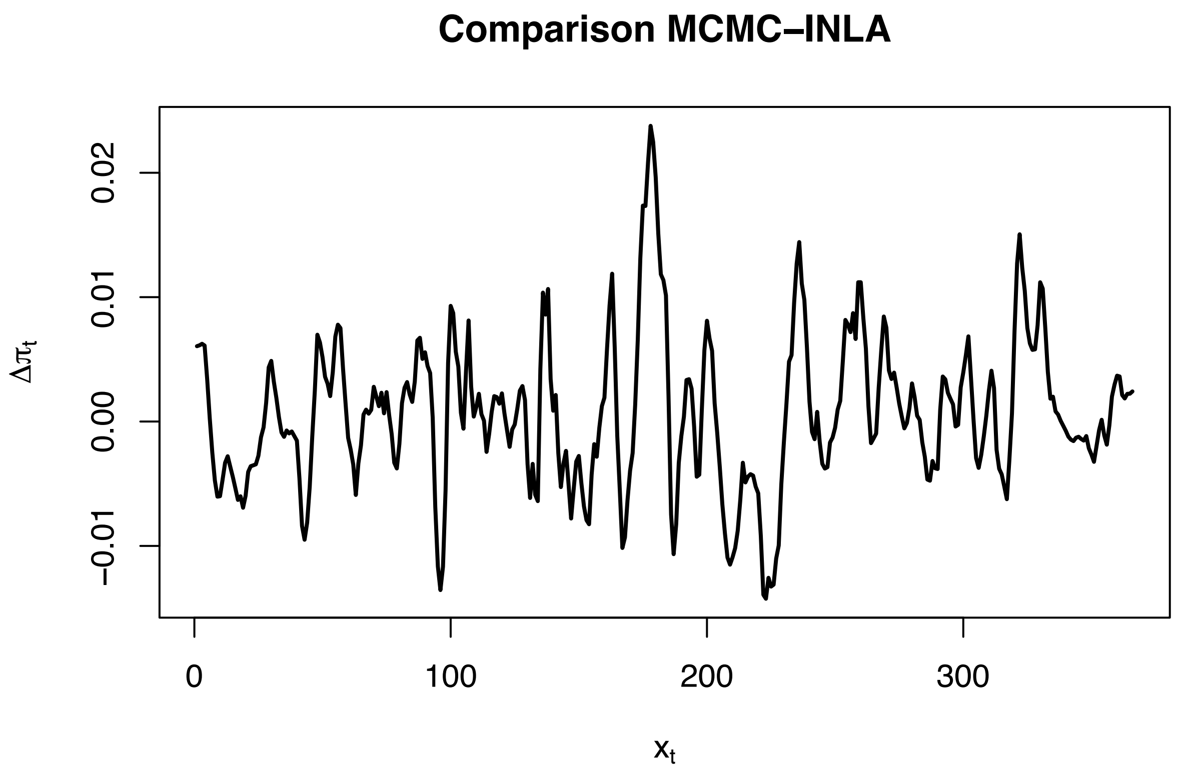


Finally we can compare all the results obtained.

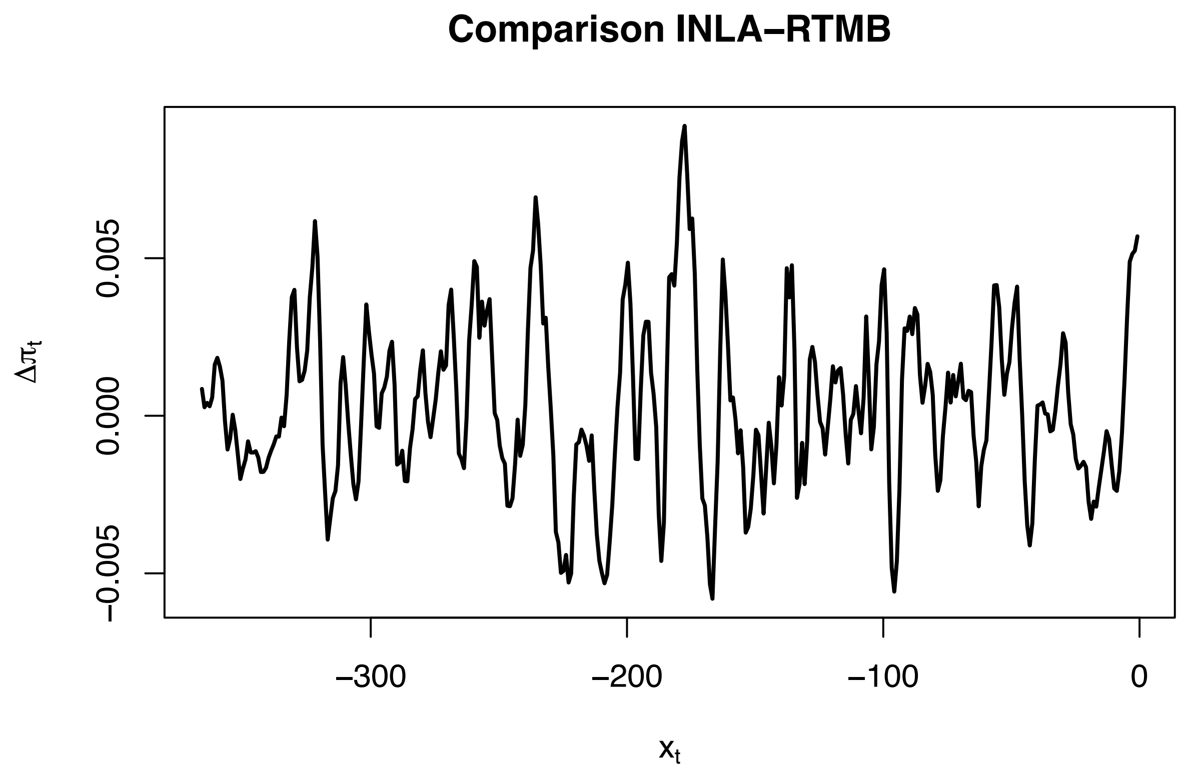
```r
plot(mean_values, type = "l", col = "blue", lwd = 2, ylab = expression(pi[t]),
     xlab = expression(x[t]), main = "Final comparison", ylim=c(0,0.6))
#lines(MCM, col='red', lwd = 2, lty = 2)
lines(y/n, col="lightgrey")
lines(lower_bound, col = "cyan", lwd = 1, lty = 2)
lines(upper_bound, col = "cyan", lwd = 1, lty = 2)
lines(rain$day, mean, col="red", lwd = 2)
lines(lower, col="red", lwd = 1, lty=2)
lines(upper, col="red", lwd = 1, lty=2)
lines(df, type="l", lwd=2, col="orange", main="RTBM")
legend("topright", legend = c("INLA", 'MCMC', "95% CI INLA", "95% CI MCMC", "RTMB"),
       col = c("blue", 'red', "cyan", "red", "orange"),
       lwd = c(2, 2, 1, 1, 2), lty = c(1, 1, 2, 2, 2),
       inset = 0.05, cex=0.5)
```



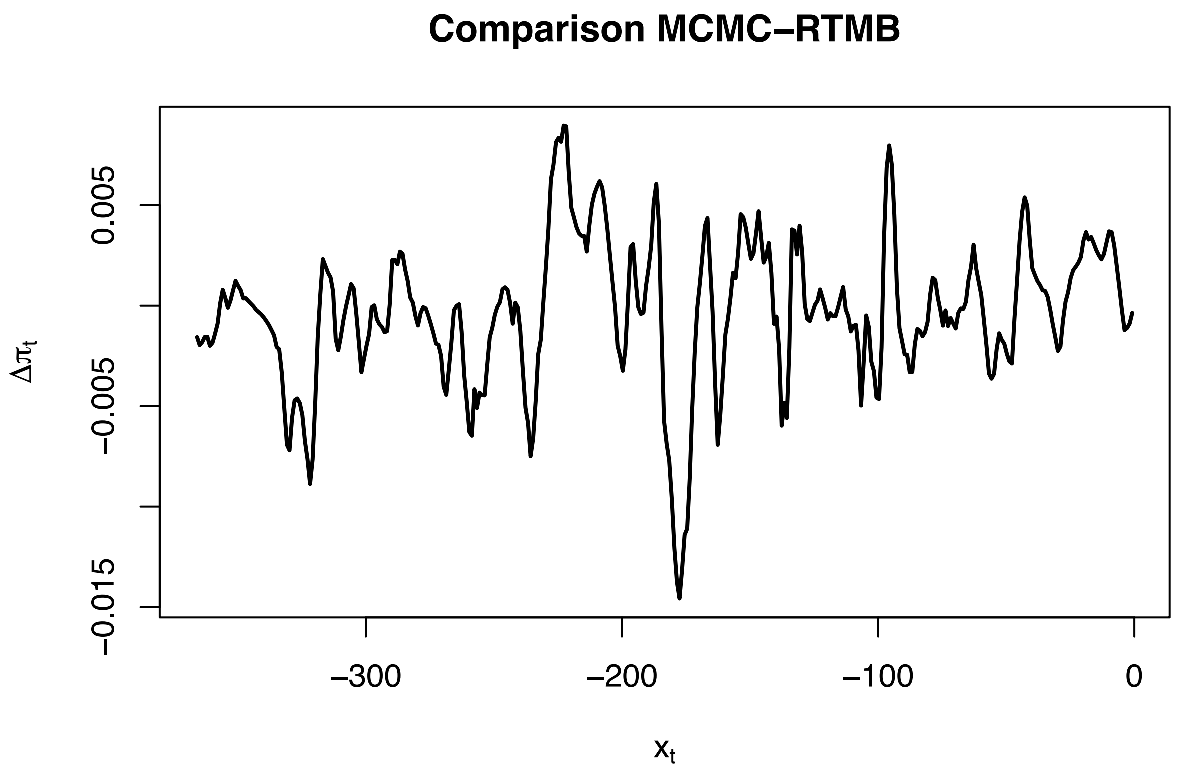
```
plot(mean_values-mean, type = "l", lwd = 2, ylab = expression(Delta* pi[t]),
     xlab = expression(x[t]), main = "Comparison MCMC-INLA")
```



```
plot(mean_values-df, type = "l", lwd = 2, ylab = expression(Delta* pi[t]),
     xlab = expression(x[t]), main = "Comparison INLA-RTMB")
```



```
plot(mean-df, type = "l", lwd = 2, ylab = expression(Delta* pi[t]),
     xlab = expression(x[t]), main = "Comparison MCMC-RTMB")
```


Comparison MCMC−RTMB
0.005
−0.005
−0.015
$\Delta\pi_t$
−300
−200
−100
0
x_t

TMA4300 – Computer Intensive Statistical Methods Project 3

Elena Dami, Marco Pozzetto & Marijan Sorić

2024-04-28

Problem 1: Bootstrapping a GLM

In this problem, we assume that $Y_1, \dots, Y_n$ are independent random variables and that each $Y_i \sim Bin(m_i, p_i)$ where $\ln(\frac{p_i}{1-p_i}) = \beta_0 + \beta_1 x_i$ for $i = 1, \dots, n$.
Firstly, we load into R the data we will use:

```
load(file=url("https://www.math.ntnu.no/emner/TMA4300/2024v/data.Rdata"))
```

This is a generalized linear model and it can be fitted in R by using the maximum likelihood. In particular, we can fit the model and extract the MLE $\widehat{\beta}$ of β. Moreover, we can obtain an approximate estimate of $\mathrm{Var}(\widehat{\beta})$ given by $(F(\widehat{\beta}))^{-1}$ where $F(\beta)$ is the Fisher information matrix.

```
mod <- glm(cbind(y, m - y) ~ x, family = binomial, data = data)
```

Maximum likelihood estimates of β_0 and β_1:

```
mod$coef
```

```
## (Intercept)           x
##  -0.6781067   0.1009905
```

Variance matrix of MLEs of $\widehat{\beta}_0$ and $\widehat{\beta}_1$:

```
vcov(mod)
```

```
##             (Intercept)            x
## (Intercept)  0.19649741 -0.028022675
## x           -0.02802267  0.005049491
```

a)

Now we use non-parametric bootstrapping in order to obtain the bootstrap replicates $\widehat{\beta}^{b*}$ of $\widehat{\beta}$. In particular, we generate $B = 10000$ bootstrap samples by resampling the observations triplets y_i, m_i, x_i with replacements and we refit the model *mod* to each bootstrap sample.

```
bootstrap=function(mod,B){
  beta=matrix(NA, nrow = B, ncol = 2)
  n=nrow(data)
  for (b in 1:B){
    i=sample(1:n, replace = TRUE)
    bootstrap_sample=data[i, ]
    mod=glm(cbind(y, m - y) ~ x, family = binomial, data=bootstrap_sample)
    beta[b,]=mod$coef
  }
  return(beta)
}
B=10000
```

```
replicates=bootstrap(mod,B)
dim(replicates)
```

```
## [1] 10000     2
```

b)

Then we can compute an estimate of $\text{Var}(\widehat{\beta})$ based on the bootstrap replicates $\widehat{\beta}^{b*}$.

```
cov(replicates)
```

```
##              [,1]        [,2]
## [1,]  0.59279212 -0.07944716
## [2,] -0.07944716  0.01133834
```

The variances of $\widehat{\beta}_0$ and $\widehat{\beta}_1$ are much higher compared to the corresponding variance obtained using `vcov(mod)`.

c)

The bias of an estimator $\widehat{\theta}$ of a parameter θ is $\text{bias}(\widehat{\theta}) = \mathbb{E}(\widehat{\theta}) - \theta$. Thus, we estimate the bias of the MLEs $\widehat{\beta}$ with $\widehat{\text{bias}(\widehat{\beta})} = \widehat{\mathbb{E}(\widehat{\beta})} - \widehat{\beta}$:

```
bias=colMeans(replicates)-mod$coef
bias
```

```
##  (Intercept)            x 
## -0.064924766  0.008894749
```

The estimators appear to be significantly biased and so we compute the bias-corrected estimator $\widehat{\theta}_c = \widehat{\theta} - \widehat{\text{bias}(\widehat{\beta})}$:

```
bc_estimator=mod$coef-bias
bc_estimator
```

```
## (Intercept)           x 
## -0.61318192  0.09209577
```

The bias-corrected estimator has typically less but not zero bias and higher variance.

d)

Now we compute an approximate 95%-confidence interval for each model parameter.

```
IC_b0=quantile(replicates[,1], c(0.025,0.975))
print(IC_b0)
```

```
##       2.5%      97.5% 
## -2.0306641  0.8482333
```

```
IC_b1=quantile(replicates[,2], c(0.025,0.975))
print(IC_b1)
```

```
##       2.5%      97.5% 
## -0.1038849  0.3007656
```

Then we compare them with the confidence intervals obtained based on the profile likelihood of each parameter.

```
confint(mod, level=0.95)
```

```
##                   2.5 %    97.5 %
## (Intercept) -1.5718941 0.1779204
## x           -0.0367474 0.2434345
```

As expected, the confidence intervals for the intercept and the slope parameters using the bootstrap replicates are much wider than the confidence intervals base on the profile likelihood because they have a higher variance.

e)

Now we use parametric bootstrapping and we estimate β by $\widehat{\beta}$ and $F(x, \beta)$ by $F(x, \widehat{\beta})$. In particular, in order to obtain the bootstrap replicates $\widehat{\beta}^{b*}$ of $\widehat{\beta}$, we generate $B = 10000$ bootstrap samples of n triplets y_i, m_i, x_i each by sampling $y_i \sim Bin(m_i, p_i)$ where $\ln(\frac{p_i}{1-p_i}) = \widehat{\beta}_0 + \widehat{\beta}_1 x_i$ for $i = 1, \ldots, n$ and we refit the model *mod* to each bootstrap sample.

```r
B=10000
parametric_replicates=matrix(NA, nrow = B, ncol = 2)
for(b in 1:B) {
  n=nrow(data)
  m=data$m
  y_hat=rbinom(n, m, predict(mod, type = "response"))
  bootstrap_data=data.frame(y=y_hat,m=data$m,x=data$x)
  par_bootstrap =glm(cbind(y, m - y) ~ x, family = binomial, data = bootstrap_data)
  parametric_replicates[b,]=par_bootstrap$coef
}
```

As before, we compute an estimate of $\mathrm{Var}(\widehat{\beta})$.

```r
cov(parametric_replicates)
```

```
##             [,1]         [,2]
## [1,]  0.20692713 -0.029627787
## [2,] -0.02962779  0.005336442
```

The variances of $\widehat{\beta}_0$ and $\widehat{\beta}_1$ are smaller compared to the corresponding variance obtained using non-parametric bootstrapping.

```r
bias=colMeans(parametric_replicates)-mod$coef
bias
```

```
## (Intercept)           x
## -0.02292487  0.00387414
```

```r
bc_estimator=mod$coef-bias
bc_estimator
```

```
## (Intercept)           x
## -0.65518182  0.09711638
```

Moreover, the estimators have less bias with respect to the case of non-parametric bootstrapping.

```r
IC_b0=quantile(parametric_replicates[,1], c(0.025,0.975))
print(IC_b0)
```

```
##       2.5%      97.5%
## -1.6351295  0.1603046
```

```r
IC_b1=quantile(parametric_replicates[,2], c(0.025,0.975))
print(IC_b1)
```

```
##         2.5%       97.5%
## -0.03543415  0.25186600
```

```
confint(mod, level=0.95)
```

```
##                   2.5 %    97.5 %
## (Intercept) -1.5718941 0.1779204
## x           -0.0367474 0.2434345
```

Finally, the confidence intervals are less wide and similar to intervals the confidence intervals obtained based on the profile likelihood of each parameter. Thus, all our results are more precise in the case of parametric bootstrapping and it means that the bootstrap samples are genereted from the correct distribution $F(x, \beta)$ where β is estimated by $\widehat{\beta}$.

Problem 2: Bootstrap confidence intervals

a)

Suppose that $X_1, X_2, ..., X_n$ is an iid sample from an exponential distribution with scale parameter β.

Let us show that pivotal quantity $Y = \frac{2\sum_{i=1}^n X_i}{\beta}$ is chi-square with $2n$ degrees of freedom. We will do that using the moment generating function.

Let us recall the moment generating function of a chi-square with ν degrees of freedom

$$M_X(t) = \left(\frac{1}{1-2t}\right)^{\frac{\nu}{2}}, \text{ for } t < \frac{1}{2},$$

and the moment generating function of an exponential distribution with scale parameter β

$$M_X(t) = \frac{1}{1-\beta t}, \text{ for } t < \frac{1}{\beta}.$$

Now let us compute the moment generating function of Y. In the computation we considered that the variables are iid,

$$\begin{aligned}
M_Y(t) = \mathbb{E}[e^{tY}] &= \mathbb{E}\left[e^{t\frac{2\sum_{i=1}^n X_i}{\beta}}\right] = \mathbb{E}\left[\prod_{i=1}^n e^{2t\frac{X_i}{\beta}}\right] \\
&= \prod_{i=1}^n \mathbb{E}\left[e^{2t\frac{X_i}{\beta}}\right] = \left(\mathbb{E}\left[e^{2t\frac{X_1}{\beta}}\right]\right)^n = \left(M_{X_1}\left(\frac{2t}{\beta}\right)\right)^n \\
&= \left(\frac{1}{1-\frac{\beta 2t}{\beta}}\right)^n = \left(\frac{1}{1-2t}\right)^n \\
&\Rightarrow \text{MGF of a chi-square distribution with } 2n \text{ degrees of freedom.}
\end{aligned}$$

Using this result, we can derive an exact $(1-\alpha)$ confidence interval for β

$$\begin{aligned}
\mathbb{P}\left(\chi^2_{2n,\frac{\alpha}{2}} < \frac{2\sum_{i=1}^n X_i}{\beta} < \chi^2_{2n,1-\frac{\alpha}{2}}\right) = 1-\alpha \\
\mathbb{P}\left(\frac{2\sum_{i=1}^n X_i}{\chi^2_{2n,1-\frac{\alpha}{2}}} < \beta < \frac{2\sum_{i=1}^n X_i}{\chi^2_{2n,\frac{\alpha}{2}}}\right) = 1-\alpha.
\end{aligned}$$

Thus, an exact $(1-\alpha)$ confidence interval for β is

$$\left(\frac{2\sum_{i=1}^n X_i}{\chi^2_{2n,1-\frac{\alpha}{2}}}, \frac{2\sum_{i=1}^n X_i}{\chi^2_{2n,\frac{\alpha}{2}}}\right). \tag{1}$$

b)

Let us now use parametric bootstrapping and construct a confidence interval for β using the percentile method, i.e. using the empirical $\alpha/2$ and $1-\alpha/2$ quantiles of the distribution of boostrap replicates $\widehat{\beta^*}$ of $\widehat{\beta}$, where $\widehat{\beta}$ is the MLE of β.

The maximum likelihood estimator of β, scale parameter of an exponential distribution, is

$$\widehat{\beta} = \frac{\sum_{i=1}^n X_i}{n}.$$

Using what we found in part a), we get that

$$\frac{2\sum_{i=1}^n X_i}{\beta} = \frac{2n}{\beta}\widehat{\beta} \sim \chi^2_{2n},$$

which implies that

$$\widehat{\beta} \sim \frac{\beta}{2n}\chi^2_{2n}.$$

Thus, when $\widehat{\beta^*}$ are based on bootstrap samples from $\text{Exp}(1/\widehat{\beta})$, we have that

$$\widehat{\beta}^* \sim \frac{\widehat{\beta}}{2n}\chi^2_{2n}.$$

From this result, we can derive the following $(1-\alpha)$ confidence interval for β

$$\left(\widehat{\beta}^*_{\frac{\alpha}{2}}, \widehat{\beta}^*_{1-\frac{\alpha}{2}}\right) = \left(\frac{\widehat{\beta}}{2n}\chi^2_{2n,\frac{\alpha}{2}}, \frac{\widehat{\beta}}{2n}\chi^2_{2n,1-\frac{\alpha}{2}}\right), \tag{2}$$

which is different from the exact confidence interval found in part a).

Note: Given the relation between the chi-square distribution and the gamma distribution, the exact distribution of $\widehat{\beta}^*$ is $\widehat{\beta}^* \sim \text{Gamma}\left(n, \frac{n}{\widehat{\beta}}\right)$.

c)

We proceed by finding an expression for the exact coverage of the parametric bootstrap percentile interval found in part b), in terms of the cdf and quantile function of the chi-square distribution.

Starting from

$$\mathbb{P}\left(\frac{\widehat{\beta}}{2n}\chi^2_{2n,\frac{\alpha}{2}} < \beta < \frac{\widehat{\beta}}{2n}\chi^2_{2n,1-\frac{\alpha}{2}}\right) = 1-\alpha,$$

given that $\beta \sim \frac{2n\widehat{\beta}}{\chi^2_{2n}}$,

$$\begin{aligned}
&\mathbb{P}\left(\frac{\widehat{\beta}}{2n}\chi^2_{2n,\frac{\alpha}{2}} < \frac{2n\widehat{\beta}}{\chi^2_{2n}} < \frac{\widehat{\beta}}{2n}\chi^2_{2n,1-\frac{\alpha}{2}}\right) \\
&= \mathbb{P}\left(\frac{\chi^2_{2n,\frac{\alpha}{2}}}{4n^2} < \frac{1}{\chi^2_{2n}} < \frac{\chi^2_{2n,1-\frac{\alpha}{2}}}{4n^2}\right) \\
&= \mathbb{P}\left(\frac{4n^2}{\chi^2_{2n,1-\frac{\alpha}{2}}} < \chi^2_{2n} < \frac{4n^2}{\chi^2_{2n,\frac{\alpha}{2}}}\right) \\
&= \mathbb{P}\left(\chi^2_{2n} \le \frac{4n^2}{\chi^2_{2n,\frac{\alpha}{2}}}\right) - \mathbb{P}\left(\chi^2_{2n} \le \frac{4n^2}{\chi^2_{2n,1-\frac{\alpha}{2}}}\right) = 1-\alpha.
\end{aligned} \tag{3}$$

Next, we write a function that computes the coverage $1-\alpha$ as in Equation 3.

```
compute_coverage <- function(n,alpha) {
  # compute chi-square quantiles
  q1= qchisq(p= alpha/2, df=2*n)
  q2= qchisq(p= 1-alpha/2, df=2*n)
  # compute exact coverage
  coverage = pchisq(q= (4*n^2)/q1, df=2*n) - pchisq(q= (4*n^2)/q2, df=2*n)
  return(coverage)
}
```

We proceed by computing the exact coverage for $n = 5, 10, 20, 50, 100$ and $\alpha = 0.05$.

```
alpha=0.05
for(i in c(5,10,20,50,100)) {
  coverage <- compute_coverage(i,alpha)
  print(paste("n=", i, ":", round(coverage,4)))
}
```

```
## [1] "n= 5 : 0.8983"
## [1] "n= 10 : 0.9228"
## [1] "n= 20 : 0.9361"
## [1] "n= 50 : 0.9443"
## [1] "n= 100 : 0.9472"
```

We can see that as n gets bigger, the coverage gets closer to 0.95.

d)

We now aim to write a function that, taking as inputs the observed sample $x_1, x_2, ..., x_n$ and α, computes a two-sided $(1-\alpha)$ BC_a-confidence interval for β based on parametric bootstrapping.

This method usually offers improvement over the simple percentile approach.

According to this approach,

$$\mathbb{P}\left[\xi_{\beta 1} \le \beta \le \xi_{\beta 2}\right] \approx 1-\alpha,$$

where

$$\beta_1 = \phi\left(b + \frac{b + z_{\alpha/2}}{1 - a(b + z_{\alpha/2})}\right),$$

$$\beta_2 = \phi\left(b + \frac{b + z_{1-\alpha/2}}{1 - a(b + z_{1-\alpha/2})}\right),$$

and $\xi_{\beta 1}$ and $\xi_{\beta 2}$ are the corresponding quantiles from the bootstrapped values of $\widehat{\beta}^*$.

We choose a and b such that

$$a = \frac{\sum_{i=1}^n \psi_i^3}{6\left(\sum_{i=1}^n \psi_i^2\right)^{\frac{3}{2}}},$$

where $\psi_i = \frac{1}{n}\sum_{i=1}^n \widehat{\beta}_{(-i)} - \widehat{\beta}_{(-i)}$, and $\widehat{\beta}_{(-i)}$ denotes the statistic computed omitting the ith observation, and

$$b = \phi^{-1}\left(F_{\widehat{\beta}^*}(\widehat{\beta})\right),$$

where $F_{\widehat{\beta}^*}$ is the exact distribution of $\widehat{\beta}^*$.

```r
# Function that computes two-sided BC_a-confidence interval for beta
# Inputs:
# x: vector of the observed sample
# alpha: significance level

BCa_confint <- function(x, alpha) {

  n = length(x)
  beta_hat = mean(x)

  # compute constant a
  psi = mean(sapply(1:n, function(i) mean(x[-i]))) -
    sapply(1:n, function(i) mean(x[-i]))
  a = sum(psi^3)/(6*sum(psi^2)^(3/2))

  # compute constant b
  p = pgamma(q=beta_hat, shape=n, rate=n/beta_hat)
  b = qnorm(p)

  # compute beta1 and beta2
  z1 = qnorm(alpha/2)
  z2 = qnorm(1-alpha/2)
  beta1 = pnorm(b + (b + z1)/(1 - a*(b + z1)))
  beta2 = pnorm(b + (b + z2)/(1 - a*(b + z2)))

  # compute quantiles
  e1 = qgamma(p= beta1, shape=n, scale=beta_hat/n)
  e2 = qgamma(p= beta2, shape=n, scale=beta_hat/n)

  return(c(e1,e2))
}
```

e)

Lastly, we apply our function to a simulated data set.
We simulate 10000 random samples, each of size $n = 10$ from the exponential distribution, assuming the true value of β to be 1.
We will estimate the coverage of the BC_a interval in part d) for $\alpha = 0.05$, and check if the interval contains β.

```r
n = 10
true_beta = 1
alpha = 0.05
```

```r
tot_samples = 10000
count <- 0
for(i in 1: tot_samples){
  sample <- rexp(n, rate=1/true_beta)
  ci <- BCa_confint(sample, alpha)
  if(ci[1] < true_beta && true_beta < ci[2]) {
    count <- count + 1
  }
}
coverage = count/tot_samples
cat("Coverage using BC_a method: ", coverage)
```

```
## Coverage using BC_a method:  0.9463
```

The coverage using BC_a method results higher then the coverage computed using the percentile interval in part c). For $n = 10$ it is close to 0.95.

Problem 3: The EM-algorithm and bootstrapping

Let $x_i \sim \mathcal{E}xp(\lambda_0)$ and $y_i \sim \mathcal{E}xp(\lambda_1)$ be independent random variables (for $1 \leq i \leq n$). Those r.v. aren't observed directly but $z_i = \max(x_i, y_i)$ and $u_i = I(x_i \geq y_i)$ are.

a)

Let's compute the Log likelihood function for the complete data (x_i, y_i):

$$
\begin{aligned}
\log f_{X,Y}(x_1, ..., x_n, y_1, ..., y_n|\lambda_0, \lambda_1) &= \log f_{X,Y}(\mathbf{x}, \mathbf{y}|\lambda_0, \lambda_1) \\
&= \log f_X(\mathbf{x}|\lambda_0, \cancel{\lambda_1}) f_Y(\mathbf{y}|\ \cancel{\lambda_0}, \lambda_1) \\
&= \log \prod_{i=1}^{n} f_{X_i}(x_i|\lambda_0) f_{Y_i}(y_i|\lambda_1) \\
&= \log \prod_{i=1}^{n} \lambda_0 e^{-\lambda_0 x_i} \lambda_1 e^{-\lambda_1 y_i} \\
&= \sum_{i=1}^{n} \log \lambda_0 - \lambda_0 x_i + \log \lambda_1 - \lambda_1 y_i \\
&= n(\log \lambda_0 + \log \lambda_1) - \sum_{i=1}^{n} \lambda_0 x_i + \lambda_1 y_i
\end{aligned}
$$

Then, we compute the expectation of the joint log likelihood for the complete data conditioned on the observed data.

$$
\mathbb{E}\left[\log f_{X,Y}(\mathbf{x}, \mathbf{y}|\lambda_0, \lambda_1)|\mathbf{z}, \mathbf{u}, \lambda_0^{(t)}, \lambda_1^{(t)}\right] = n(\log \lambda_0 + \log \lambda_1) - \lambda_0 \sum_{i=1}^{n} \mathbb{E}\left[x_i|\mathbf{z}, \mathbf{u}, \lambda_0^{(t)}, \lambda_1^{(t)}\right] - \lambda_1 \sum_{i=1}^{n} \mathbb{E}\left[y_i|\mathbf{z}, \mathbf{u}, \lambda_0^{(t)}, \lambda_1^{(t)}\right]
$$

We want to express conditional expectation: $\mathbb{E}\left[x_i|\mathbf{z}, \mathbf{u}, \lambda_0^{(t)}, \lambda_1^{(t)}\right]$ and $\mathbb{E}\left[y_i|\mathbf{z}, \mathbf{u}, \lambda_0^{(t)}, \lambda_1^{(t)}\right]$ in terms of conditional expectation conditionate on x_i, y_i instead of $\mathbf{z}, \mathbf{u}$. We keep in mind that $u_i \in \{0,1\}$.

$$\begin{aligned}
\mathbb{E}\left[x_i|\mathbf{z},\mathbf{u},\lambda_0^{(t)},\lambda_1^{(t)}\right] &= \mathbb{E}\left[x_i|\max(x_i,y_i)=z_i, I(x_i \geq y_i)=u_i,\lambda_0^{(t)},\lambda_1^{(t)}\right] \\
&= \sum_{u_i\in\{0,1\}} I(U_i=u_i)\mathbb{E}\left[x_i|\max(x_i,y_i)=z_i, I(x_i \geq y_i)=u_i,\lambda_0^{(t)},\lambda_1^{(t)}\right] \\
&= u_i\mathbb{E}\left[x_i|\max(x_i,y_i)=z_i, x_i \geq y_i,\lambda_0^{(t)},\lambda_1^{(t)}\right] + (1-u_i)\mathbb{E}\left[x_i|\max(x_i,y_i)=z_i, x_i < y_i,\lambda_0^{(t)},\lambda_1^{(t)}\right] \\
&= u_i\mathbb{E}\left[x_i|x_i=z_i, x_i \geq y_i,\lambda_0^{(t)},\lambda_1^{(t)}\right] + (1-u_i)\mathbb{E}\left[x_i|y_i=z_i, x_i < y_i,\lambda_0^{(t)},\lambda_1^{(t)}\right] \\
&= u_i\mathbb{E}\left[x_i|x_i=z_i,\lambda_0^{(t)}\right] + (1-u_i)\mathbb{E}\left[x_i|x_i<z_i,\lambda_0^{(t)},\lambda_1^{(t)}\right] \\
&= u_i z_i + (1-u_i)\mathbb{E}\left[x_i|x_i<z_i,\lambda_0^{(t)}\right]
\end{aligned}$$

Similarly,

$$\mathbb{E}\left[y_i|\mathbf{z},\mathbf{u},\lambda_0^{(t)},\lambda_1^{(t)}\right] = (1-u_i)z_i + u_i\mathbb{E}\left[y_i|x_i<z_i,\lambda_1^{(t)}\right]$$

By definition of the conditional expectation, we have:

$$\mathbb{E}\left[x_i|x_i<z_i,\lambda_0^{(t)}\right] = \int_{\mathbb{R}_+} x\mathbb{P}\left[x_i=x|x_i<z_i,\lambda_0^{(t)}\right]\mathrm{d}x$$

We use Bayes formula to expression the conditional probability density function (that is a function of x). Note: $x_i \sim \mathcal{E}xp(\lambda_0^{(t)})$.

$$\mathbb{P}\left[x_i=x|x_i<z_i,\lambda_0^{(t)}\right] = \frac{\mathbb{P}\left[x_i<z_i|x_i=x,\lambda_0^{(t)}\right]\mathbb{P}\left[x_i=x|\lambda_0^{(t)}\right]}{\mathbb{P}\left[x_i<z_i|\lambda_0^{(t)}\right]} = \frac{\mathbb{P}\left[x<z_i\right]\lambda_0^{(t)}e^{-\lambda_0^{(t)}x}}{1-e^{-\lambda_0^{(t)}z_i}} = \frac{I(x<z_i)\lambda_0^{(t)}e^{-\lambda_0^{(t)}x}}{1-e^{-\lambda_0^{(t)}z_i}}$$

We can finally compute the conditional expectation.

$$\begin{aligned}
\mathbb{E}\left[x_i|x_i<z_i,\lambda_0^{(t)}\right] &= \int_{\mathbb{R}_+} x\mathbb{P}\left[x_i=x|x_i<z_i,\lambda_0^{(t)}\right]\mathrm{d}x \\
&= \int_{\mathbb{R}_+} x\frac{I(x<z_i)\lambda_0^{(t)}e^{-\lambda_0^{(t)}x}}{1-e^{-\lambda_0^{(t)}z_i}}\mathrm{d}x \\
&= \frac{1}{1-e^{-\lambda_0^{(t)}z_i}}\int_0^{z_i} x\lambda_0^{(t)}e^{-\lambda_0^{(t)}x}\mathrm{d}x \\
&= \frac{1}{1-e^{-\lambda_0^{(t)}z_i}}\left(\left[x(-e^{-\lambda_0^{(t)}x})\right]_0^{z_i} - \int_0^{z_i} 1(-e^{-\lambda_0^{(t)}x})\mathrm{d}x\right) \\
&= \frac{1}{1-e^{-\lambda_0^{(t)}z_i}}\left(-z_ie^{-\lambda_0^{(t)}z_i} + \int_0^{z_i} e^{-\lambda_0^{(t)}x}\mathrm{d}x\right) \\
&= \frac{1}{1-e^{-\lambda_0^{(t)}z_i}}\left(-z_ie^{-\lambda_0^{(t)}z_i} + \left[-\frac{1}{\lambda_0^{(t)}}e^{-\lambda_0^{(t)}x}\right]_0^{z_i}\right) \\
&= \frac{1}{1-e^{-\lambda_0^{(t)}z_i}}\left(-z_ie^{-\lambda_0^{(t)}z_i} + \frac{1}{\lambda_0^{(t)}}\left(1-e^{-\lambda_0^{(t)}z_i}\right)\right) \\
&= \frac{-z_ie^{-\lambda_0^{(t)}z_i}}{1-e^{-\lambda_0^{(t)}z_i}} + \frac{1}{\lambda_0^{(t)}} \\
&= \frac{1}{\lambda_0^{(t)}} - \frac{z_i}{e^{+\lambda_0^{(t)}z_i}-1}
\end{aligned}$$

We can process similarly for the term with $\lambda_1^{(t)}$:

$$\mathbb{E}\left[y_i|y_i < z_i, \lambda_0^{(t)}\right] = \frac{1}{\lambda_1^{(t)}} - \frac{z_i}{e^{+\lambda_1^{(t)} z_i} - 1}$$

To sum up, we showed that:

$$\mathbb{E}\left[x_i|\mathbf{z}, \mathbf{u}, \lambda_0^{(t)}, \lambda_1^{(t)}\right] = u_i z_i + (1 - u_i)\left(\frac{1}{\lambda_0^{(t)}} - \frac{z_i}{e^{\lambda_0^{(t)} z_i} - 1}\right)$$
$$\mathbb{E}\left[y_i|\mathbf{z}, \mathbf{u}, \lambda_0^{(t)}, \lambda_1^{(t)}\right] = (1 - u_i) z_i + u_i\left(\frac{1}{\lambda_1^{(t)}} - \frac{z_i}{e^{\lambda_1^{(t)} z_i} - 1}\right)$$

Finally, we obtain the expression:

$$\begin{aligned}\mathbb{E}\left[\log f_{X,Y}(\mathbf{x}, \mathbf{y}|\lambda_0, \lambda_1)|\mathbf{z}, \mathbf{u}, \lambda_0^{(t)}, \lambda_1^{(t)}\right] &= n(\log\lambda_0 + \log\lambda_1) \\ &- \lambda_0 \sum_{i=1}^{n}\left[u_i z_i + (1 - u_i)\left(\frac{1}{\lambda_0^{(t)}} - \frac{z_i}{e^{\lambda_0^{(t)} z_i} - 1}\right)\right] \\ &- \lambda_1 \sum_{i=1}^{n}\left[(1 - u_i) z_i + u_i\left(\frac{1}{\lambda_1^{(t)}} - \frac{z_i}{e^{\lambda_1^{(t)} z_i} - 1}\right)\right]\end{aligned}$$

For the following part, we will denote:

$$\mathbb{E}\left[\log f_{X,Y}(\mathbf{x}, \mathbf{y}|\lambda_0, \lambda_1)|\mathbf{z}, \mathbf{u}, \lambda_0^{(t)}, \lambda_1^{(t)}\right] = n(\log\lambda_0 + \log\lambda_1) - \lambda_0 A_0(\mathbf{z}, \mathbf{u}, \lambda_0^{(t)}, \lambda_1^{(t)}) - \lambda_1 A_1(\mathbf{z}, \mathbf{u}, \lambda_0^{(t)}, \lambda_1^{(t)})$$

Where A_i are functions that don't depend on λ_j but $\lambda_j^{(t)}$.

b)

We want to use the EM algorithm to find a recursion in $(\lambda_0^{(t)}, \lambda_1^{(t)})$ for finding the MLE for (λ_0, λ_1).

Let $Q(\lambda_0, \lambda_1|\lambda_0^{(t)}, \lambda_1^{(t)}) := \mathbb{E}\left[\log f_{X,Y}(\mathbf{x}, \mathbf{y}|\lambda_0, \lambda_1)|\mathbf{z}, \mathbf{u}, \lambda_0^{(t)}, \lambda_1^{(t)}\right]$.

Algorithm: *Repeat until convergence*

1. E-step: Find $Q(\lambda|\lambda^{(t)})$
2. M-step: Set $\lambda \in \arg\max_\lambda Q(\lambda|\lambda^{(t)})$

Implementation:

1. We obtained the analytic expression for the E-step previously.
2. For the M-step, we have to solve an optimization problem. Since Q is $\mathcal{C}^1(\mathbb{R}_+^*)$, we can use the first order optimality condition. At first, we derive the gradient of Q with respect to λ_i

$$\frac{\partial Q}{\partial \lambda_i} = \frac{n}{\lambda_i} - A_i(\mathbf{z}, \mathbf{u}, \lambda_0^{(t)}, \lambda_1^{(t)}), \qquad \forall i \in \{0, 1\}$$

With the first order condition optimality,

$$\nabla Q(\lambda_0, \lambda_1) = \mathbf{0}_{2,1}$$
$$\implies \lambda_i^{(t+1)} = \frac{n}{A_i(\mathbf{z}, \mathbf{u}, \lambda_0^{(t)}, \lambda_1^{(t)})}, \qquad \forall i \in \{0, 1\}$$

The convergence criteria is: Stopping criteria for optimization problems are discussed are usually built upon: $\|Q(\lambda^{(t+1)}|\lambda^{(t)}) - Q(\lambda^{(t)}|\lambda^{(t)})\|_2$ or $\|\lambda^{(t+1)} - \lambda^{(t)}\|_2$. We choose the second option.

$$\|\lambda^{(t+1)} - \lambda^{(t)}\|_2 < \varepsilon$$

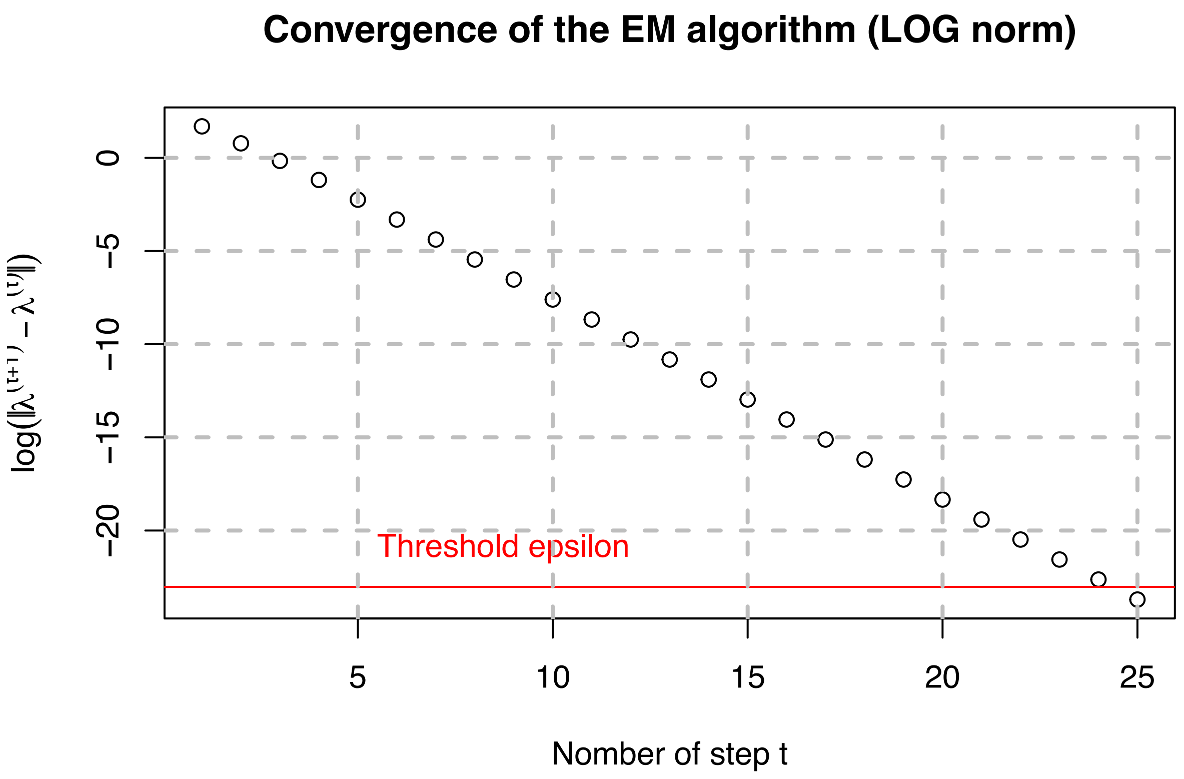
We choose $\varepsilon = 10^{-10}$ and a maximum number of iteration (to make sure that the function stops at some point) $N = 500$.

```r
# import file
u <- scan(file="https://www.math.ntnu.no/emner/TMA4300/2024v/u.txt")
z <- scan(file="https://www.math.ntnu.no/emner/TMA4300/2024v/z.txt")

# function that computes lambdas for (t+1) (dQ=0)
lambda_argmax_Q = function(lambda0T, lambda1T, u, z){
  n = length(u)
  lambda0 = n/sum(u*z + (1-u)*(1/lambda0T - z/(exp(lambda0T*z)-1)))
  lambda1 = n/sum((1-u)*z + u*(1/lambda1T - z/(exp(lambda1T*z)-1)))
  return(c(lambda0, lambda1))
}

# EM function, epsilon for stopping criteria and N max number of iteration
EM = function(u, z, epsilon, N){
  # initialize values
  lambda0T = c(1)
  lambda1T = c(1)
  norm = 1
  iter = 0
  convergence = c()
  # if stopping criteria is not
  while ((norm >= epsilon) && (iter < N)){
    iter = iter + 1
    # find argmax of Q
    lambdas = lambda_argmax_Q(tail(lambda0T,1),
                              tail(lambda1T,1),
                              u, z)
    # compute Euclidian norm between lambda (t) and (t+1)
    norm = norm(lambdas - c(tail(lambda0T,1), tail(lambda1T,1)), type = "2")

    # add to lists
    convergence = c(convergence, norm)
    lambda0T = c(lambda0T, lambdas[1])
    lambda1T = c(lambda1T, lambdas[2])
  }
  return(list(lambdas=lambdas, convergence=convergence, allLambda0=lambda0T, allLambda1=lambda1T))
}

# parameters for optimization (stooping criteria)
epsilon = 10^(-10)
N = 500
result = EM(u,z, epsilon, N)
cat("Lambda 0 =",result$lambdas[1], "Lambda 1 =",result$lambdas[2])
```

```
## Lambda 0 = 3.465735 Lambda 1 = 9.353215
```

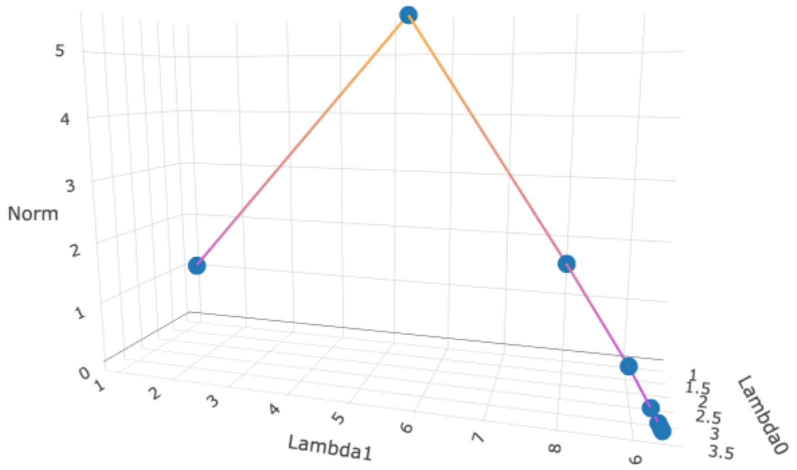
We can visualize the convergence of the algorithm in a plot: $\|\lambda^{(t+1)} - \lambda^{(t)}\|_2 = f(t)$

```r
plot(log(result$convergence), main="Convergence of the EM algorithm (LOG norm)",
     xlab="Nomber of step t", ylab=expression(log(abs(abs(lambda^{(t+1)}-lambda^{(t)})))))
abline(h = log(epsilon), col = "red")
text(x = 5, y = -21, labels = "Threshold epsilon", pos = 4, col = "red")
grid(nx = NULL, ny = NULL,
     lty = 2, col = "gray", lwd = 2)
```



Visualisation of the EM algorithm, step by step: $\|\lambda^{(t+1)} - \lambda^{(t)}\|_2 = f(\lambda_0^{(t)}, \lambda_1^{(t)})$.

```r
library(plotly)

Lambda0 = result$allLambda0
Lambda1 = result$allLambda1
Norm = c(1,result$convergence)

plot_ly(x = ~Lambda0, y = ~Lambda1, z = ~Norm, type = 'scatter3d', mode = 'lines+markers',
             line = list(width = 4, color = ~Norm, colorscale = list(c(0,'#BA52ED'), c(1,'#FCB040'))))
```



c)

We want to use bootstrapping to estimate the standard deviations and the biases of each of $\widehat{\lambda}_i$ and to estimate their correlation. We decided to use non parametric bootstrapping: we used the data that we already have $(\mathbf{z}, \mathbf{u})$ instead of parametric boostrapping, where we would draw from $x_i^{b*} \sim \mathcal{E}(\lambda_0)$ (and $y_i^{b*} \sim \mathcal{E}(\lambda_1)$). We consider that it's more relevant to trust the observed data (non parametric bootstrapping) instead of relying on hypothesis under distribution of x_i, y_j.

Algorithm 1: Non parametric Bootstrapping

Input : B: number of bootstrap samples
Output: θ: Matrix of size $(B, 2)$ lambda for each bootstrap

```
1 for $b \in [1, B]$ do
2     Sample $\mathbf{z}^{b*}$ from $\mathbf{z}$
3     Sample $\mathbf{u}^{b*}$ from $\mathbf{u}$ (same index as for $\mathbf{z}$)
4     $\lambda^b = EM(\mathbf{z}^{b*}, \mathbf{u}^{b*})$
5     $\theta_{b,:} \leftarrow \lambda^b$
6 end
```

```r
# Non parametric Bootstrapping
Bootstrap = function(B){
  theta = matrix(nrow = B, ncol=2)
  for (b in 1:B){
    # choice index for sample, with replacement
    index = sample.int(length(z), replace = TRUE)
    z_star = z[index]
    u_star = u[index]

    # EM-algorithm
    result = EM(u_star, z_star, epsilon, N)

    # add lambdas to the theta matrix
    theta[b, ] = result$lambdas

  }
  return(theta)
}
```

```r
# run the bootstrapping
theta = Bootstrap(B=1000)

mean_theta <- colMeans(theta)
bias_theta <- mean_theta - result$lambdas
sd_theta <- c(sd(theta[, 1]), sd(theta[, 2]))
correlation_theta <- cor(theta)[1, 2]
```

```
##                  Metric            Lambda0           Lambda1
## 1
## 2                  Mean   3.48995373173714  9.45747929117133
## 3                 Bias: 0.0242187632035282  0.10426436234332
## 4 Standard Deviation:  0.243097773785075 0.767519229414149
## 5          Correlation: 0.0128244411139459
```

Result review:

- Bias: There are slightly positive.
- Standard Deviation: $\sigma \sim 10^{-1}$ low enough, give a small 95 % confidence interval for parameter estimation.
- Correlation: $\rho \approx 0^{-}$ expected since $X \perp\!\!\!\perp Y \implies Cov(X, Y) = 0$.

We would prefer to go for a bias-corrected estimate because we are interested in the true values of the parameters λ_i.

In this example the maximum likelihood estimate of λ from the observed data can be determined from elementary analytic methods without reliance on any fancy numerical optimization strategy like EM. That is what we will show in the next part.

d)

Let's find an analytical formula for $f_{Z_i,U_i}(z_i, u_i|\lambda_0, \lambda_1)$.

$$\begin{aligned}
f_{Z_i,U_i}(z_i, u_i|\lambda_0, \lambda_1) &= \mathbb{P}[Z_i = z_i, U_i = u_i|\lambda_0, \lambda_1] \\
&= \mathbb{P}[\max(X_i, Y_i) = z_i, I(X_i \geq Y_i) = u_i|\lambda_0, \lambda_1, X_i, Y_i] \\
&= \sum_{u_i \in \{0,1\}} I(U_i = u_i)\mathbb{P}[\max(X_i, Y_i) = z_i, U_i = u_i|\lambda_0, \lambda_1, X_i, Y_i] \\
&= u_i\mathbb{P}[\max(X_i, Y_i) = z_i, U_i = 1|\lambda_0, \lambda_1, X_i, Y_i] + (1u_i)\mathbb{P}[\max(X_i, Y_i) = z_i, U_i = 0|\lambda_0, \lambda_1, X_i, Y_i] \\
&= u_i\mathbb{P}[X_i = z_i, X_i \geq Y_i|\lambda_0, \lambda_1, X_i, Y_i] + (1 - u_i)\mathbb{P}[Y_i = z_i, X_i < Y_i|\lambda_0, \lambda_1, X_i, Y_i] \\
&= u_i\mathbb{P}[X_i \geq Y_i|\lambda_0\lambda_1, X_i = z_i, Y_i]\mathbb{P}[X_i = z_i|, \lambda_0, \lambda_1, X_i, Y_i] \\
&\qquad + (1 - u_i)\mathbb{P}[X_i < Y_i|\lambda_0, \lambda_1, X_i, Y_i = z_i]\mathbb{P}[Y_i = z_i|, \lambda_0, \lambda_1, X_i, Y_i] \quad \text{with Bayes} \\
&= u_i\mathbb{P}[Y_i \leq z_i|\lambda_1]\mathbb{P}[X_i = z_i|\lambda_0] + (1 - u_i)\mathbb{P}[X_i < z_i|\lambda_0]\mathbb{P}[Z_i = z_i|\lambda_1] \\
&= u_i(1 - e^{-\lambda_1 z_i})\lambda_0 e^{-\lambda_0 z_i} + (1 - u_i)(1 - e^{-\lambda_0 z_i})\lambda_1 e^{-\lambda_1 z_i}
\end{aligned}$$

It is also possible to find analytical formulas for the maximum likelihood estimators $\widehat{\lambda}_i$.

$$\begin{aligned}
\log f_{Z,U}(z,u|\lambda_0,\lambda_1) &= \log \prod_{i=1}^{n} f_{Z_i,U_i}(z_i,u_i|\lambda_0,\lambda_1) \qquad \text{by independence} \\
&= \sum_{i=1}^{n} \log f_{Z_i,U_i}(z_i,u_i|\lambda_0,\lambda_1) \\
&= \sum_{i=1|u_i=1}^{n} \log f_{Z_i,U_i}(z_i,1|\lambda_0,\lambda_1) + \sum_{i=1|u_i=0}^{n} \log f_{Z_i,U_i}(z_i,0|\lambda_0,\lambda_1) \\
&= \sum_{i=1|u_i=1}^{n} \big(\log u_i + \log(1-e^{-\lambda_1 z_i}) + \log\lambda_0 - \lambda_0 z_i\big) \\
&\qquad + \sum_{i=1|u_i=0}^{n} \big(\log(1-u_i) + \log(1-e^{-\lambda_0 z_i}) + \log\lambda_1 - \lambda_1 z_i\big) \\
&= \sum_{i=1|u_i=1}^{n} \big(\log(1-e^{-\lambda_1 z_i}) + \log\lambda_0 - \lambda_0 z_i\big) + \sum_{i=1|u_i=0}^{n} \big(\log(1-e^{-\lambda_0 z_i}) + \log\lambda_1 - \lambda_1 z_i\big)
\end{aligned}$$

With `optim` we can find numerically $\widehat{\lambda}_i$.

```r
loglikelihood = function(lambdas, z, u){
  ll = 0
  for (i in 1:length(u)){
    if (u[i] == 1){
      ll = ll + log(1-exp(-lambdas[2]*z[i])) - lambdas[1]*z[i]
    }
    else{
      ll = ll + log(1-exp(-lambdas[1]*z[i])) - lambdas[2]*z[i]
    }
  }
  ll = ll + sum(u)*log(lambdas[1]) + sum(1-u)*log(lambdas[2])
  # return the negative log likelihood
  return(-ll)
}
lambdas = optim(c(1, 1), loglikelihood, z=z, u=u)
cat('Lambda 1 =', lambdas$par[1], 'Lambda 2 =', lambdas$par[2])
```

```
## Lambda 1 = 3.46589 Lambda 2 = 9.351103
```

```r
cat(lambdas$par-result$lambdas)
```

```
## 0.0001550136 -0.002111513
```

The advantages of optimizing the likelihood directly compared to the EM algorithm are:

- Easier to implement
- Generally more efficient
- Only one optimization, while as many as time step are needed for EM algorithm (optimized iteratively)
- Quicker
- Do not dependent on the amount of missing data
- More features with optimization methods (hessian…)

However, when we want to maximize difficult likelihood, there are no analytic formula, then the EM algorithm is the solution.