TMA4300 - Project 2

Jakob Heide and Celine Olsson

2024-03-11

```r
load(file = "rain.rda")
library(boot)
library(matrixStats)
library(EfficientMaxEigenpair) #for tridiag setup
library(MASS)
library("INLA")
```

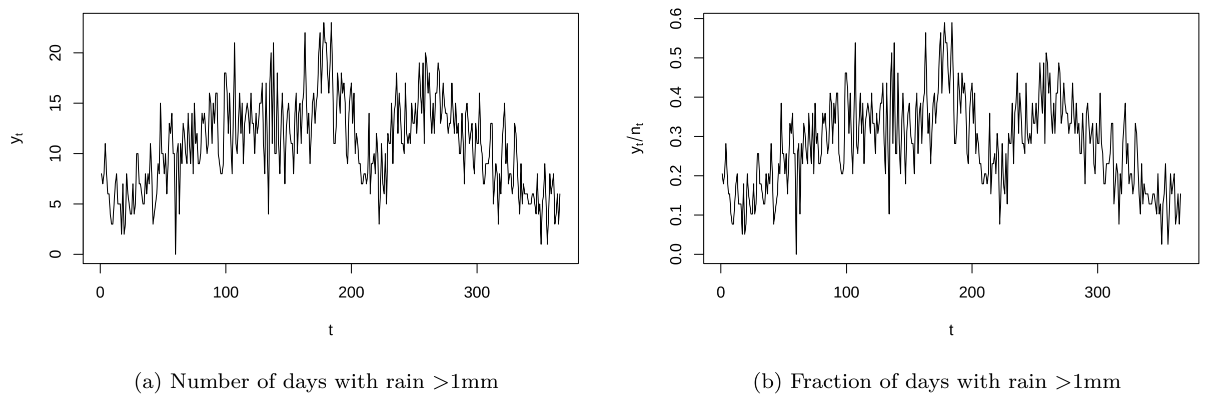
Problem 1

In this project, we consider a portion of the Tokyo rainfall dataset. The response is whether the amount of rainfall on a given day exceeded 1mm, where the rain has been measured every day in the years 1951 - 1989. That is,

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

for $n_{60} = 10$ and $n_t = 39$, for $t = 1, \dots, 59, 61, \dots, 366$. We assume conditional independence in $y_t|x_t$, for $t = 1, \dots, 366$. The plot of the data can be seen in Figure 1.

```r
plot(rain$day, rain$n.rain, type="l", ylab=expression(y[t]), xlab="t")
plot(rain$day, rain$n.rain/rain$n.years, type="l", ylab=expression(y[t]/n[t]), xlab="t")
```


(a) Number of days with rain >1mm

(b) Fraction of days with rain >1mm

Figure 1: Plot of data

The plot shows that the amount of rainfall tends to increase towards $t \approx 190$, with one period with less rain at approximately $t = 220$. If we ignore the trends, the data looks like a random walk.

1b)

Let $\boldsymbol{y} = (y_1, \ldots, y_T)^T$, $\boldsymbol{x} = (x_1, \ldots, x_T)^T$ and $\boldsymbol{\pi} = (\pi(x_1), \ldots, \pi(x_T))^T$. We use the definition of the likelihood to obtain:

$$L(\boldsymbol{\pi}|\boldsymbol{y}, \boldsymbol{x}) = \prod_{t=1}^{366} f(y_t|x_t; n_t, \pi(x_t)) = \prod_{t=1}^{366} \binom{n_t}{y_t|x_t} \pi(x_t)^{y_t|x_t}(1-\pi(x_t))^{n_t - y_t|x_t}$$

$$= \prod_{t=1}^{366} \binom{n_t}{y_t|x_t} \left(\frac{1}{1+\exp(-x_t)}\right)^{y_t|x_t} \left(\frac{\exp(-x_t)}{1+\exp(-x_t)}\right)^{n_t - y_t|x_t} \tag{1}$$

1c)

A Bayesian hierarchical model will be applied to the dataset. A random walk of order 1 will be used to model the trend on a logit scale:

$$x_t = x_{t-1} + u_t \tag{2}$$

where $u_t \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma_u^2)$, such that

$$p(\boldsymbol{x}|\sigma_u^2) \propto \prod_{t=2}^{T} \frac{1}{\sigma_u} \exp\left(-\frac{1}{2\sigma_u^2}(x_t - x_{t-1})^2\right) \tag{3}$$

The following inverse gamma prior will be placed on σ_u^2,

$$p(\sigma_u^2) = \frac{\beta^\alpha}{\Gamma(\alpha)} \left(\frac{1}{\sigma_u^2}\right)^{\alpha+1} \exp\left(-\frac{\beta}{\sigma_u^2}\right) \tag{4}$$

with shape α and scale β.

Now we want to find the conditional $p(\sigma_u^2|\boldsymbol{y}, \boldsymbol{x})$. A conditional probability can be written as the joint probability divided by the marginal probability:

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

This is used to get the following result:

$$\begin{aligned} p(\sigma_u^2|\boldsymbol{y}, \boldsymbol{x}) &= \frac{p(\boldsymbol{y}, \boldsymbol{x}, \sigma_u^2)}{p(\boldsymbol{y}, \boldsymbol{x})} = \frac{p(\boldsymbol{y}|\sigma_u^2, \boldsymbol{x}) \cdot p(\sigma_u^2, \boldsymbol{x})}{p(\boldsymbol{y}, \boldsymbol{x})} \\ &= \frac{p(\boldsymbol{y}|\sigma_u^2, \boldsymbol{x}) \cdot p(\boldsymbol{x}|\sigma_u^2) \cdot p(\sigma_u^2)}{p(\boldsymbol{y}, \boldsymbol{x})} = \frac{p(\boldsymbol{x}|\sigma_u^2) \cdot p(\sigma_u^2)}{p(\boldsymbol{x})} \\ &\propto p(\boldsymbol{x}|\sigma_u^2) \cdot p(\sigma_u^2) \end{aligned}$$

where $p(\boldsymbol{y}|\sigma_u^2, \boldsymbol{x}) = p(\boldsymbol{y}|\boldsymbol{x})$ since $\boldsymbol{y}$ is not dependent on σ_u^2. Both $p(\boldsymbol{x}|\sigma_u^2)$ and $p(\sigma_u^2)$ are known from Equations (3) and (4). Plugging these in gives the following probability mass function.

$$\begin{aligned} p(\boldsymbol{x}|\sigma_u^2) \cdot p(\sigma_u^2) &\propto \frac{\beta^\alpha}{\Gamma(\alpha)} \left(\frac{1}{\sigma_u^2}\right)^{\alpha+1} \exp\left(\frac{-\beta}{\sigma_u^2}\right) \prod_{t=2}^{T} \frac{1}{\sigma_u} \exp(x_t - x_{t-1})^2 \\ &= \frac{\beta^\alpha}{\Gamma(\alpha)} \left(\frac{1}{\sigma_u^2}\right)^{\alpha + \frac{T+1}{2}} \exp\left(-\frac{1}{\sigma_u^2}\left(\beta + \frac{1}{2}\sum_{t=2}^{T}(x_t - x_{t-1})^2\right)\right) \end{aligned}$$

This is proportional to a inverse gamma distribution with $\alpha^* = \alpha + \frac{T-1}{2}$ and $\beta^* = \beta + \frac{1}{2}\sum_{t=2}^{T}(x_t - x_{t-1})^2$.

1d)

We partition the vector $\boldsymbol{x}$ into the sets $\boldsymbol{x}_{\mathcal{I}}, \boldsymbol{x}_{-\mathcal{I}}$ for some $\mathcal{I} \subseteq \{1, \dots, 366\}$. Let $Q(\boldsymbol{x}'_{\mathcal{I}}|\boldsymbol{x}_{-\mathcal{I}}, \sigma_u^2, \boldsymbol{y}) = p(\boldsymbol{x}'_{\mathcal{I}}|\boldsymbol{x}_{-\mathcal{I}}, \sigma_u^2)$ be the conditional prior proposal distribution in the Hastings algorithm. The acceptance probability of a step in the algorithm is given by

$$\alpha(\boldsymbol{x}'_{\mathcal{I}}|\boldsymbol{x}_{-\mathcal{I}}, \sigma_u^2, \boldsymbol{y}) = \min\left(1, \frac{\pi(\boldsymbol{x}'_{\mathcal{I}}|\boldsymbol{x}_{-\mathcal{I}}, \sigma_u^2, \boldsymbol{y}_{\mathcal{I}}) \cdot p(\boldsymbol{x}_{\mathcal{I}}|\boldsymbol{x}_{-\mathcal{I}}, \sigma_u^2)}{\pi(\boldsymbol{x}_{\mathcal{I}}|\boldsymbol{x}_{-\mathcal{I}}, \sigma_u^2, \boldsymbol{y}_{\mathcal{I}}) \cdot p(\boldsymbol{x}'_{\mathcal{I}}|\boldsymbol{x}_{-\mathcal{I}}, \sigma_u^2)}\right)$$

Using the product rule, we can write the limiting distribution function $\pi(\boldsymbol{x}'_{\mathcal{I}}|\boldsymbol{x}_{-\mathcal{I}}, \sigma_u^2, \boldsymbol{y}_{\mathcal{I}})$ as

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

and similarly for the limiting distribution function $\pi(\boldsymbol{x}_{\mathcal{I}}|\boldsymbol{x}_{-\mathcal{I}}, \sigma_u^2, \boldsymbol{y}_{\mathcal{I}})$. Inserting these, we find the acceptance probability

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

The product rule is again used on both expressions in the first fraction, which gives the following.

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

Again equal terms can be canceled out, which leaves us with the following expression.

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

The last equality comes from the fact that $\boldsymbol{y}_{\mathcal{I}}$ does not depend on $\boldsymbol{x}_{-\mathcal{I}}$ and σ_u^2.

1e

Equation (3) can be rewritten as

$$p(\boldsymbol{x}|\sigma_u^2) \propto \exp\left\{-\frac{1}{2}\boldsymbol{x}^T\mathbf{Q}\boldsymbol{x}\right\} \tag{6}$$

where the precision matrix $\mathbf{Q}$ is defined as

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

The components of $\boldsymbol{x}$ is now partitioned into two subvectors, so that

$$\boldsymbol{x} = \begin{pmatrix} \boldsymbol{x}_A \\ \boldsymbol{x}_B \end{pmatrix}. \tag{7}$$

The precision matrix is partitioned in the same way giving us

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

Now we want to find the distribution of $\boldsymbol{x}_A$ conditional on $\boldsymbol{x}_B$. First the new partition in Equation (7) is inserted into Equation (6) as shown below.

$$\begin{aligned} p(\boldsymbol{x}_A|\boldsymbol{x}_B) &\propto p(\boldsymbol{x}) \\ &\propto \exp\left\{-\frac{1}{2}\begin{pmatrix}\boldsymbol{x}_A^T & \boldsymbol{x}_B^T\end{pmatrix}\begin{pmatrix}\mathbf{Q}_{AA} & \mathbf{Q}_{AB} \\ \mathbf{Q}_{BA} & \mathbf{Q}_{BB}\end{pmatrix}\begin{pmatrix}\boldsymbol{x}_A \\ \boldsymbol{x}_B\end{pmatrix}\right\} \\ &\propto \exp\left\{-\frac{1}{2}(\boldsymbol{x}_A^T\mathbf{Q}_{AA}\boldsymbol{x}_A + \boldsymbol{x}_A^T\mathbf{Q}_{AB}\boldsymbol{x}_B + \boldsymbol{x}_B^T\mathbf{Q}_{BA}\boldsymbol{x}_A)\right\} \end{aligned} \tag{8}$$

In the last step, all the expressions not dependent on $\boldsymbol{x}_A$ are removed.

We want an expression on the form

$$\begin{aligned} &\exp\left(-\frac{1}{2}\left((\boldsymbol{x}_A - \mu_{A|B})^T\mathbf{Q}_{A|B}(\boldsymbol{x}_A - \mu_{A|B})\right)\right) \\ \propto &\exp\left(-\frac{1}{2}\left(\boldsymbol{x}_A^T\mathbf{Q}_{A|B}\boldsymbol{x}_A - \mu_{A|B}^T\mathbf{Q}_{A|B}\boldsymbol{x}_A - \boldsymbol{x}_B^T\mathbf{Q}_{A|B}\mu_{A|B}\right)\right) \end{aligned} \tag{9}$$

Equating the coefficients in Equation (8) and Equation (9), we obtain the following results:

$$\begin{aligned} \mathbf{Q}_{A|B} &= \mathbf{Q}_{AA}, \\ -\mathbf{Q}_{A|B}\mu_{A|B} = \mathbf{Q}_{AB}\boldsymbol{x}_B \implies \mu_{A|B} &= -\mathbf{Q}_{AA}^{-1}\mathbf{Q}_{AB}\boldsymbol{x}_B \end{aligned}$$

Hence, the conditional $\boldsymbol{x}_A|\boldsymbol{x}_B$ follows a Gaussian distribution with expected value $\mu_{A|B} = -\mathbf{Q}_{AA}^{-1}\mathbf{Q}_{AB}\boldsymbol{x}_B$ and precision matrix $\mathbf{Q}_{A|B} = \mathbf{Q}_{AA}$.

1f)

We note that if the partitioning of $\boldsymbol{x} = (x_1, \ldots, x_T)^T$ in Equation (7) is such that $\boldsymbol{x}_\mathcal{I}$ is only one element, then the mean in the conditional Gaussian distribution becomes

$$\mu_{\mathcal{I}|-\mathcal{I}} = \begin{cases} x_2, & \mathcal{I} = 1 \\ (x_{\mathcal{I}-1} + x_{\mathcal{I}+1})/2, & \mathcal{I} \in \{2, \ldots, T-1\} \\ x_{T-1}, & \mathcal{I} = T \end{cases}$$

and the variance becomes

$$\Sigma_{\mathcal{I}|-\mathcal{I}} = \begin{cases} \sigma_u^2, & \mathcal{I} = 1 \\ \sigma_u^2/2, & \mathcal{I} \in \{2, \ldots, T-1\} \\ \sigma_u^2, & \mathcal{I} = T \end{cases}$$

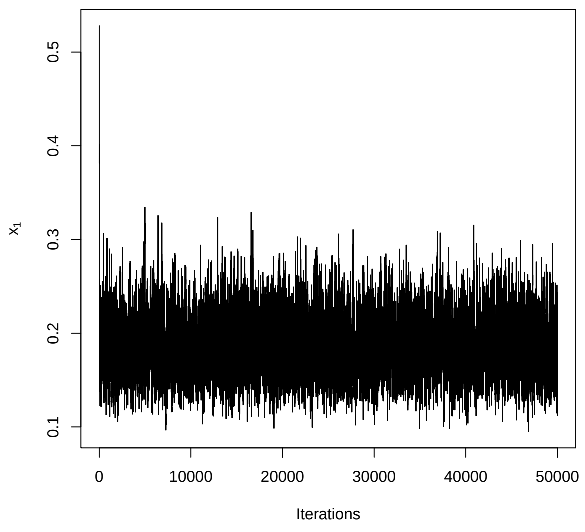
In addition, we note that the ratio of likelihoods found in Equation 5 can be rewritten as

$$\begin{aligned} \frac{p(\boldsymbol{y}_t|\boldsymbol{x}_t')}{p(\boldsymbol{y}_t|\boldsymbol{x}_t)} &= \frac{\exp(\sum_t \ln(\binom{n_t}{y_t|x_t'})) - n_t x_t' + y_t x_t' - n_t \ln(1 + \exp(-x_t'))}{\exp(\sum_t \ln(\binom{n_t}{y_t|x_t})) - n_t x_t + y_t x_t - n_t \ln(1 + \exp(-x_t))} \\ &= \exp\left(\sum_t (n_t - y_t)(x_t - x_t') + n_t \ln\left(\frac{1 + \exp(-x_t)}{1 + \exp(-x_t')}\right)\right), \end{aligned}$$

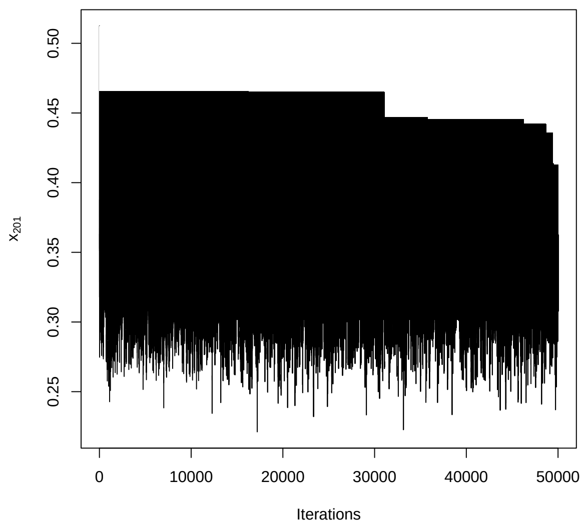
which we will exploit in the following implementation. We implement a MCMC sampler with a Metropolis-Hastings step for the individual x_t-parameters and Gibbs steps for σ_u^2. The prior on σ_u^2 uses $\alpha = 2$ and $\beta = 0.05$ as parameters, and we perform $N = 50000$ iterations.

```r
sample_sigma <- function(x){
  #Function to sample sigma^2 from the inverse gamma
  #distribution. Uses Gibbs sampling (acceptance probability 1)
  alpha <- 2
  beta <- 0.05

  a <- alpha + (length(x)-1)/2
  b <- beta + 0.5*sum(diff(x)^2)

  return (1/rgamma(1,shape = a,rate = b))
}

sample_proposal <- function(x,idx,sigma){
  if (idx == 1){
    return (rnorm(1,mean = x[idx+1],sd = sqrt(sigma)))
  }
  else if (idx == length(x)){
    return (rnorm(1,mean = x[idx-1],sd = sqrt(sigma)))
  }
  else{
    return (rnorm(1,mean = (x[idx-1]+x[idx+1])*0.5,sd = sqrt(sigma/2)))
  }
}

loglik_ratio <- function(x,prop,y,n){
  #Ratio of the log-likelihoods of the binomial distribution..
  return ((n-y) * (x-prop) + n * log((1+exp(-x))/(1+exp(-prop))))
}

sampler <- function(iter){
  #MCMC sampler.
  t <- proc.time()[3]
  end <- 366
  x <- rnorm(end)

  states <- matrix(NA, nrow = iter,ncol = end)
  states[1,] <- x
  sigmas <- numeric(iter)
  sigma <- 0.05
  sigmas[1] <- sigma

  acceptance_rate <- 0
  for (i in 2:iter){
    unifs <- runif(end)

    for (j in 1:end){ #Could be implemented using vectors only
      proposals <- sample_proposal(x,j,sigma)

      accept <- min(1,exp(loglik_ratio(x[j],proposals,rain$n.rain[j],rain$n.years[j])))

      if (unifs[j] < accept){ #Change state
        x[j] <- proposals
        states[i,j] <- proposals
      }
      else{ #Keep previous state
        x[j] <- states[i-1,j]
        states[i,j] <- states[i-1,j]
      }
      acceptance_rate <- acceptance_rate + accept
    }
    x <- states[i,]
    sigma <- sample_sigma(x)
    sigmas[i] <- sigma
  }

  return (list(x = states,sigma = sigmas,
               runtime = proc.time()[3]-t,
               accept_rate = acceptance_rate/(iter*end)))
}
```

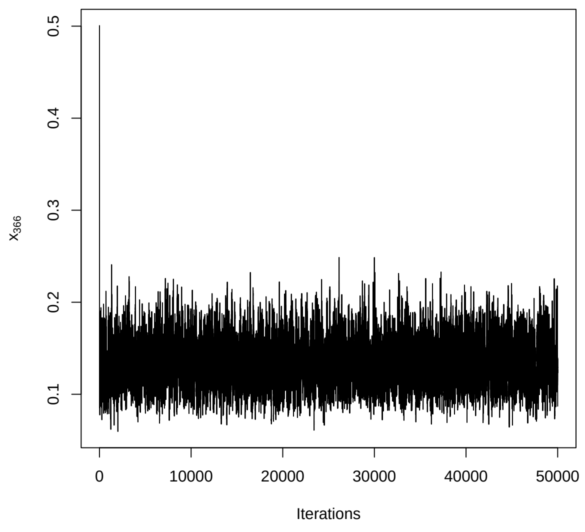
```r
N = 50000

MC <- sampler(N)

pi_1 <- inv.logit(MC$x[,1])
pi_201 <- inv.logit(MC$x[,201])
pi_366 <- inv.logit(MC$x[,366])
#Traceplots

plot(1:N,pi_1,type = "l",xlab = "Iterations", ylab = expression(x[1]))
plot(1:N,pi_201,type = "l",xlab = "Iterations", ylab = expression(x[201]))
plot(1:N,pi_366,type = "l",xlab = "Iterations", ylab = expression(x[366]))
plot(1:N,MC$sigma,type = "l",xlab = "Iterations", ylab = expression(sigma[u]^2))
```



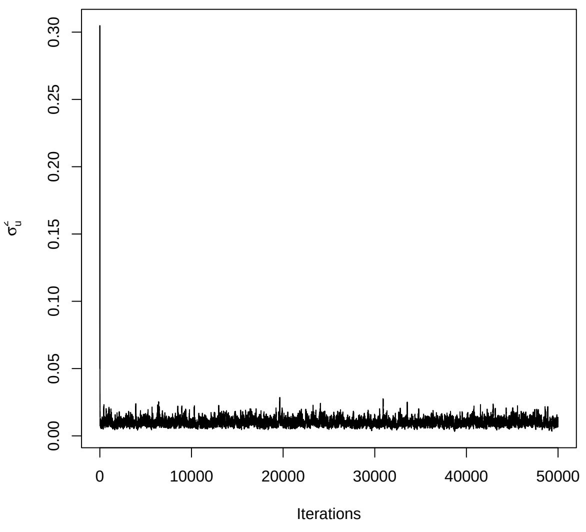
(a)



(b)



(c)

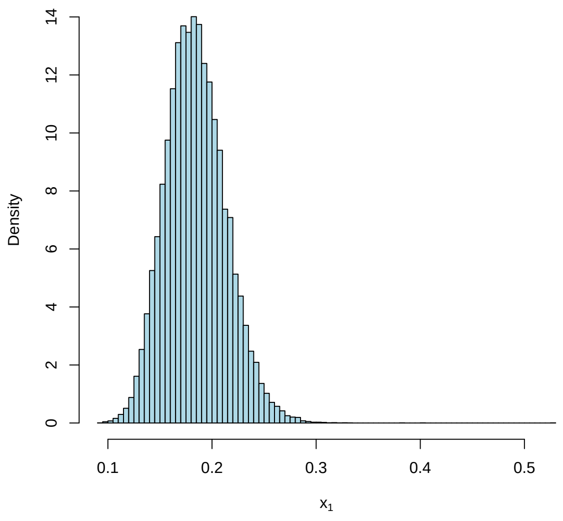


(d)

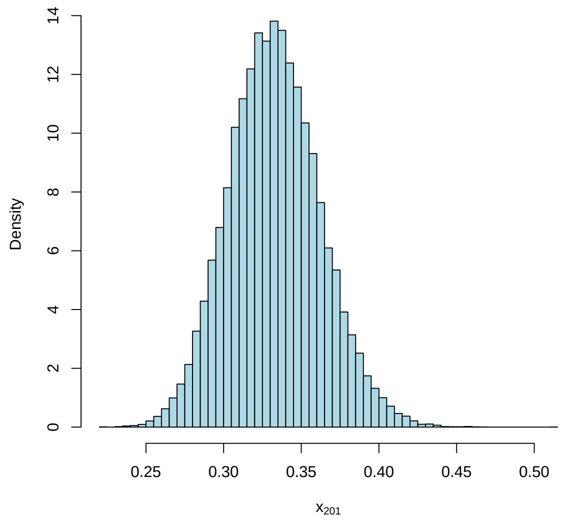
Figure 2: Traceplots

All the traceplots in Figure 2 indicate that the Markov chain has converged. In fact, the Markov chains seem to converge very fast, which leads us to assume that a burn-in period of 1000 iterations should be more than sufficient.

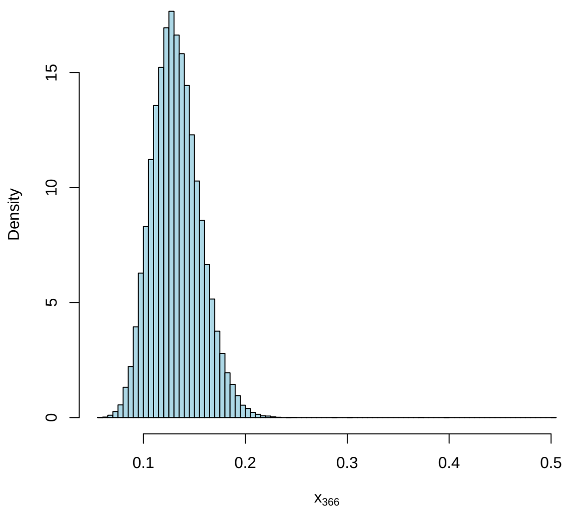
```
hist(pi_1,freq = F,breaks = 100,xlab = expression(x[1]),main = "",col = "lightblue")
hist(pi_201,freq = F,breaks = 100,xlab = expression(x[201]),main = "",col = "lightblue")
hist(pi_366,freq = F,breaks = 100,xlab = expression(x[366]),main = "",col = "lightblue")
hist(MC$sigma,freq = F,breaks = 1000,xlab = expression(sigma[u]^2),main = "",
    col = "lightblue",xlim = c(0,0.05))
```



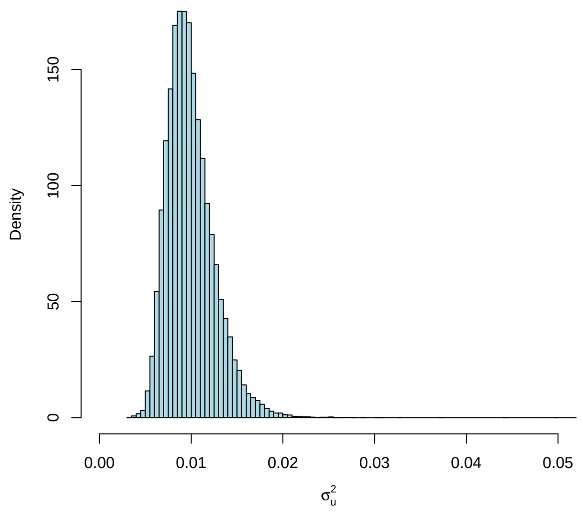
(a)



(b)



(c)

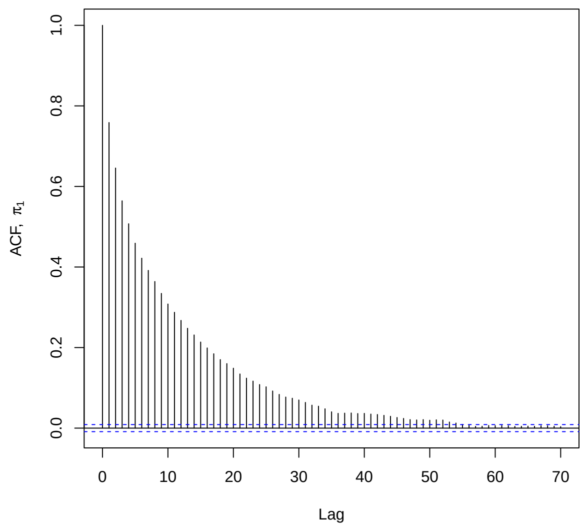


(d)

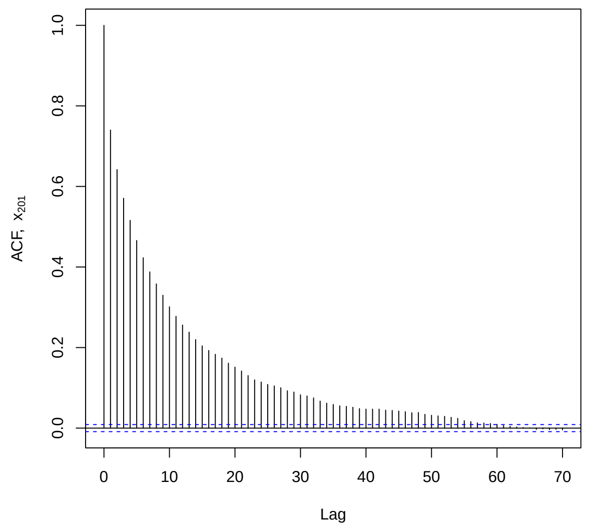
Figure 3: Histograms

In Figure 3, we see the histograms, i.e. the estimated posterior distributions for $\pi(x_1)$ in (a), $\pi(x_{201})$ in (b), $\pi(x_{366})$ in (c) and σ_u^2 in (d).

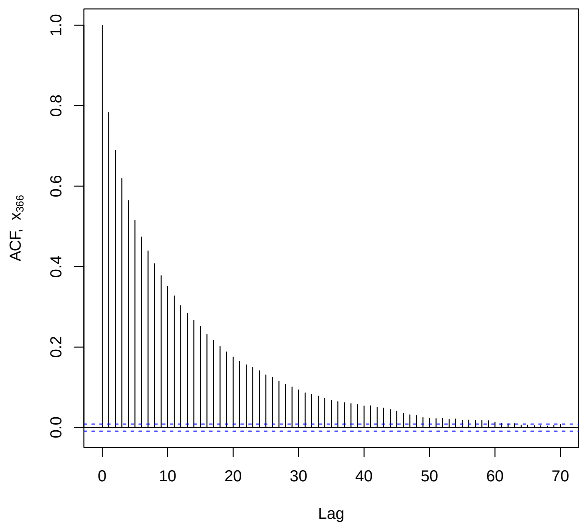
```r
acf(pi_1,lag.max = 70,main = "",ylab = expression("ACF, "~pi[1]))
acf(pi_201,lag.max = 70,main = "",ylab = expression("ACF, "~x[201]))
acf(pi_366,lag.max = 70,main = "",ylab = expression("ACF, "~x[366]))
acf(MC$sigma,lag.max = 80,main = "",ylab = expression("ACF, "~sigma[u]^2))
```



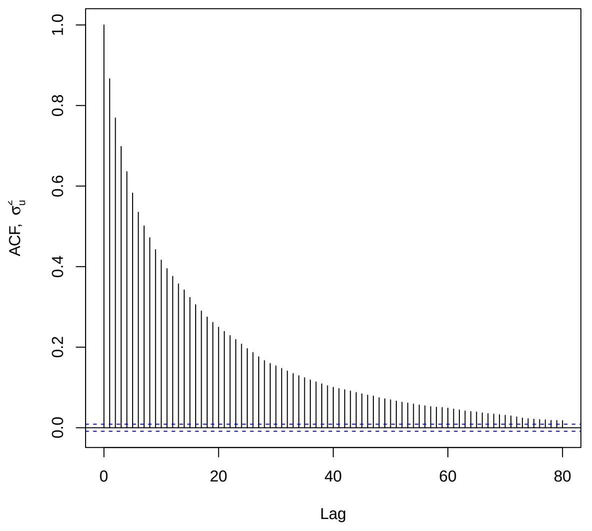
(a)



(b)



(c)



(d)

Figure 4: Autocorrelation plots

In Figure 4, we see the autocorrelation function for each of the chosen parameters. After lag 80, all the plots show that there are no more significant spikes. We can interpret this as an indication that the Markov chain has converged, however the ACF plot should be interpreted in relation to other diagnostics as well.

```
cat("The procedure used ", MC$runtime, " seconds, and had an average acceptance probability of "
    , MC$accept_rate, "%. \n")
```

```
## The procedure used  126.5  seconds, and had an average acceptance probability of  0.9176563 %.
```

We calculate the central estimates and 95% credible intervals.

```
sigma <- c(mean(MC$sigma), quantile(MC$sigma, probs=c(0.025, 0.975)))
pi_1 <- c(mean(pi_1), quantile(pi_1, probs=c(0.025, 0.975)))
pi_201 <- c(mean(pi_201), quantile(pi_201, probs=c(0.025, 0.975)))
pi_366 <- c(mean(pi_366), quantile(pi_366, probs=c(0.025, 0.975)))

df <- data.frame(sigma=sigma, pi_1=pi_1, pi_201=pi_201, pi_366=pi_366)
rownames(df) <- c("Mean", "2.5%", "97.5%")
print(t(df))
```

```
##                Mean        2.5%      97.5%
## sigma  0.009987912 0.006085843 0.01595639
## pi_1   0.184645733 0.133062916 0.24499151
## pi_201 0.332838470 0.277864785 0.39325947
## pi_366 0.132250693 0.090958118 0.18213753
```

We calculate the posterior mean $\pi(x_t)$ as a function of t, with 95% credible limits, and plot this against the observed y_t/n_t. We use a burn-in period of 1000 iterations.

```
CI <- function(x){
  pi <- inv.logit(x)
  means <- colMeans(pi)
  ci <- colQuantiles(pi, probs = c(0.025,0.975))  # returns 95% interval
  return(list(mean=means, ci=ci))
}


burn <- 1000
sample <- MC$x[burn:N,]
MC_pi <- CI(sample)

plot(rain$day, rain$n.rain/rain$n.years, type="l", col="darkgray", ylim=c(0,0.8),
     ylab=expression(pi(x[t])), xlab="t", cex.axis=1.2, cex.lab=1.5)
lines(rain$day, MC_pi$ci[,1], col="red", lty=1, lwd=2)
lines(rain$day, MC_pi$ci[,2], col="red", lty=1, lwd=2)
lines(rain$day, MC_pi$mean, col="blue", lwd=2)
legend("topright", inset=0.01, legend=c("Observed", "MCMC", "90% CI - MCMC"),
       col=c("darkgray", "blue", "red"), lwd=2, box.lty = 0)
```

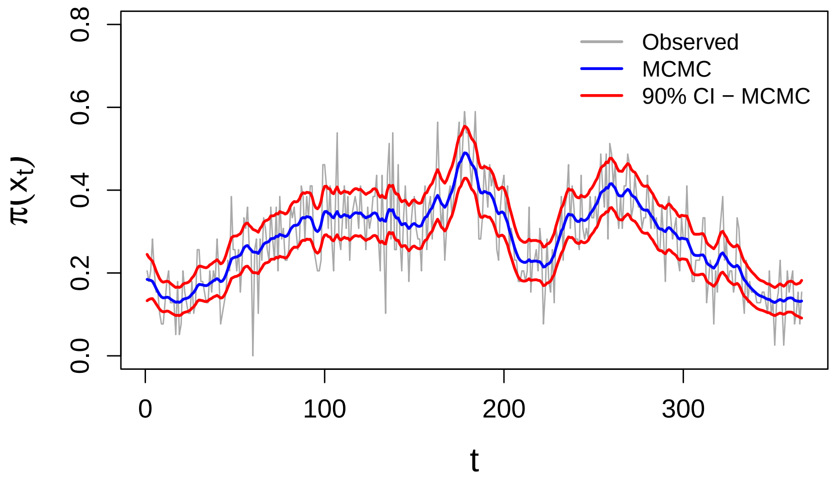


Figure 5: The posterior mean with credible intervals and the data.

In Figure 5, we see that the MCMC algorithm captures the trend in the data well. The posterior mean $\pi(x_t)$ has a smoothing effect on the data, and most of the data points lie within the 95% credible interval.

1g)

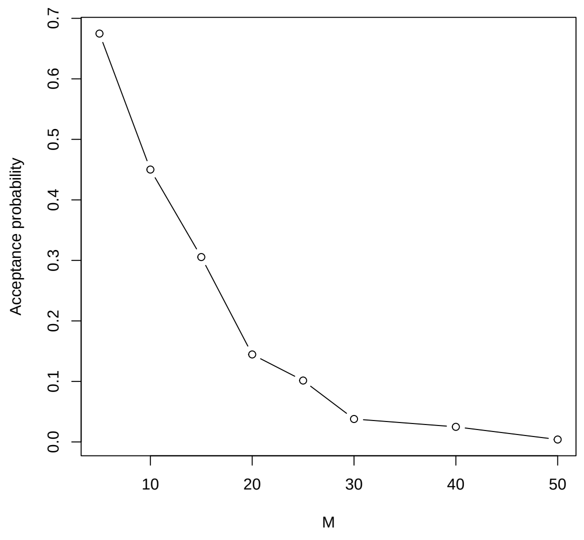
We repeat the previous exercise, but this time we allow for the partitioning of $\boldsymbol{x}$ into $\boldsymbol{x}_{\mathcal{I}}$ to contain more than one element, that is, we use the conditional prior proposal $p(\boldsymbol{x}_{(a,b)}|\boldsymbol{x}_{-(a,b)}, \sigma_u^2)$, where (a,b) defines an interval of length M. The main difference here is that we can no longer use any shortcut in the calculation of the mean and variance of the proposal distribution. The implementation of the block step-MCMC sampler is shown below.

```
sampler_block <- function(M,iter){
  t <- proc.time()[3]
  end <- 366
  Q <- tridiag(rep(-1,(end-1)),rep(-1,(end-1)),c(1,rep(2,end-2),1))

  last_block_size <- end %% M
  number_of_blocks <- floor(366/M)

  Q_AA_start <- Q[1:M,1:M]
  Q_AA_start_inv <- solve(Q_AA_start)
  Q_AA <- Q[2:(M+1),2:(M+1)]
  Q_AA_inv <- solve(Q_AA)
  Q_AA_end <- Q[(end-last_block_size+1):end,(end-last_block_size+1):end]
  Q_AA_end_inv <- solve(Q_AA_end)

  Q_AB_start <- Q[1:M,(M+1):end]
  Q_AB_end <- Q[(end-last_block_size+1):end,1:(end-last_block_size)]

  x <- rnorm(end)
```

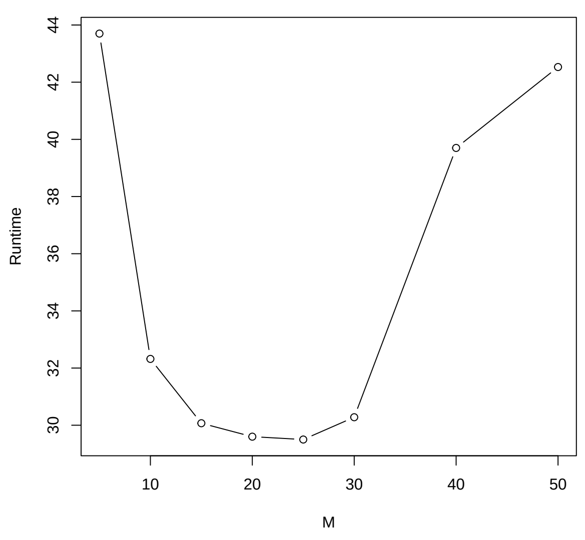
```r
states <- matrix(NA, nrow = iter,ncol = end)
states[1,] <- x
sigmas <- numeric(iter-1)
acceptance_rate <- 0

for (i in 2:iter){
  sigma <- sample_sigma(x)
  sigmas[i-1] <- sigma
  unifs <- runif(number_of_blocks+1)

  #Handle first case
  proposals <- mvrnorm(1,mu = -Q_AA_start_inv %*% Q_AB_start %*%
                         x[-(1:M)], Sigma = Q_AA_start_inv*sigma)
  ratio <- exp(sum(loglik_ratio(x[1:M],proposals,
                                rain$n.rain[1:M],rain$n.years[1:M])))
  accept_samples <- unifs[1] < min(1,ratio)
  acceptance_rate <- acceptance_rate + accept_samples*M
  states[i,1:M] <- proposals*accept_samples + states[i-1,1:M]*!accept_samples
  x[1:M] <- states[i,1:M]

  for (j in 1:(number_of_blocks-1)){ #Handle middle cases
    proposals <- mvrnorm(1,mu = -Q_AA_inv %*%
                           Q[(j*M+1):(j*M+M),c(1:(j*M),(j*M+M+1):end)] %*%
                           x[-((j*M+1):(j*M+M))],
                         Sigma = Q_AA_inv*sigma)
    ratio <- exp(sum(loglik_ratio(x[(j*M+1):(j*M+M)],proposals,
                                  rain$n.rain[(j*M+1):(j*M+M)],
                                  rain$n.years[(j*M+1):(j*M+M)])))
    accept_samples <- unifs[j+1] < min(1,ratio)
    acceptance_rate <- acceptance_rate + accept_samples*M
    states[i,(j*M+1):(j*M+M)] <- proposals*accept_samples +
      states[i-1,(j*M+1):(j*M+M)]*!accept_samples
    x[(j*M+1):(j*M+M)] <- states[i,(j*M+1):(j*M+M)]
  }

  #Handle last case
  proposals <- mvrnorm(1, mu = -Q_AA_end_inv %*% Q_AB_end %*%
                         x[1:(end-last_block_size)],
                       Sigma = Q_AA_end_inv*sigma)
  ratio <- exp(sum(loglik_ratio(x[(end-last_block_size+1):end],proposals,
                                rain$n.rain[(end-last_block_size+1):end],
                                rain$n.years[(end-last_block_size+1):end])))
  accept_samples <- unifs[number_of_blocks+1] < min(1,ratio)
  acceptance_rate <- acceptance_rate + last_block_size*accept_samples

  states[i,(end-last_block_size+1):end] <- proposals*accept_samples +
    states[i-1,(end-last_block_size+1):end]*!accept_samples

  x[(end-last_block_size+1):end] <- states[i,(end-last_block_size+1):end]

}

return (list(x = states,sigma = sigmas,runtime = proc.time()[3] - t,
               accept_rate = acceptance_rate/(iter*end)))
}
```

We must make a choice of M, the size of each block step. We therefore run the algorithm for different values of M, and calculate the runtime and average acceptance probability of each run. Note that we here use only $N = 5000$ iterations due to runtime issues, so some of the Markov Chains may not converge.

```
#Test different values of M
M = c(5,10,15,20,25,30,40,50) #For the final run

accept_vals <- numeric(length = length(M))
runtime <- numeric(length = length(M))
a <- numeric(length = length(M))
for (i in 1:length(M)){
  MC_block <- sampler_block(M[i],5000)
  accept_vals[i] <- MC_block$accept_rate
  runtime[i] <- MC_block$runtime
}

plot(M,accept_vals,ylab = "Acceptance probability",type = "b")
plot(M,runtime,ylab = "Runtime",type = "b")
```



(a)



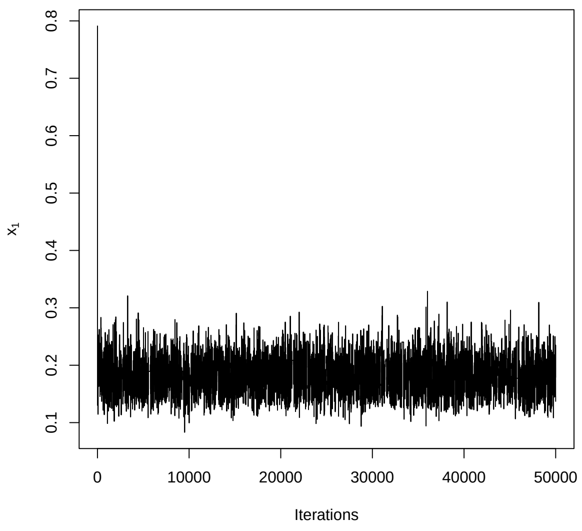
(b)

Figure 6: Acceptance probability (a) and runtime (b) as functions of the block size M.

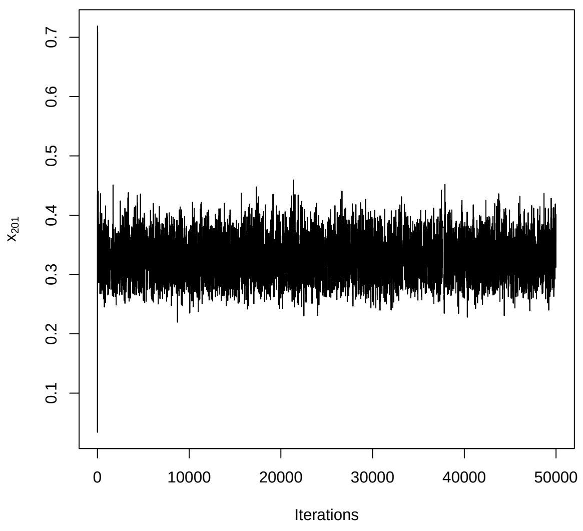
In Figure 6We see that the acceptance probability decreases with increasing M, and that the runtime decreases with increasing M up to about $M = 25$, where the runtime starts to increase. We therefore use $M = 15$, which has an approximate acceptance probability of 0.29.

Below, we run the algorithm with $M = 15$ for $N = 50000$ iterations.

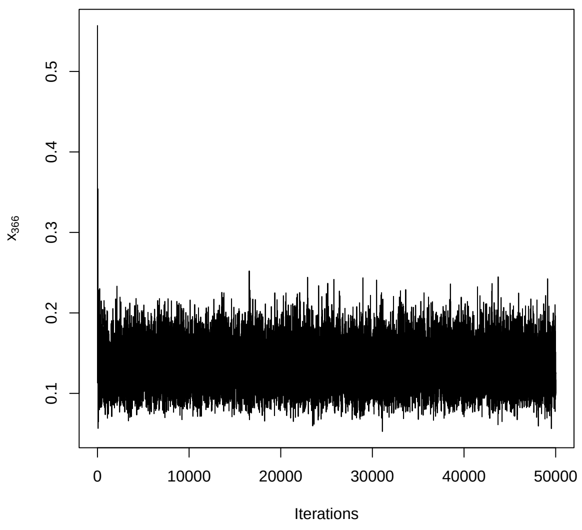
```r
#Choose M = 15
M <- 15
N <- 50000
MC_block <- sampler_block(M,N)

pi_1 <- inv.logit(MC_block$x[,1])
pi_201 <- inv.logit(MC_block$x[,201])
pi_366 <- inv.logit(MC_block$x[,366])

#Traceplots
plot(1:N,pi_1,type = "l",xlab = "Iterations", ylab = expression(x[1]))
plot(1:N,pi_201,type = "l",xlab = "Iterations", ylab = expression(x[201]))
plot(1:N,pi_366,type = "l",xlab = "Iterations", ylab = expression(x[366]))
plot(1:(N-1),MC_block$sigma,type = "l",xlab = "Iterations", ylab = expression(sigma[u]^2),ylim = c(0,0.
```



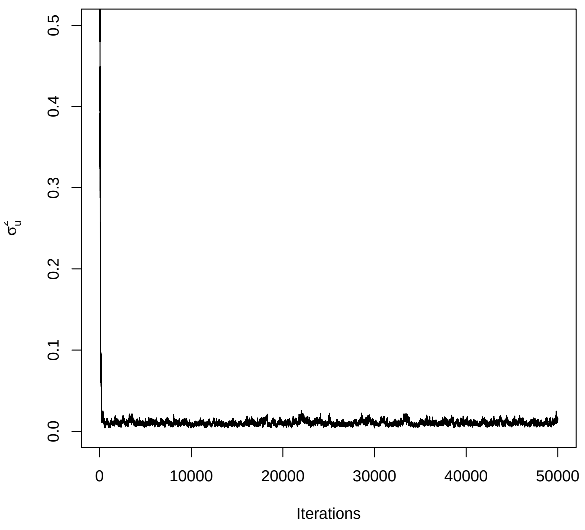
(a)



(b)



(c)

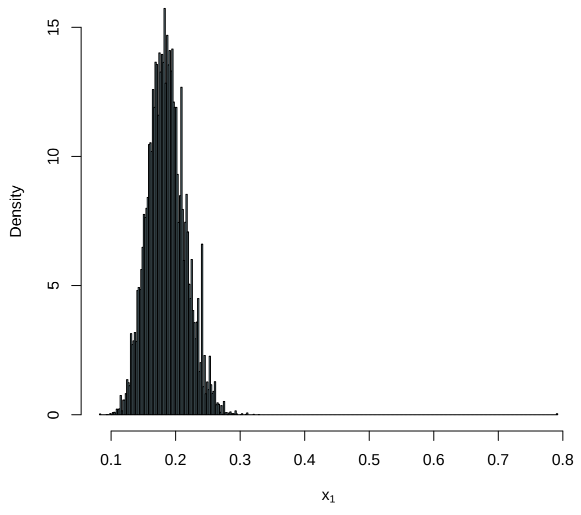


(d)

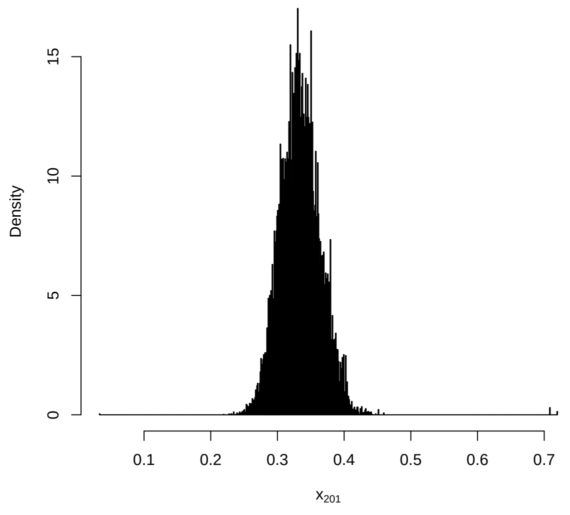
Figure 7: Traceplots

The traceplots in Figure 7 indicate that the chosen components of $\boldsymbol{x}$ and σ_u^2 have converged.

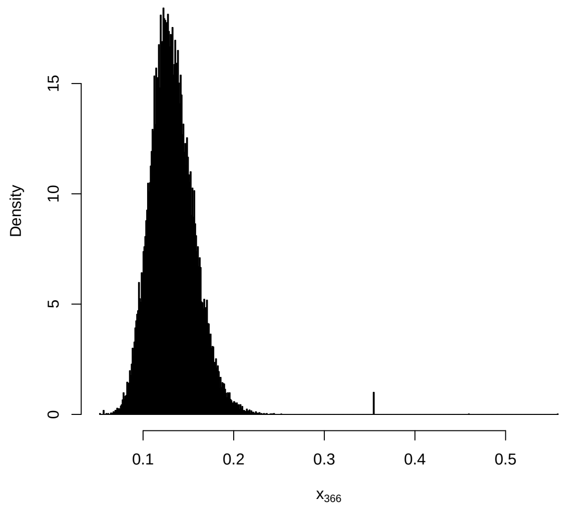
```
hist(pi_1,freq = F,breaks = 500,xlab = expression(x[1]),main = "",col = "lightblue")
hist(pi_201,freq = F,breaks = 500,xlab = expression(x[201]),main = "",col = "lightblue")
hist(pi_366,freq = F,breaks = 500,xlab = expression(x[366]),main = "",col = "lightblue")
hist(MC_block$sigma,freq = F,breaks = 1000,xlab = expression(sigma[u]^2),main = "",col = "lightblue",xl
```



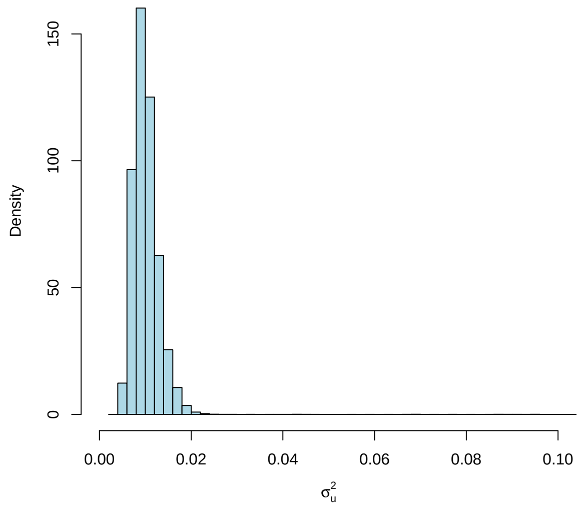
(a)



(b)



(c)

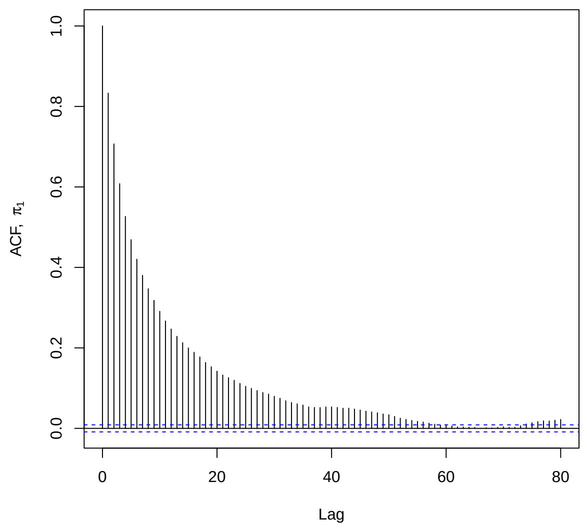


(d)

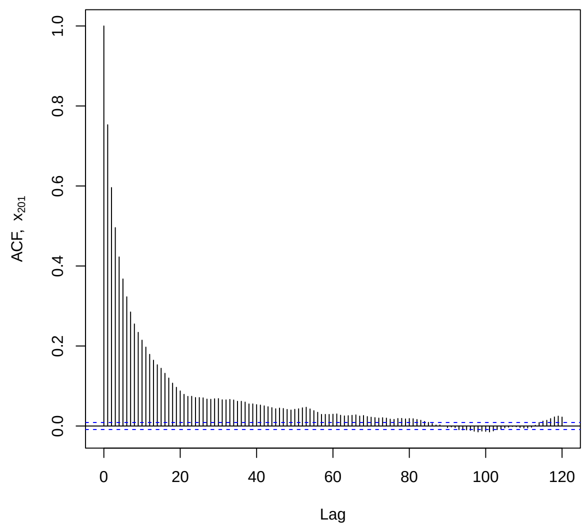
Figure 8: Histograms

In Figure 8, we see the posterior distributions.

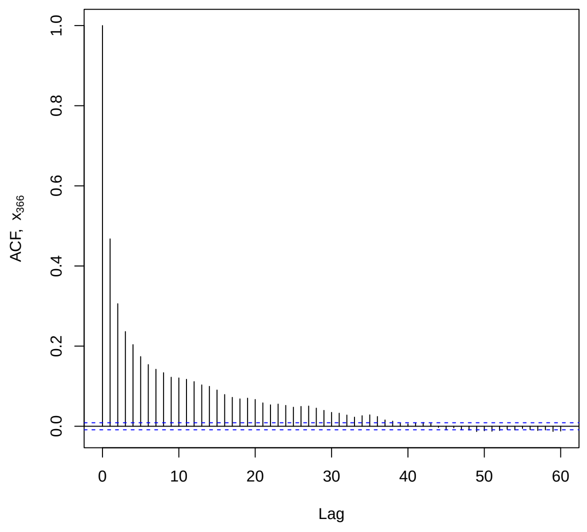
```r
acf(pi_1,lag.max = 80,main = "",ylab = expression("ACF, "~pi[1]))
acf(pi_201,lag.max = 120,main = "",ylab = expression("ACF, "~x[201]))
acf(pi_366,lag.max = 60,main = "",ylab = expression("ACF, "~x[366]))
acf(MC_block$sigma,lag.max = 200,main = "",ylab = expression("ACF, "~sigma[u]^2))
```



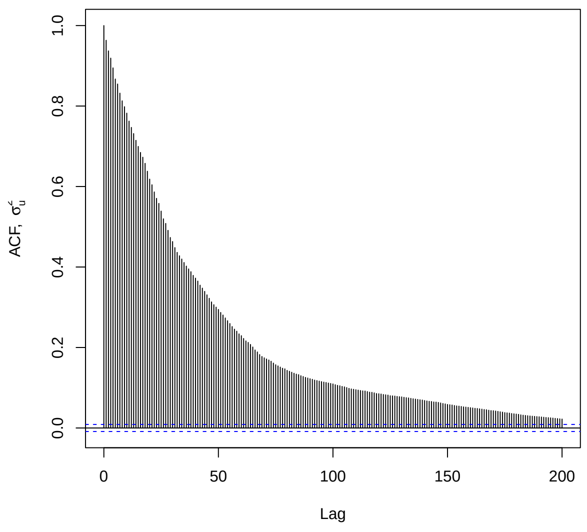
(a)



(b)



(c)



(d)

Figure 9: Autocorrelation plots

In Figure 9, we see that the autocorrelation becomes insignificant after 200 iterations, which indicates that a burn-in period of 1000 iterations should again be enough.

```
cat("The procedure used ", MC_block$runtime, " seconds, and had an average acceptance probability of ",
```

```
## The procedure used  291.83  seconds, and had an average acceptance probability of  0.3125854 %.
```

We calculate the central estimates and 95% credible intervals for $\pi(x_1), \pi(x_{201}), \pi(x_{366})$, and σ_u^2.

```
sigma <- c(mean(MC_block$sigma), quantile(MC_block$sigma, probs=c(0.025, 0.975)))
pi_1 <- c(mean(pi_1), quantile(pi_1, probs=c(0.025, 0.975)))
pi_201 <- c(mean(pi_201), quantile(pi_201, probs=c(0.025, 0.975)))
pi_366 <- c(mean(pi_366), quantile(pi_366, probs=c(0.025, 0.975)))

df <- data.frame(sigma=sigma, pi_1=pi_1, pi_201=pi_201, pi_366=pi_366)
rownames(df) <- c("Mean", "2.5%", "97.5%")
print(t(df))
```

```
##               Mean        2.5%      97.5%
## sigma  0.01132305 0.006006425 0.01672773
## pi_1   0.18615369 0.133379230 0.24615484
## pi_201 0.33371907 0.278638620 0.39382064
## pi_366 0.13251253 0.090791403 0.18316244
```

We calculate the posterior mean $\pi(x_t)$ as a function of t, with 95% credible limits. This can be seen in Figure 10 along with the observed y_t/n_t.

```
burn <- 1000
sample <- MC_block$x[burn:N,]
MC_block_pi <- CI(sample)

plot(rain$day, rain$n.rain/rain$n.years, type="l", col="darkgray", ylim=c(0,0.8),
     ylab=expression(pi(x[t])), xlab="t", cex.axis=1.2, cex.lab=1.5)
lines(rain$day, MC_block_pi$ci[,1], col="red", lty=1, lwd=2)
lines(rain$day, MC_block_pi$ci[,2], col="red", lty=1, lwd=2)
lines(rain$day, MC_block_pi$mean, col="blue", lwd=2)
legend("topright", inset=0.01, legend=c("Observed", "MCMC", "90% CI - MCMC"),
       col=c("darkgray", "blue", "red"), lwd=2, box.lty = 0)
```

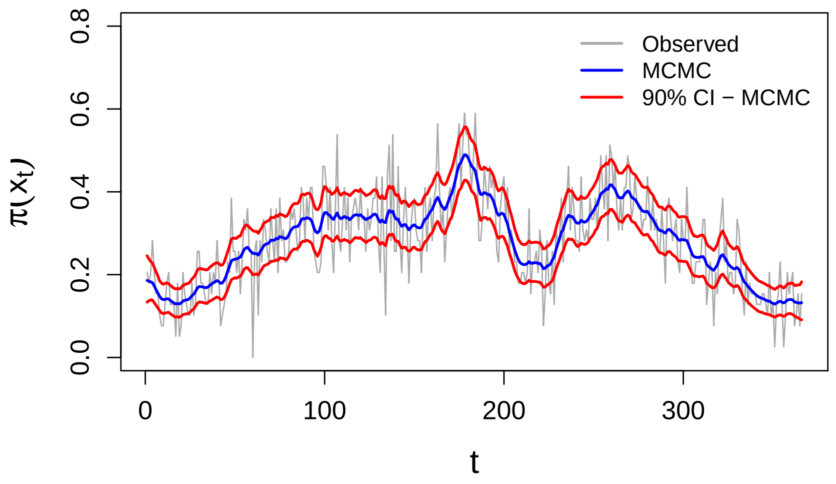


Figure 10: The posterior mean with credible intervals and the data.

Problem 2

For this part INLA is used instead of MCMC. We want to fit the same model as in Problem 1, but by using INLA instead we expect a much faster computation time.

```
#opt <- options()
#options(pkgType="both")
#options(opt)  ## switch back to old options
#install.packages("INLA",repos=c(getOption("repos"),
#  INLA="https://inla.r-inla-download.org/R/stable"), type="binary", dep=TRUE)
```

2a)

First we have to make sure the same model is fitted as when using MCMC. The INLA method places a prior on the logarithmic precision rather than the variance. This gives us the following hyperparameter

$$\theta = \log\left(\frac{1}{\sigma_u^2}\right)$$

Since σ_u^2 has a inverse gamma distribution, the hyperparameter θ gets the following distribution

$$\theta \sim \text{log-gamma}(\alpha, \beta)$$

where $\alpha = 2$ and $\beta = 0.05$. Simplified Laplace is used as the approximation strategy, and `ccd` is used as the integration strategy. The intercept is also removed in this model. The INLA model is fitted below, and we refer to it later as Model 1.

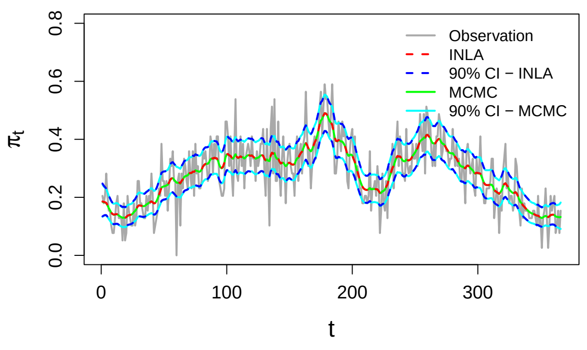
```
control.inla = list(strategy="simplified.laplace", int.strategy="ccd")

alpha <- 2
beta <- 0.05

time <- proc.time()[3]
mod1 <- inla(n.rain ~ -1 + f(day, model="rw1", constr=FALSE,
                             hyper=list(prec=list(prior="loggamma", param=c(alpha,beta))) ),
            data=rain, Ntrials=n.years, control.compute=list(config = TRUE),
            family="binomial", verbose=TRUE, control.inla=control.inla)
time <- proc.time()[3] - time
cat("Computational time: ", time)
```

```
## Computational time:  0.97
```

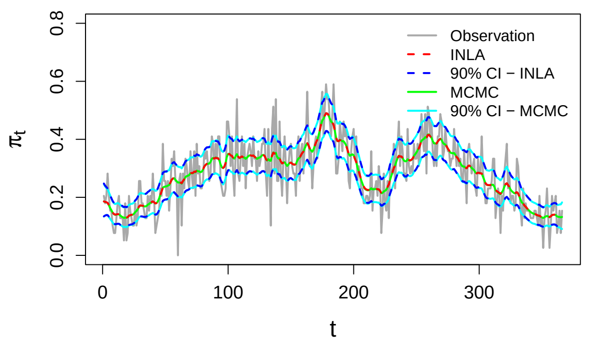
We see from the computation time, this method is far more efficient than the one used in Problem 1. But what about the results? Next the INLA model is compared to the MCMC models created before. Figure 11 shows the mean prediction of $\boldsymbol{\pi}$ and a 95% credible interval for both the MCMC models compared to the INLA model.

```
# Plotting INLA vs MC
par(pty="m")
# Data
plot(rain$day, rain$n.rain/rain$n.years, type="l", col="darkgrey", lwd=2,
     xlab="t", ylab=expression(pi[t]), ylim=c(0,0.8), cex.axis=1.2, cex.lab=1.5)
```

```r
# MCMC from 1f)
lines(rain$day, MC_pi$mean, col="green", lwd=2)
lines(rain$day, MC_pi$ci[,1], col="cyan", lwd=2)
lines(rain$day, MC_pi$ci[,2], col="cyan", lwd=2)
#INLA
lines(mod1$summary.fitted.values$mean, col="red", lwd=2, lty=2)
lines(mod1$summary.fitted.values$`0.025quant`, col="blue", lwd=2, lty=2)
lines(mod1$summary.fitted.values$`0.975quant`, col="blue", lwd=2, lty=2)
legend("topright", inset=0.01,
       legend=c("Observation", "INLA", "90% CI - INLA", "MCMC", "90% CI - MCMC"),
       col=c("darkgray", "red", "blue", "green", "cyan"), lty=c(1,2,2,1,1),
       lwd=2, box.lty = 0)



# Plotting INLA vs MC_block
par(pty="m")
# Data
plot(rain$day, rain$n.rain/rain$n.years, type="l", col="darkgrey", lwd=2,
     xlab="t", ylab=expression(pi[t]), ylim=c(0,0.8), cex.axis=1.2, cex.lab=1.5)
# MCMC from 1g)
lines(rain$day, MC_block_pi$mean, col="green", lwd=2)
lines(rain$day, MC_block_pi$ci[,1], col="cyan", lwd=2)
lines(rain$day, MC_block_pi$ci[,2], col="cyan", lwd=2)
#INLA
lines(mod1$summary.fitted.values$mean, col="red", lwd=2, lty=2)
lines(mod1$summary.fitted.values$`0.025quant`, col="blue", lwd=2, lty=2)
lines(mod1$summary.fitted.values$`0.975quant`, col="blue", lwd=2, lty=2)
legend("topright", inset=0.01,
       legend=c("Observation", "INLA", "90% CI - INLA", "MCMC", "90% CI - MCMC"),
       col=c("darkgray", "red", "blue", "green", "cyan"), lty=c(1,2,2,1,1),
       lwd=2, box.lty = 0)
```



(a) INLA vs MCMC



(b) INLA vs block-MCMC

Figure 11: Comparing INLA to MCMC

Looking at the plots the methods are almost indistinguishable from each other. Below the estimated σ_u^2 value given by the INLA method is printed.

```
sigma_inla <- 1/(mod1$summary.hyperpar$mean)
print(sigma_inla)
```

```
## [1] 0.009389275
```

This is ~ 0.0007 away from the estimate given from the MCMC methods. This means the major difference between the methods is the run time, which is much faster when using INLA. INLA would therefor be the preferred method to use.

2b)

Now to test the robustness of the INLA method. This is done by checking all the possible approximation and integration strategies, and comparing the results. The approximation strategies tested is `gaussian`, `simplified.laplace`, `laplace` and `adaptive`. The integration strategies tested are `ccd`, `grid` and `eb`. The plot of all the different combinations can be seen in Figure 12.

```
test_control <- function(){
  alpha=2
  beta=0.05
  strats <- c("gaussian", "simplified.laplace", "laplace", "adaptive")
  int.strats <- c("ccd", "grid", "eb")

  times <- matrix(nrow=length(strats), ncol=length(int.strats))
  colnames(times) <- int.strats
  rownames(times) <- strats

  plot(rain$day, rain$n.rain/rain$n.years, type="l", col="white", xlab="t",
       ylab=expression(pi[t]), ylim=c(0,0.6), cex.axis=1.2, cex.lab=1.5, lwd=2.5)

  for (i in 1:length(strats)) {
    for (j in 1:length(int.strats)) {
      control.inla = list(strategy=strats[i], int.strategy=int.strats[j])
      time <- proc.time()[3]
      mod <- inla(n.rain ~ -1 + f(day, model="rw1", constr=FALSE,
                                  hyper=list(prec=list(prior="loggamma", param=c(alpha,beta))) ),
                  data=rain, Ntrials=n.years, control.compute=list(config = TRUE),
                  family="binomial", verbose=TRUE, control.inla=control.inla)
      times[i,j] <-  proc.time()[3] - time

      lines(mod$summary.fitted.values$mean, col=i*j, lwd=2)
    }
  }
  return(times)
}

test <- test_control()
```

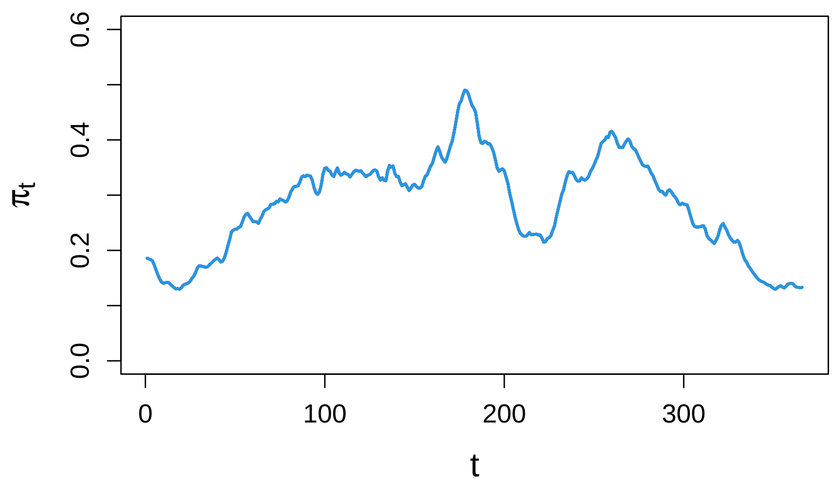


Figure 12: Comparing different approximation and integration strategies using INLA

The simulations are so similar they are indistinguishable by eye. This indicates that the INLA method is very robust regardless of strategy.

```
print(test)
```

```
##                       ccd grid   eb
## gaussian             0.77 0.85 0.80
## simplified.laplace 0.69 0.95 0.71
## laplace              0.78 0.99 0.70
## adaptive             0.67 0.85 0.71
```

The run time for all the different combinations can be seen above. There is no considerable difference between the run times.

Taking into account both the plots of the simulations and the fast run times, the INLA method can be considered very robust when considering the `control.inla` parameters.

2c)

Now a new INLA model is considered. The intercept is now kept and the `constr` parameter in `f` is now set to be `TRUE`. When including the intercept the response is now given by

$$y_t|\tau_t \sim \text{Bin}(n_t, \pi(\tau_t))$$

where $\tau_t = \beta_0 + x_t$ and β_0 is the intercept. Setting `constr=TRUE` means there is now a sum to zero constraint. This model will be referred to as Model 2. Model 1 and Model 2 can be seen in Figure 13

```
control.inla = list(strategy="simplified.laplace", int.strategy="ccd")
mod2 <- inla(n.rain ~ f(day, model="rw1", constr=TRUE,
                            hyper=list(prec=list(prior="loggamma", param=c(alpha,beta)))),
             data=rain, Ntrials=n.years, control.compute=list(config = TRUE),
```

```r
              family="binomial", verbose=TRUE,control.inla=control.inla)

plot(rain$day, rain$n.rain/rain$n.years, type="l", col="darkgray", lwd=2,
     xlab="t", ylab=expression(pi[t]), ylim=c(0,0.7), cex.axis=1.2, cex.lab=1.5)
lines(rain$day, mod1$summary.fitted.values$mean, col="red", lwd=2.5)
lines(rain$day, mod2$summary.fitted.values$mean, col="blue", lwd=2.5)
legend("topright", inset=0.01, legend=c("Observation","Model 1", "Model 2"),
       col=c("darkgray","red", "blue"), lwd=2, box.lty = 0)
```

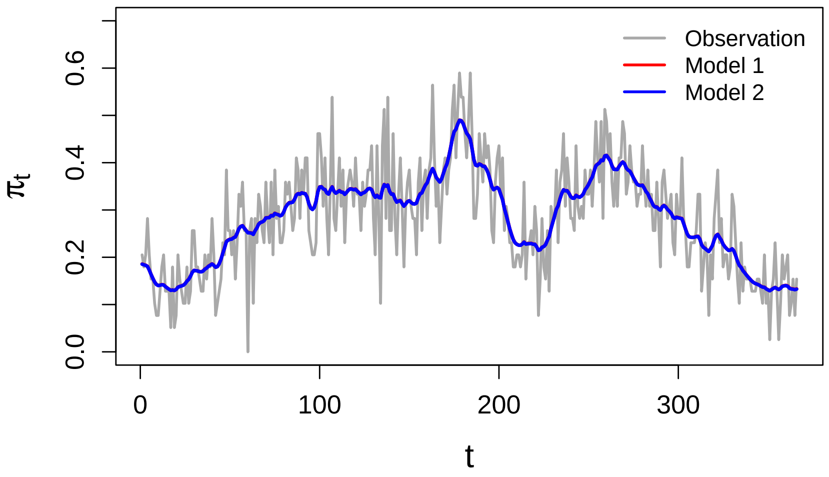


Figure 13: Comparing Model 1 (without intercept and sum to zero contraint) to Model 2 (with intercept and sum to zero constraint).

Looking at the plot is does not seem to be any major difference between the two models.

To conclude, using INLA is more efficient than the MCMC.

Problem 3

We implement the same model as in part 1 and 2, only now using the R-package `RTMB`. We begin by computing the joint likelihood $\pi(\boldsymbol{y}, \boldsymbol{x}|\sigma_u^2)$, which we can write as

$$\pi(\boldsymbol{y}, \boldsymbol{x}|\sigma_u^2) = p(\boldsymbol{y}|\boldsymbol{x}) \cdot p(\boldsymbol{x}|\sigma_u^2),$$

where we have seen the terms on the right hand side in Equation (3) and Equation (1). We implement the negative log-likelihood function with parameters $\sigma, \mathbf{x}$ in the code block below.

```r
library(RTMB)
parameters <- list(sigma = 1,x = rep(0,366))

f <- function(params){
  getAll(params, warn = FALSE)
  y <- OBS(rain$n.rain)
  p <- 1/(1+exp(-x))
  ll <- sum(dbinom(y,rain$n.years,p,log = T))
  ll <- ll + sum(-log(sigma) - (1/(2*sigma^2)) * diff(x)^2)
  n_ll <- -ll
  return (n_ll)
}
```

The RTMB method approximates the marginal likelihood $\pi(\boldsymbol{y}|\sigma_u^2)$ using the Laplace approximation in the integral $\pi(\boldsymbol{y}|\sigma_u^2) = \int \pi(\boldsymbol{y}, \boldsymbol{x}|\sigma_u^2)d\boldsymbol{x}$. We use the `MakeADFun` function to do this for us. This function can then be optimised w.r.t. σ_u^2 to find the maximum likelihood estimate. The implementation is shown below.

```r
obj <- MakeADFun(func = f, parameters = parameters,random = c("x"))

est <- nlminb(obj$par,obj$fn,obj$gr)
```

```
## iter: 1  value: 772.8688 mgc: 18.5 ustep: 1
## iter: 2  value: 732.0511 mgc: 3.761861 ustep: 1
## iter: 3  value: 731.309 mgc: 0.7297733 ustep: 1
## iter: 4  value: 731.3079 mgc: 0.04124705 ustep: 1
## iter: 5  value: 731.3079 mgc: 0.0001372943 ustep: 1
## iter: 6  mgc: 1.425925e-09
## iter: 1  mgc: 1.425925e-09
## Matching hessian patterns... Done
## outer mgc:  240.2704
## iter: 1  Error in iterate(par) :
##   Newton dropout because inner gradient had non-finite components.
## iter: 1  value: 698.2018 mgc: 0.483941 ustep: 1
## iter: 2  value: 698.2018 mgc: 0.006668156 ustep: 1
## iter: 3  value: 698.2018 mgc: 1.567174e-06 ustep: 1
## iter: 4  mgc: 1.434408e-13
## iter: 1  mgc: 1.434408e-13
## outer mgc:  249.3308
## iter: 1  value: 621.0747 mgc: 1.553556 ustep: 1
## iter: 2  value: 621.0741 mgc: 0.04508973 ustep: 1
## iter: 3  value: 621.0741 mgc: 4.430443e-05 ustep: 1
## iter: 4  mgc: 6.542966e-11
## iter: 1  value: 26.37286 mgc: 155.3015 ustep: 1
## iter: 2  value: 26.11185 mgc: 1.103906 ustep: 1
## iter: 3  value: 26.11184 mgc: 0.002188459 ustep: 1
## iter: 4  value: 26.11184 mgc: 9.423665e-08 ustep: 1
## iter: 5  mgc: 3.774758e-14
## iter: 1  value: NaN mgc: 8.666403 ustep: 0
## mgc: 8.666403
## iter: 1  mgc: 3.774758e-14
## outer mgc:  72.30457
## iter: 1  value: NaN mgc: 6.994092 ustep: 0
## mgc: 6.994092
## iter: 1  value: -101.9976 mgc: 10.47343 ustep: 1
## iter: 2  value: -101.9978 mgc: 0.01225027 ustep: 1
## iter: 3  value: -101.9978 mgc: 2.159939e-06 ustep: 1
## iter: 4  mgc: 2.033929e-13
## iter: 1  value: -24.72737 mgc: 3.203309 ustep: 1
## iter: 2  value: -24.72738 mgc: 0.001802125 ustep: 1
## iter: 3  value: -24.72738 mgc: 6.714225e-08 ustep: 1
## iter: 4  mgc: 5.151435e-14
## iter: 1  mgc: 5.151435e-14
## outer mgc:  11.35926
## iter: 1  value: -35.16388 mgc: 0.585074 ustep: 1
## iter: 2  value: -35.16388 mgc: 7.806032e-05 ustep: 1
## iter: 3  mgc: 1.162034e-10
## iter: 1  mgc: 1.162034e-10
## outer mgc:  3.967689
## iter: 1  value: -32.42972 mgc: 0.1474621 ustep: 1
## iter: 2  value: -32.42972 mgc: 5.379679e-06 ustep: 1
## iter: 3  mgc: 5.663959e-13
## iter: 1  mgc: 5.663959e-13
## outer mgc:  0.1543926
## iter: 1  value: -32.53171 mgc: 0.00554609 ustep: 1
## iter: 2  mgc: 7.472985e-09
## iter: 1  mgc: 7.472985e-09
## outer mgc:  0.002012072
## iter: 1  value: -32.53306 mgc: 7.322176e-05 ustep: 1
## iter: 2  mgc: 1.29674e-12
## iter: 1  mgc: 1.29674e-12
## outer mgc:  1.055895e-06
## iter: 1  value: -32.53306 mgc: 3.840555e-08 ustep: 1
## mgc: 5.612177e-14
## iter: 1  mgc: 1.29674e-12
```

```r
cat("The maximum likelihood estimate for sigma^2 is: ",est$par^2)
```

```
## The maximum likelihood estimate for sigma^2 is:  0.006961482
```

We see that the parameter estimate for σ_u^2 is 0.00696. In part 1, the 95% credible intervals from the MCMC method were found to be $[0.00594, 0.0164]$, so the new parameter estimate lies within this interval.