Project 2

Marcus Hjørund & Gard Gravdal

2024-03-17

Contents


```
setwd("C:/Users/marcu/OneDrive/Dokumenter/8. semester/Beregningskrevende statistiske metoder/Project2Be
load("~/8. semester/Beregningskrevende statistiske metoder/Project2Beregningskrevende/rain.rda")
library(ggplot2)
library(matrixStats)
library(MASS)
library("INLA")
library(RTMB)
library(Matrix)
library(coda)
```

1 Problem 1

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

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

for n_t being 10 for $t = 60$ (Feburary 29th) and 39 for all other days. $\pi(x_t)$ is the probability of rainfall exceeding 1mm of days $t = 1, \ldots, T$ and $T = 366$. Note that x_t is the logit probability of exceedence and can be obtained from $\pi(x_t)$ via $x_t = \log(\pi(x_t)/(1-\pi(x_t)))$. We assume conditional independence among the $y_t|x_t$ for all $t = 1, \ldots, 366$.

1.1 a)

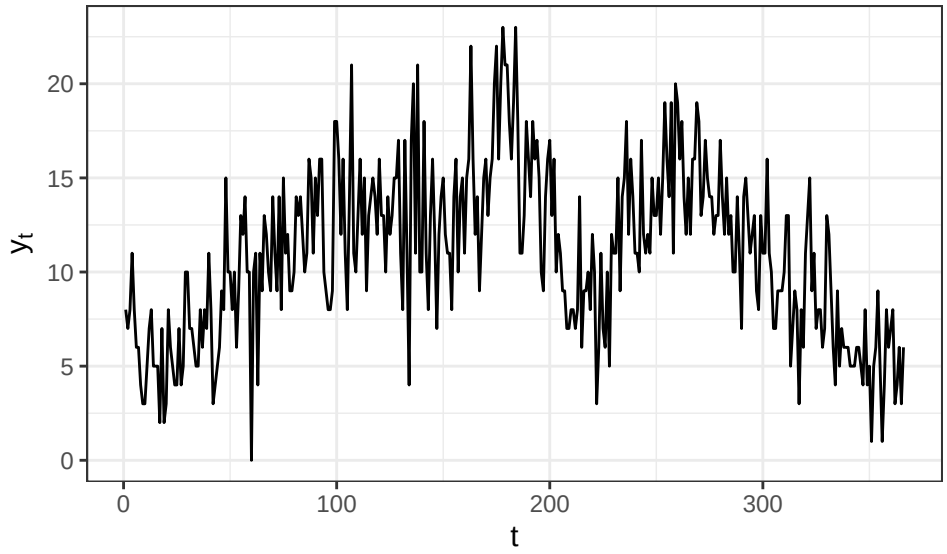


Figure 1: Tokyo rain data

As we can see in Figure 1 the rain in Tokyo seems to show seasonal trend. There are few occurences of rain more than 1mm in the period December-January, the peak of rainfall is in June-July, with a following drier period in August-September.

1.2 b)

Next we want to obtain an expression for the likelihood of y_t depending on the parameters $\pi(x_t)$ for $t = 1, \ldots, 366$. This gives the likeihood for the data $\boldsymbol{y}$ given the probabilities $\pi(\boldsymbol{x})$, where we will later update $\boldsymbol{x}$ using our a MCMC algorithm. Since we assume conditional intependence for $y_t|x_t$ we can write the likehood, using the expression for $\pi(x_t)$ given in Equation (1):

$$\begin{aligned}
L(\pi(\boldsymbol{x})|\boldsymbol{y}) &= \prod_{t=1}^{T} \binom{n_t}{y_t} \left(\frac{\exp(x_t)}{1+\exp(x_t)}\right)^{y_t} \left(1 - \frac{\exp(x_t)}{1+\exp(x_t)}\right)^{n_t - y_t} \\
&= \prod_{t=1}^{T} \binom{n_t}{y_t} \left(\frac{\exp(x_t)}{1+\exp(x_t)}\right)^{y_t} \left(\frac{1}{1+\exp(x_t)}\right)^{n_t - y_t} \\
&= \prod_{t=1}^{T} \binom{n_t}{y_t} \frac{\exp(x_t y_t)}{(1+\exp(x_t))^{n_t}}
\end{aligned} \tag{2}$$

1.3 c)

Now we want to apply a Bayesian hierarchical model to the dataset, where we use 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,

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

Then we place a inverse gamma prior on σ_u^2 such that,

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

for shape and scale α and β. We let $\boldsymbol{y} = (y_1, y_2, \ldots, y_T)^T$, $\boldsymbol{x} = (x_1, x_2, \ldots, x_T)^T$ and $\boldsymbol{\pi} = (\pi(x_1), \pi(x_2), \ldots, \pi(x_T))^T$.

Now we will derive the posteriori distribution for $\sigma_u^2|\boldsymbol{x}$, which is used in Gibbs sampling in the hybrid sampler in Section 1.6. If we draw a directed acyclical graph for the hierarchical Bayesian model, we see that $\boldsymbol{y}$ is independent of σ_u^2, as all information from $\boldsymbol{y}$ is stored in $\boldsymbol{x}$. From Bayes theorem we know that

$$p(\sigma_u^2|\boldsymbol{y}, \boldsymbol{x}) \propto p(\sigma_u^2)p(\boldsymbol{x}|\sigma_u^2).$$

We then have that

$$\begin{aligned}
p(\sigma_u^2|\boldsymbol{x}, \boldsymbol{y}) &\propto \frac{\beta^\alpha}{\Gamma(\alpha)}(1/\sigma_u^2)^{\alpha+1} \exp(-\beta/\sigma_u^2) \prod_{t=2}^{T} \frac{1}{\sigma_u} \exp\left\{-\frac{1}{2\sigma_u^2}(x_t - x_{t-1})^2\right\} \\
&= \frac{\beta^\alpha}{\Gamma(\alpha)}(1/\sigma_u^2)^{\alpha+1} \exp(-\beta/\sigma_u^2) \cdot (\sigma_u^2)^{-\frac{T-1}{2}} \exp\left\{-\frac{1}{2\sigma_u^2}\sum_{t=2}^{T}(x_t - x_{t-1})^2\right\} \\
&\propto (\sigma_u^2)^{-(\alpha+\frac{T-1}{2}+1)} \exp\left\{-\frac{1}{\sigma_u^2}\left(\beta + \frac{1}{2}\sum_{t=2}^{T}(x_t - x_{t-1})^2\right)\right\}
\end{aligned} \tag{4}$$

which follows an inverse gamma distribution with parameters

$$\alpha^* = \alpha + \frac{T-1}{2} \text{ and } \beta^* = \beta + \frac{1}{2}\sum_{t=2}^{T}(x_t - x_{t-1})^2.$$

1.4 d)

In the Metropolis-Hastings algorithm, which we will implement, we need to find an expression for the acceptance probability α, which in general is given as

$$\alpha(y|x) = \min\left\{1, \frac{\pi(y)Q(x|y)}{\pi(x)Q(y|x)}\right\}$$

where $Q(x|y)$ is a proposed conditional probability mass function (pmf), $\pi(x)$ is the probability of being in state x, and the transition probability is given as

$$p(y|x) = \begin{cases} Q(y|x)\alpha(y|x) & \text{for } y \neq x \\ 1 - \sum_{y \neq x} Q(y|x)\alpha(y|x) & \text{for } y = x \end{cases}$$

which is the probability of transitioning to state y given that the current state is x. The expression for α can be derived from the detailed balance equation, $\pi(x)p(y|x) = \pi(y)p(x|y)$ which is true for a time reversible Markov chain.

We will now consider the conditional prior proposal distribution, $Q(\boldsymbol{x}'_{\mathcal{I}}|\boldsymbol{x}_{-\mathcal{I}}, \sigma_u^2, \boldsymbol{y}) = p(\boldsymbol{x}'_{\mathcal{I}}|\boldsymbol{x}_{-\mathcal{I}}, \sigma_u^2)$, where $\boldsymbol{x}'_{\mathcal{I}}$ is the proposed values for $\boldsymbol{x}_{\mathcal{I}}$, $\mathcal{I} \subseteq \{1, \ldots, 366\}$ is a set of time indices, and $\boldsymbol{x}_{-\mathcal{I}} = \boldsymbol{x}_{\{1,\ldots,366\}\setminus\mathcal{I}}$ is a subset of $\boldsymbol{x}$ that includes all other indices than those in $\mathcal{I}$. Our expression for the acceptance probability is then

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

We can simplify this expression by looking at $p(\boldsymbol{x}_{\mathcal{I}}|\boldsymbol{x}_{-\mathcal{I}}, \sigma_u^2, \boldsymbol{y})$ and $p(\boldsymbol{x}'_{\mathcal{I}}|\boldsymbol{x}_{-\mathcal{I}}, \sigma_u^2, \boldsymbol{y})$. From probability theory we know that $P(A|B) = \frac{P(A\cap B)}{P(B)}$, such that we can divide the expression of $p(\boldsymbol{x}'_{\mathcal{I}}|\boldsymbol{x}_{-\mathcal{I}}, \sigma_u^2, \boldsymbol{y})$ as follows:

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

where we have used that $\boldsymbol{y}$ is independent of σ_u^2 and that $\boldsymbol{x}_{\mathcal{I}}$ and $\boldsymbol{x}_{-\mathcal{I}}$ are disjoint sets, and $y_t|x_t$ are conditionally independent. We perform a similar calculation for $p(\boldsymbol{x}_{\mathcal{I}}|\boldsymbol{x}_{-\mathcal{I}}, \sigma_u^2, \boldsymbol{y})$ and obtain

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

We can then simplify $\alpha(\boldsymbol{x}'_{\mathcal{I}}|\boldsymbol{x}_{-\mathcal{I}}, \sigma_u^2, \boldsymbol{y})$, which becomes

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

We can then see that $\alpha(\boldsymbol{x}'_{\mathcal{I}}|\boldsymbol{x}_{-\mathcal{I}}, \sigma_u^2, \boldsymbol{y})$ becomes a ratio of likelihoods for the proposed values $\boldsymbol{x}'_{\mathcal{I}}$ against the existing values $\boldsymbol{x}_{\mathcal{I}}$.

1.5 e)

We note that the density specified by (3) is improper, for example for $T = 2$, density takes the shape of an infinite 'Gaussian ridge' centered around the line given by $x_1 = x_2$. Equation (3) can be rewritten as

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

which resembles a multivariate normal density but where the improperiety of the density translates into the precision matrix

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

having one zero eigenvalue. We partition the components of $\boldsymbol{x}$ into two subvectors writing

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

and partitioning the precision matrix in the same way as

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

We then want to derive the conditional distribution of $\boldsymbol{x}_A$ conditional on $\boldsymbol{x}_B$. We know from before that the conditional density is always proportional to the joint (improper) density. We assume without loss of generality that $\boldsymbol{\mu} = \mathbf{0}$ and find that

$$\begin{aligned} f_{\boldsymbol{x}_A|\boldsymbol{x}_B}(\boldsymbol{x}_A) &\propto \exp\left\{-\frac{1}{2}(\boldsymbol{x}_A^T, \boldsymbol{x}_B^T)\begin{bmatrix} \mathbf{Q}_{AA} & \mathbf{Q}_{AB} \\ \mathbf{Q}_{BA} & \mathbf{Q}_{BB} \end{bmatrix}\begin{pmatrix} \boldsymbol{x}_A \\ \boldsymbol{x}_B \end{pmatrix}\right\} \\ &= \exp\left\{-\frac{1}{2}(\boldsymbol{x}_A^T\mathbf{Q}_{AA}\boldsymbol{x}_A + \boldsymbol{x}_B^T\mathbf{Q}_{BA}\boldsymbol{x}_A + \boldsymbol{x}_A^T\mathbf{Q}_{AB}\boldsymbol{x}_B + \boldsymbol{x}_B^T\mathbf{Q}_{BB}\boldsymbol{x}_B)\right\} \\ &\propto \exp\left\{-\frac{1}{2}(\boldsymbol{x}_A - \boldsymbol{\mu}_{A|B})^T\mathbf{Q}_{AA}(\boldsymbol{x}_B - \boldsymbol{\mu}_{A|B})\right\} \\ &\propto \exp\left\{-\frac{1}{2}(\boldsymbol{x}_A^T\mathbf{Q}_{AA}\boldsymbol{x}_A - \boldsymbol{\mu}_{A|B}^T\mathbf{Q}_{AA}\boldsymbol{x}_A - \boldsymbol{x}_A^T\mathbf{Q}_{AA}\boldsymbol{\mu}_{A|B})\right\} \end{aligned}$$

where we only consider terms involving $\boldsymbol{x}_A$, which is random in this case, and we use the fact that we can write the terms involving $\boldsymbol{x}_A$ in the second line in quadratic form, completing the square. We then equate the coefficients in the last line to the terms in the second line to find our expressions for $\boldsymbol{\mu}_{A|B}$ and $\mathbf{Q}_{A|B}$. We have

$$-\mathbf{Q}_{AA}\boldsymbol{\mu}_{A|B} = \mathbf{Q}_{AB}\boldsymbol{x}_B \implies \boldsymbol{\mu}_{A|B} = -\mathbf{Q}_{AA}^{-1}\mathbf{Q}_{AB}\boldsymbol{x}_B \tag{7}$$

$$\mathbf{Q}_{A|B} = \mathbf{Q}_{AA}. \tag{8}$$

These results is used to simulate from $\boldsymbol{x}'_{\mathcal{I}}|\boldsymbol{x}_{-\mathcal{I}}$ later on when we generate realizations from the random walk.

1.6 f)

In this section we want to implement an MCMC sampler from the posterior distribution $p(\boldsymbol{\pi}, \sigma_u^2|\boldsymbol{y})$ using Metropolis-Hastings (MH) steps for individual x_t parameters using the conditional prior, $p(x_t|\boldsymbol{x}_{-t}, \sigma_u^2)$, and Gibbs steps for σ_u^2.

We will begin by deriving the conditional prior distribution using the results from Equations (7) and (8) as the conditional mean and precision (inverse variance) of $x'_{\mathcal{I}}|\boldsymbol{x}_{-\mathcal{I}}, \sigma_u^2$. We need to derive the conditional prior distribution for three cases, $t = 1$, $2 \leq t \leq 365$ and $t = 366$. For the case $t = 1$, $\mathcal{I} = \{1\}$, we have for the conditional mean that $-\mathbf{Q}_{\mathcal{I}\mathcal{I}}^{-1} = -\sigma_u^2$, $\mathbf{Q}_{\mathcal{I}-\mathcal{I}} = \frac{1}{\sigma_u^2}(-1, 0, \ldots, 0)$ with dimension $1 \times (T-1)$. Thus the conditional expectation is given as

$$x_{\mathcal{I}}|\boldsymbol{x}_{-\mathcal{I}}, \sigma_u^2 = -\mathbf{Q}_{\mathcal{I}\mathcal{I}}^{-1}\mathbf{Q}_{\mathcal{I}-\mathcal{I}}\boldsymbol{x}_{-\mathcal{I}} = -\sigma_u^2 \cdot \frac{1}{\sigma_u^2}(-1, 0, \ldots, 0)(x_2, x_3, \ldots, x_T)^T = x_2. \tag{9}$$

Similarly we have for $t = 366$, $\mathcal{I} = \{366\}$, we have

$$x_{\mathcal{I}}|\boldsymbol{x}_{-\mathcal{I}}, \sigma_u^2 = -\sigma_u^2 \cdot \frac{1}{\sigma_u^2}(0, \ldots, 0, -1)(x_1, \ldots, x_{364}, x_{365})^T = x_{365}. \tag{10}$$

Then finally for the interior points, $2 \leq t \leq 365$, $\mathcal{I} = \{2\}, \{3\}, \ldots, \{365\}$, we have $-\mathbf{Q}_{\mathcal{I}\mathcal{I}}^{-1} = -\frac{\sigma_u^2}{2}$ and $\mathbf{Q}_{\mathcal{I}-\mathcal{I}} = \frac{1}{\sigma_u^2}(0, \ldots, 0, -1, -1, 0, \ldots, 0)$ with dimension $1 \times (T-1)$. We then calculate the conditional expectation as

$$x_{\mathcal{I}}|\boldsymbol{x}_{-\mathcal{I}}, \sigma_u^2 = -\frac{\sigma_u^2}{2} \cdot \frac{1}{\sigma_u^2}(0, \ldots, 0, -1, -1, 0, \ldots, 0)(x_1, \ldots, x_{t-1}, x_{t+1}, \ldots, x_{366})^T = \frac{1}{2}(x_{t-1} + x_{t+1}). \tag{11}$$

We perform a similar calculation to calculate the conditional variance. We have that the conditional variance $\boldsymbol{\Sigma}_{\mathcal{I}|-\mathcal{I},\sigma_u^2} = \mathbf{Q}^{-1}_{\mathcal{I}|-\mathcal{I},\sigma_u^2}$. For $t = 1$ and $t = 366$ we have that $\mathbf{Q}^{-1}_{\mathcal{I}\mathcal{I}} = \sigma_u^2$ and for $2 \leq t \leq 365$ we have $\mathbf{Q}^{-1}_{\mathcal{I}\mathcal{I}} = \frac{\sigma_u^2}{2}$. Thus we can summarize the conditional distribution of $x_{\mathcal{I}}|\boldsymbol{x}_{-\mathcal{I}}$ as

$$x_{\mathcal{I}}|\boldsymbol{x}_{-\mathcal{I}}, \sigma_u^2 \sim \begin{cases} \mathcal{N}(x_2, \sigma_u^2) & \text{for } t = 1 \\ \mathcal{N}(\frac{1}{2}(x_{t-1} + x_{t+1}), \frac{\sigma_u^2}{2}) & \text{for } t = 2, \ldots, 365 \\ \mathcal{N}(x_{365}, \sigma_u^2) & \text{for } t = 366. \end{cases} \tag{12}$$

This is the distribution we simulate from on the random walk to generate realizations of x'_t, which is the proposed value for x at index t. Next we will derive the expression for the acceptance probability $\alpha(x'_t|\boldsymbol{x}_{-t}, \sigma_u^2, \boldsymbol{y})$ for $\mathcal{I} = \{t\}$. We use the expression for the acceptance probability derived in Equation (5). We have

$$\begin{aligned} \alpha(x'_t|\boldsymbol{x}_{-t}, \sigma_u^2, \boldsymbol{y}) &= \min\left\{1, \frac{p(y_t|x'_t)}{p(y_t|x_t)}\right\} = \min\left\{1, \frac{\binom{n_t}{y_t}\frac{\exp(x'_t y_t)}{(1+\exp(x'_t))^{n_t}}}{\binom{n_t}{y_t}\frac{\exp(x_t y_t)}{(1+\exp(x_t))^{n_t}}}\right\} \\ &= \min\left\{1, \exp\left(y_t(x'_t - x_t)\right) \cdot \frac{(1+\exp(x_t))^{n_t}}{(1+\exp(x'_t))^{n_t}}\right\} \\ &= \min\left\{1, \exp\left(y_t(x'_t - x_t) - n_t \ln\left(\frac{1+\exp(x'_t)}{1+\exp(x_t)}\right)\right)\right\} \end{aligned}$$

Note that in the final line, we have written the expression on log-form, then exponentiating in the end. We do this to avoid potential overflow/underflow problems in the implementation. Having derived the expressions needed, we give the algorithm for the MCMC-sampler below:

MCMC-algorithm:

Initialize $x = x_0$ and $\sigma_u^2 \sim p(\sigma_u^2)$

Repeat n times:

for $t = 1, \ldots, T$:

Perform MH-stem for x_t:

generate $x'_t \sim p(\sigma_u^2|\boldsymbol{x}_{-t})$

$\alpha \leftarrow \min\left\{1, \exp\left(y_t(x'_t - x_t) - n_t \ln\left(\frac{1+\exp(x'_t)}{1+\exp(x_t)}\right)\right)\right\}$

generate $u \sim \text{unif}(0, 1)$

if $(u < \alpha)$:

$x_t \leftarrow x'_t$

else

$x_t \leftarrow x_t$

end for

Perform Gibbs step for σ_u^2 :

$\beta^* = \beta + \frac{1}{2}\sum_{t=2}^{T}(x_t - x_{t-1})^2$

generate $\sigma_u^2 \sim \text{InvGamma}(\alpha + \frac{T-1}{2}, \beta^*)$

end for

Return samples $x_1, ..., x_T$

Next, we implement the MCMC-algorithm. We being by implementing some helper-functions for the MCMC-algorithm, as well as defining the global variables needed.

```r
# Problem 1
expit <- function(x){
  return(exp(x)/(1 + exp(x)))
}

# f)

# Defining global variables
alpha = 2
beta = 0.05
N = 50000 #Number of iterations in MCMC

alpha_prob <- function(x_prop, x, t){
  # Calculates acceptance probability
  y = rain$n.rain[t]
  n = rain$n.years[t]
  ans = exp(y*(x_prop - x) - n * log((1+exp(x_prop))/(1+exp(x))))
  return(min(1,ans))
}

MH_step <- function(x, sigma2, accept_count, mu, t){
  # Calculates one step of Metropolis-Hastings for x_t
  # All inputs are numbers
  # Returns updated accept count and x_t (updated/not updated)
  x_prop <- rnorm(1, mu, sqrt(sigma2))

  # Probability of acceptance
  alpha_accept <- alpha_prob(x_prop, x, t)
  u <- runif(1)

  if (alpha_accept > u){
    x <- x_prop
    accept_count = accept_count + 1
  }
  return(list(x = x, accept_count = accept_count))

}

moments_prop <- function(x,t,sigma2, T_max = 366){
  # Returns the first two moments as defined in text
  # Input x is (1 x T_max) vector, t,sigma2 are numbers
  if (t == 1){
    return(list(mu= x[2],sigma2 = sigma2 ))
  }
  else if (t == T_max) {
    return(list(mu = x[T_max-1], sigma2 = sigma2))
  }
  else{
    return(list(mu = 0.5*(x[t-1] + x[t+1]), sigma2 = sigma2/2))
  }
}
```

The MCMC-algorithm, which returns samples of $\boldsymbol{x}$, is implemented. Note that the algorithm returns the samples from the burn-in period also.

```
MCMC <- function(n = N, T_max = 366){
  # Time start
  time = proc.time()[3]
  # Initialization
  accept_count = 0
  x <- rep(1,T_max*n)
  x <- matrix(x, nrow = n, ncol = T_max)
  sigma2 <- rep(1,n)
  sigma2[1] <- rgamma(1, alpha, rate = beta)^-1
    # MCMC-iterations:
  for (i in 2:n){
    old_sigma2 = sigma2[i-1]
    for (t in 1:T_max){
      param = moments_prop(x[i-1,],t,old_sigma2, T_max = T_max)
      mu_temp = param$mu
      sigma2_temp = param$sigma2
      MH_step_t <- MH_step(x[i-1,t], sigma2_temp, accept_count, mu_temp, t)
      x[i,t] <- MH_step_t$x
      accept_count <- MH_step_t$accept_count
    }

    # Gibbs step, updating sigma2
    z = beta + 0.5*sum((x[i,-T_max] - x[i, -1])^2)
    sigma2[i] <- 1/rgamma(1,alpha + (T_max-1)/2, rate = z)
  }
  accept_rate = accept_count/ (n * T_max)
  total_time = proc.time()[3] - time

  return(list(x = x, total_time = total_time, accept_rate = accept_rate, sigma2 = sigma2))
}
set.seed(1337)
MCMC_run <- MCMC()
```

First we check the runtime and acceptance rate for the algorithm.

```
MCMC_accept <- MCMC_run$accept_rate
MCMC_accept
```

```
## [1] 0.7568243
```

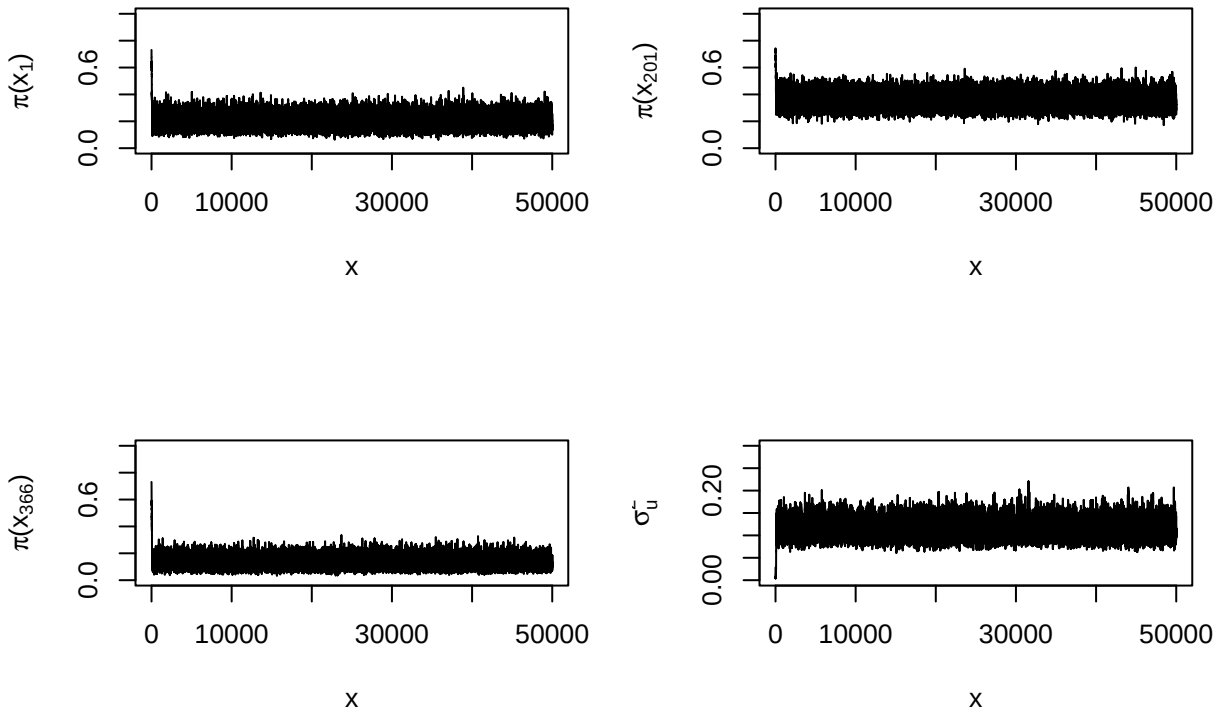
```
MCMC_time <- MCMC_run$total_time
MCMC_time
```

```
## elapsed
##  207.46
```

We see that the algorithm runs in 207.46 seconds, and obtains an acceptance rate of 0.7568243. In the single-site algorithm we have used here we need to do all the calculations for each step t and for each iteration i, such that we perform 366×50000 calculations in the algorithm. As we will see later on, we can greatly speed up this process by performing block iterations, computing several matrices beforehand and thus greatly improve the time used.

Next, we display the traceplots of $\pi(x_1), \pi(x_{201}), \pi(x_{366})$ and σ_u^2 from our realization of the MCMC-run.

```r
burnin <- ceiling(N/10)
interval <- burnin:N
x_vec = seq(1,N)
ylab1 = expression(paste(pi, "(x"[1],")"))
ylab2 = expression(paste(pi, "(x"[201],")"))
ylab3 = expression(paste(pi, "(x"[366],")"))
ylab4 = expression(paste(sigma[u]^2))
par(mfrow = c(2, 2), mar = c(5,4,4,2))
plot(x_vec, expit(MCMC_run$x[,1]), ylim = c(0,1), ylab = ylab1, xlab = "x", type = "l")
plot(x_vec, expit(MCMC_run$x[,201]), ylim = c(0,1), ylab = ylab2, xlab = "x", type = "l")
plot(x_vec, expit(MCMC_run$x[,366]), ylim = c(0,1), ylab = ylab3, xlab = "x", type = "l")
plot(x_vec, MCMC_run$sigma2, ylim = c(0,0.3), ylab = ylab4, xlab = "x", type = "l")
```

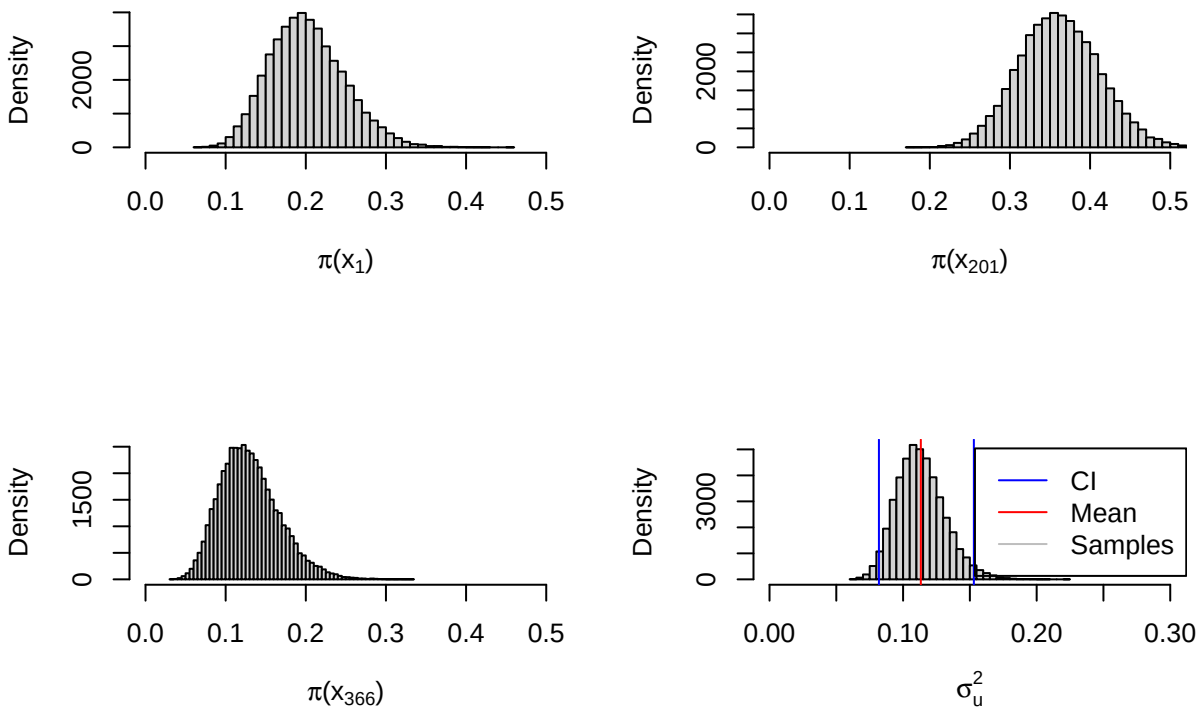


We see from the plots that the Markov Chain converges quite quickly. For $\pi(x_1)$ we obtain values around 0.2, for $\pi(x_{201})$ we obtain values around 0.4 and for $\pi(x_{366})$ we obtain values around 0.18. The variance also converges to a value around 0.12.

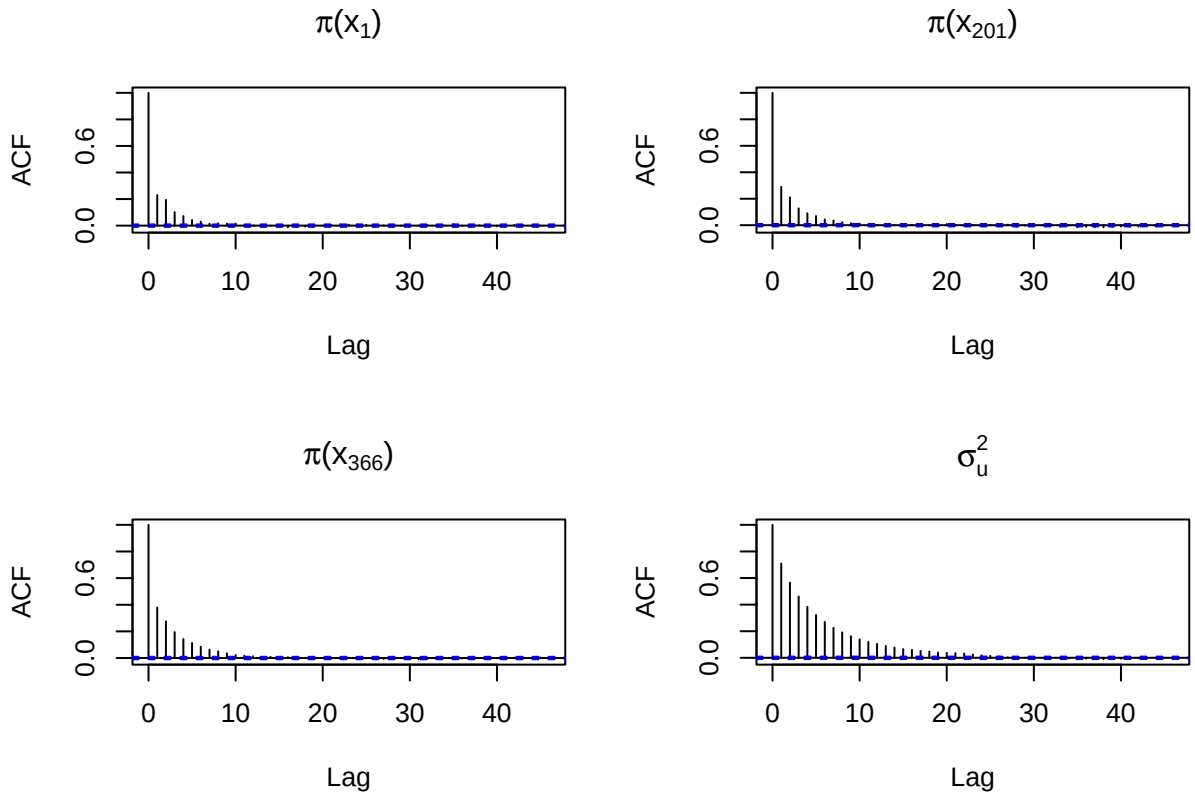
We also explore the histograms and auto correlation functions of $\pi(x_1), \pi(x_{201}), \pi(x_{366})$ and σ_u^2, which is given below. For σ_u^2, we also provide the mean and 95% confidence interval.

```r
# Plotting histograms
# General histogram function
plot_hist <- function(x, xlab = "x", ylab = "Density", breaks = 50, xlim = c(0,1), alpha = 0.05){
  # Plots general histogram with quantiles and mean
  quantiles <- quantile(x, probs = c(alpha/2, 1 - alpha/2))
  hist(x, breaks = breaks, xlab = xlab, ylab = ylab, xlim = xlim, main = NULL)
  abline(v = quantiles[1], col = "blue")
```

```
  abline(v = quantiles[2], col = "blue")
  abline(v = mean(x), col = "red")
  legend("right",legend = c("CI", "Mean", "Samples"), col = c("blue", "red", "grey"), lty = 1)
}

# Plots
xlab1 = expression(paste(pi, "(x"[1],")"))
xlab2 = expression(paste(pi, "(x"[201],")"))
xlab3 = expression(paste(pi, "(x"[366],")"))
xlab4 = expression(paste(sigma[u]^2))
par(mfrow = c(2, 2), mar = c(5,4,4,2))
hist(expit(MCMC_run$x[,1])[interval], breaks = 55, xlab = xlab1,ylab = "Density", xlim = c(0,0.5), main
hist(expit(MCMC_run$x[,201])[interval], breaks = 55, xlab = xlab2,ylab = "Density", xlim = c(0,0.5), ma
hist(expit(MCMC_run$x[,366])[interval], breaks = 55, xlab = xlab3,ylab = "Density", xlim = c(0,0.5), ma
plot_hist(MCMC_run$sigma2[interval],xlim = c(0,0.3), xlab = xlab4)
```




```
par(mfrow = c(1, 1))

# ACFs:
par(mfrow = c(2, 2), mar = c(5,4,4,2))
acf(expit(MCMC_run$x[,1])[interval], main = xlab1)
acf(expit(MCMC_run$x[,201])[interval], main = xlab2)
acf(expit(MCMC_run$x[,366])[interval], main = xlab3)
acf(MCMC_run$sigma2[interval], main = xlab4)
```

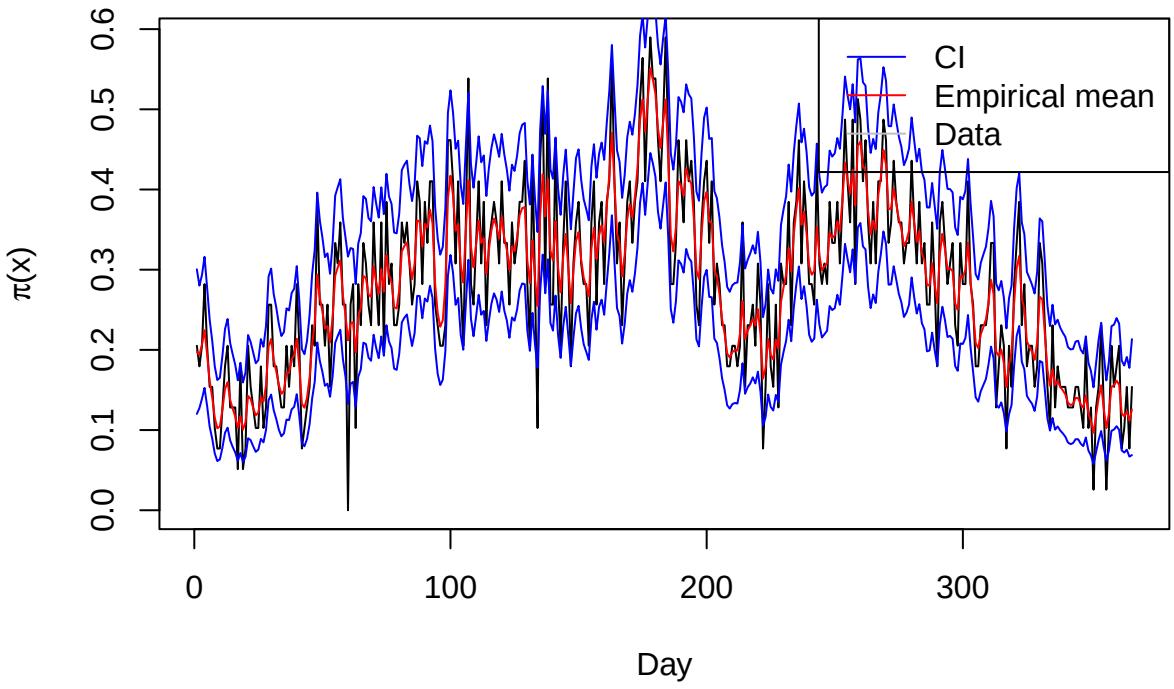



```
par(mfrow = c(1, 1), mar = c(5,5,1,1))
```

We see that the histograms coincides with the traceplots. We see from the auto correlation plots that the acf decays quickly as we have little correlation for increasing lag. This indicates the MCMC-algorithm works well, as the acf decays rapidly, and the samples are approximately independent and identically distributed.

In the next section we display a graph showing 95% credible limits and the posterior mean $\pi(\boldsymbol{x}_t)$ as a function of $t = 1, ..., 366$ and compare the predictions for π and associated uncertainties to the data y_t/n_t as a function of t-

```
mean_x = colMeans(MCMC_run$x[interval,])
pi_data = rain$n.rain/rain$n.years

x_quantiles <- colQuantiles(MCMC_run$x[interval,], probs = c(0.025, 0.975))
par(mfrow = c(1, 1))
plot(1:366,pi_data, type = "l",xlab = "Day", ylab = expression(paste(pi, "(x)")))
lines(expit(mean_x), type = "l", col = "red")
lines(expit(x_quantiles[,1]), type = "l", col = "blue")
lines(expit(x_quantiles[,2]), type = "l", col = "blue")
legend("topright",legend = c("CI", "Empirical mean", "Data"), col = c("blue", "red", "grey"), lty = 1)
```



From the plot we can see that the implementation of the MCMC algorithm works, as we can see that most of the data falls within the 90% credible intervals.

1.7 g)

In this section we will analyze the effect of using conditional prior proposal involving $p(\boldsymbol{x}_{(a,b)}|\boldsymbol{x}_{-(a,b)}, \sigma_u^2)$ where $\boldsymbol{x}_{(a,b)} = (x_a, ..., x_b)^T$ where the interval (a, b) is of length M, as opposed to updating individual x_t. We will repeat the previous section for this setup. We will analyze which M gives the most desirable result based on the diagnostics run time and effective sample size (ESS), and also analyze how implementing a block step over the $\boldsymbol{x}_{(a,b)}$-parameters in this way affects the efficiency of the MCMC-sampler. The idea is that precomputing $\mathbf{Q}_{AA}^{-1}\mathbf{Q}_{AB}$ as well as the Cholesky decomposition $\mathbf{L}\mathbf{L}^T = \mathbf{Q}_{AA}^{-1}$ will speed up the result.

To implement the MCMC-algorithm on block form, we need to update some parameters as well as introducing some new notation. We are now working with the interval $\mathcal{I} = \{a, ..., b\}$ as opposed to $\mathcal{I} = \{t\}$ in the previous section. First, we find that the new expression for the acceptance probability is given by

$$\begin{aligned}\alpha &= \min\left\{1, \prod_{i=a}^{b} \frac{p(\boldsymbol{y}_{(a,b)}|\boldsymbol{x}'_{(a,b)})}{p(\boldsymbol{y}_{(a,b)}|\boldsymbol{x}_{(a,b)})}\right\} \\ &= \min\left\{1, \exp\left(\sum_{i=a}^{b} y_t(x'_t - x_t) - n_t \ln\left(\frac{1+\exp(x'_t)}{1+\exp(x_t)}\right)\right)\right\}\end{aligned}$$

We also note that the derivation for the Gibbs step of σ_u^2 still holds, such that each Gibbs step we generate $\sigma_u^2 \sim \text{InvGamma}(\alpha + \frac{T-1}{2}, \beta^*)$ in the MCMC-algorithm.

To minimize the run time in the MCMC_block-algorithm, we precompute $\mathbf{Q}_{AA}^{-1}\mathbf{Q}_{AB}$ as well as the Cholesky decomposition $\mathbf{L}\mathbf{L}^T = \mathbf{Q}_{AA}^{-1}$, where $A = (a, b)$ and $B = -(a, b)$. Given the properties of the matrix $\mathbf{Q}$, we

need to do this for three different cases, namely for the first interval, when $a = 1, b = M$, the last interval, where $b = T$, and then for all other intervals, when $a > 1, b < T$. This is because all the rows of $\mathbf{Q}$ are the same, except for the first and last row, which will result in three different results for $\mathbf{L}$ and $\mathbf{Q}_{(a,b)-(a,b)}$. Note that the last interval is only of length M when M is a multiple of the total amount of days $T = 366$, otherwise it is of length T mod M. We introduce the notation $\eta := -\mathbf{Q}^{-1}_{(a,b)(a,b)}\mathbf{Q}_{(a,b)-(a,b)} = \mathbf{L}\mathbf{L}^T\mathbf{Q}_{(a,b)-(a,b)}$, such that $\boldsymbol{\mu}_{(a,b)|-(a,b)} = \eta\ \boldsymbol{x}_{-(a,b)}$. We also introduce the notation $* \in \{1, m, M\}$, where 1 denotes the index for the first interval, i.e. $(1, M)$, M denotes the index for the last interval, i.e. (a, T), and m denotes the index for all the other intervals, i.e. the inner intervals. Having introduced this notation, we find that we need to precompute the parameters $\mathbf{L}_1, \mathbf{L}_m, \mathbf{L}_M$ as well as η_1, η_M. The means $\mu_{(1,M)|-(1,M)} = \eta_1\ \boldsymbol{x}_{-(1,M)}$ and $\mu_{(a,T)|-(a,T)} = \eta_M\ \boldsymbol{x}_{-(a,T)}$ are then calculated for each iteration in the MCMC-algorithm. Note that we do not precompute η_m, since the matrix $\mathbf{Q}_{(a,b)-(a,b)}$ is different for all the inner blocks. We do not want to compute η_m for each iteration of the MCMC-algorithm. To avoid this, we use the fact that $\mathbf{Q}_{(a,b)-(a,b)}$ is sparse, such that we can calculate a simple expression for the mean $\boldsymbol{\mu}_{(a,b)|-(a,b)}$ in terms of the elements of $\boldsymbol{x}_{-(a,b)}$. We do not want to compute η_m for each iteration of the MCMC-algorithm. To avoid this, we use the fact that $\mathbf{Q}_{(a,b)-(a,b)}$ is sparse. The matrix $\mathbf{Q}_{(a,b)-(a,b)}$ will look like

$$\mathbf{Q}_{(a,b)-(a,b)} = \begin{pmatrix} 0 & \dots & -1 & 0 & \dots & 0 \\ 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \\ 0 & \dots & 0 & -1 & \dots & 0 \\ . & & & & & \end{pmatrix}$$

The index where we have the two -1's will shift to index $(t-1)\cdot M$ for the first column and $(t-1)\cdot M + 1$ for the second column for $t = 2, \dots, \lfloor T/M \rfloor$. When we multiply $\mathbf{Q}_{(a,b)-(a,b)}$ with $-Q^{-1}_{AA}$ we obtain a matrix with dimension $M \times (T - M)$ with the first column of Q^{-1}_{AA} at column index $(t-1)\cdot M$, and the last column of Q^{-1}_{AA} at index $(t-1)\cdot M + 1$, which is our matrix η_m. Finally, we give the full MCMC_block-algorithm below. Note that we have divided the algorithm into the two algorithms MCMC_Block-algorithm and MH_block-step-algorithm.

MCMC_Block-algorithm:

Initialize $x = x_0$ and $\sigma_u^2 \sim p(\sigma_u^2)$

Precompute the following:

$\mathbf{L}_*$: From Cholesky decomposition of $\mathbf{Q}^{-1}_{(a,b)(a,b)}$, i.e. $\mathbf{Q}^{-1}_{(a,b)(a,b)} = \mathbf{L}\mathbf{L}^T$

$\eta_{*\setminus\{m\}} = \mathbf{L}_{*\setminus\{m\}}\mathbf{L}^T_{*\setminus\{m\}}$

If T mod $M \neq 0$:

Inner iterations $T_{iter} = \lfloor T/M \rfloor$

else

Inner iterations $T_{iter} = \lfloor T/M \rfloor - 1$

Repeat n times:

$\boldsymbol{\mu}_1 = \eta_1 \boldsymbol{x}_{-(1,M)}$

$\boldsymbol{x}_{(1,M)} \leftarrow$ MH_block-step$(\boldsymbol{x}_{(1,M)}, \sigma_u^2, \boldsymbol{\mu}_1, \mathbf{L}_1)$

for $t = 2, ..., T_{iter}$:

$(a, b) = (M(t-1), tM)$

$\boldsymbol{\mu}_m = \eta_m \boldsymbol{x}_{-(a,b)}$

$\boldsymbol{x}_{(a,b)} \leftarrow$ MH_block-step$(\boldsymbol{x}_{(a,b)}, \sigma_u^2, \boldsymbol{\mu}_m, \mathbf{L}_m)$

end for

$(a,b) = (T - \lfloor T/M \rfloor M, T)$

$\boldsymbol{\mu}_M = \eta_M \boldsymbol{x}_{-(a,b)}$

$ $ $\boldsymbol{x}_{(a,b)} \leftarrow$ MH_block-step$(\boldsymbol{x}_{(a,b)}, \sigma_u^2, \boldsymbol{\mu}_M, \mathbf{L}_M)$

Perform Gibbs step for σ_u^2 :

$\beta^* = \beta + \frac{1}{2}\sum_{t=2}^{T}(x_t - x_{t-1})^2$

generate $\sigma_u^2 \sim \text{InvGamma}(\alpha + \frac{T-1}{2}, \beta^*)$

Return samples $x_1, ..., x_T$

MH_Block-step-algorithm:

Input: $\boldsymbol{x}_{(a,b)} \in \mathbb{R}^M, \sigma_u^2 \in \mathbb{R}, \boldsymbol{\mu} = \boldsymbol{\mu}_{(a,b)|-(a,b)}$ and $\mathbf{L} = \mathbf{L}_*$

generate $\boldsymbol{x}'_{(a,b)} = \boldsymbol{\mu} + \mathbf{L}\boldsymbol{Z}_M \in \mathbb{R}^M$, where $\boldsymbol{Z}_M = (Z_1, ..., Z_M)$, $Z \overset{iid}{\sim} N(0,1)$

$\alpha \leftarrow \min\left\{1, \exp\left(\sum_{i=a}^{b} y_t(x'_t - x_t) - n_t \ln\left(\frac{1+\exp(x'_t)}{1+\exp(x_t)}\right)\right)\right\}$

generate $u \sim \text{unif}(0,1)$

If $u < \alpha$:

$\boldsymbol{x}_{(a,b)} \leftarrow \boldsymbol{x}'_{(a,b)}$

else

$\boldsymbol{x}_{(a,b)} \leftarrow \boldsymbol{x}_{(a,b)}$

end for

Return $\boldsymbol{x}_{(a,b)}$

Next we implement the MCMC-block algorithm. Some helper-functions for algorithm is given below.

```
MH_block_step <- function(x_ab, sigma2, accept_count, mu, L, interval){
  # Generate proposed x in R^M
  # MH_block-step as defined in text
  m = length(x_ab)
  x_prop <- mu + (sqrt(sigma2)*L) %*% rnorm(m)
  alpha_accept <- alpha_prob_block(x_prop, x_ab, interval)
  u <- runif(1)
  if (alpha_accept > u){
    x_ab <- x_prop
    accept_count <- accept_count + 1
  }
  return(list(x = x_ab, accept_count = accept_count))
  }

alpha_prob_block <- function(x_prop, x, interval){
  # Calculates acceptance probability
  # x_prop,x in R^M
  y = rain$n.rain[interval]
  n = rain$n.years[interval]
  ans = exp(sum(y*(x_prop - x) - n* log((1+exp(x_prop))/(1+exp(x)))))
  # ans = exp(sum(y*(x_prop - x) - n * log((1+exp(x_prop))/(1+exp(x)))))
  return(min(1,ans))
}
```

Below is an MCMC-block-algorithm that takes care of the special case of $M > T/2$. In this special case, the main algorithm will not work. This function is summoned by the main MCMC-block-algorithm if it is ran in this special case, but since this case is not tested in the report, the function does not demand attention.

```
MCMC_special <- function(n = N, T_max = 366, M = 3){
  # Monte Carlo Markov chain simulation using blocks,
  # expansion of function MCMC()
  # Special case for M > T/2
  # Time start
  time = proc.time()[3]
  # Initialization
  # This boolean helps control some special cases
  T_mod_M_is_zero = (T_max %% M) == 0
  accept_count = 0
  x <- rep(1,T_max*n)
  x <- matrix(x, nrow = n, ncol = T_max)
  sigma2 <- rep(1,n)
  sigma2[1] <- rgamma(1, alpha, rate = beta)^-1
  # Create matrix Q as defined in text
  Q <- diag(x = 2, nrow  = T_max)
  Q[abs(row(Q) - col(Q)) == 1] <- -1
  Q[1,1] <- 1
  Q[T_max,T_max] <- 1
  # Precompute eta and cholesky
  # Defining the intervals of importance
  idx_1 <- 1:M
  idx_M <- (floor(T_max/M)*M):T_max
  idx_M = (T_max-M):T_max
  L_1 <- t(chol(solve(Q[idx_1,idx_1])))
  L_m <- t(chol(solve(Q[idx_m,idx_m])))
  L_M <- t(chol(solve(Q[idx_M,idx_M])))

  eta_1 <- -L_1%*%t(L_1)%*%Q[idx_1,-idx_1]
  eta_M <- -L_M%*%t(L_M)%*%Q[idx_M,-idx_M]
  for (i in 2:n){
    # Solving for case a = 1, b = M:
    temp_idx <- 1:M
    temp_mu <- eta_1 %*% x[i-1,-(1:M)]
    MH_elem <- MH_block_step(x[i-1,1:M],sigma2[i-1], accept_count, temp_mu,L_1, temp_idx)
    x[i,1:M] <- MH_elem$x
    accept_count <- MH_elem$accept_count

    # Solving for case b = T_max
    temp_idx <- idx_M
    temp_mu <- eta_M %*% x[i-1,-idx_M]
    MH_elem <- MH_block_step(x[i-1,idx_M], sigma2[i-1], accept_count, temp_mu, L_M, temp_idx)
    x[i,idx_M] <- MH_elem$x
    accept_count <- MH_elem$accept_count
    # Updating sigma2 by Gibbs
    z = beta + 0.5*sum((x[i,-T_max] - x[i, -1])^2)
    sigma2[i] <- 1/rgamma(1,alpha + (T_max-1)/2, rate = z)
  }
  if (!T_mod_M_is_zero){
```

```
    accept_rate = accept_count/ (n * (iter + 1))
  } else{
    accept_rate = accept_count/ (n*iter)
  }

  total_time = proc.time()[3] - time
  return(list(x = x, total_time = total_time, accept_rate = accept_rate, sigma2 = sigma2))
}
```

Below we implement the MCMC-block-algorithm as described in the text. Note that we also store the count of accepted proposals in the algorithm.

```
MCMC_block <- function(n = N, T_max = 366, M = 3){
  # Monte Carlo Markov chain simulation using blocks,
  # expansion of function MCMC()
  # Special case of M >= T/2
  if (M >= T_max/2){
    return(MCMC_special(n,T_max,M))
  }
  # Time start
  time = proc.time()[3]
  # Initialization
  # This boolean helps control some special cases
  T_mod_M_is_zero = (T_max %% M) == 0
  accept_count = 0
  x <- rep(1,T_max*n)
  x <- matrix(x, nrow = n, ncol = T_max)
  sigma2 <- rep(1,n)
  sigma2[1] <- rgamma(1, alpha, rate = beta)^-1
  # Create matrix Q as defined in text
  Q <- diag(x = 2, nrow  = T_max)
  Q[abs(row(Q) - col(Q)) == 1] <- -1
  Q[1,1] <- 1
  Q[T_max,T_max] <- 1
  # Precompute eta and cholesky
  # Defining the intervals of importance
  idx_1 <- 1:M
  idx_m <- c((M+1):(2*M))
  idx_M <- (floor(T_max/M)*M):T_max
  iter <- floor(T_max/M)
  if (T_mod_M_is_zero){
    iter = iter - 1
    idx_M = (T_max-M):T_max
  }
  L_1 <- t(chol(solve(Q[idx_1,idx_1])))
  L_m <- t(chol(solve(Q[idx_m,idx_m])))
  L_M <- t(chol(solve(Q[idx_M,idx_M])))

  eta_1 <- -L_1%*%t(L_1)%*%Q[idx_1,-idx_1]
  Sigma_AA <- L_m%*%t(L_m)
  eta_m <- matrix(rep(0, M*(T_max-M)), nrow = M,
                  ncol = (T_max-M))
  eta_M <- -L_M%*%t(L_M)%*%Q[idx_M,-idx_M]
```

```
  for (i in 2:n){
    # Solving for case a = 1, b = M:
    temp_idx <- 1:M
    temp_mu <- eta_1 %*% x[i-1,-(1:M)]
    MH_elem <- MH_block_step(x[i-1,1:M],sigma2[i-1], accept_count, temp_mu,L_1, temp_idx)
    x[i,1:M] <- MH_elem$x
    accept_count <- MH_elem$accept_count

    # Solving for case a>1, b<T_max
    for (t in 2:iter){
      a <- (M*(t-1) + 1)
      b <- t*M
      temp_idx <- a:b
      eta_m[,(t-1)*M] <- Sigma_AA[,1]
      eta_m[,(t-1)*M+1] <- Sigma_AA[,M]
      temp_mu <- eta_m %*% x[i-1,-(a:b)]
      MH_elem <- MH_block_step(x[i-1,a:b],sigma2[i-1],accept_count, temp_mu, L_m, temp_idx)
      x[i,a:b] <- MH_elem$x
      accept_count <- MH_elem$accept_count
      # Reset eta_m
      eta_m[,(t-1)*M] <- rep(0,M)
      eta_m[,((t-1))*M+1] <- rep(0,M)
    }
    # Solving for case b = T_max
    temp_idx <- idx_M
    temp_mu <- eta_M %*% x[i-1,-idx_M]
    MH_elem <- MH_block_step(x[i-1,idx_M], sigma2[i-1], accept_count, temp_mu, L_M, temp_idx)
    x[i,idx_M] <- MH_elem$x
    accept_count <- MH_elem$accept_count
    # Updating sigma2 by Gibbs
    z = beta + 0.5*sum((x[i,-T_max] - x[i, -1])^2)
    sigma2[i] <- 1/rgamma(1,alpha + (T_max-1)/2, rate = z)
  }
  if (!T_mod_M_is_zero){

    accept_rate = accept_count/ (n * (iter + 1))
  } else{
    accept_rate = accept_count/ (n*iter)
  }

  total_time = proc.time()[3] - time
  return(list(x = x, total_time = total_time, accept_rate = accept_rate, sigma2 = sigma2))
}
```

We test our implementation below. We display the same plots for $\pi(x_1), \pi(x_{201}), \pi(x_{366})$ and σ_u^2 as we did in the previous section, only now with the realizations from the MCMC-block-algorithm with $M = 10$.

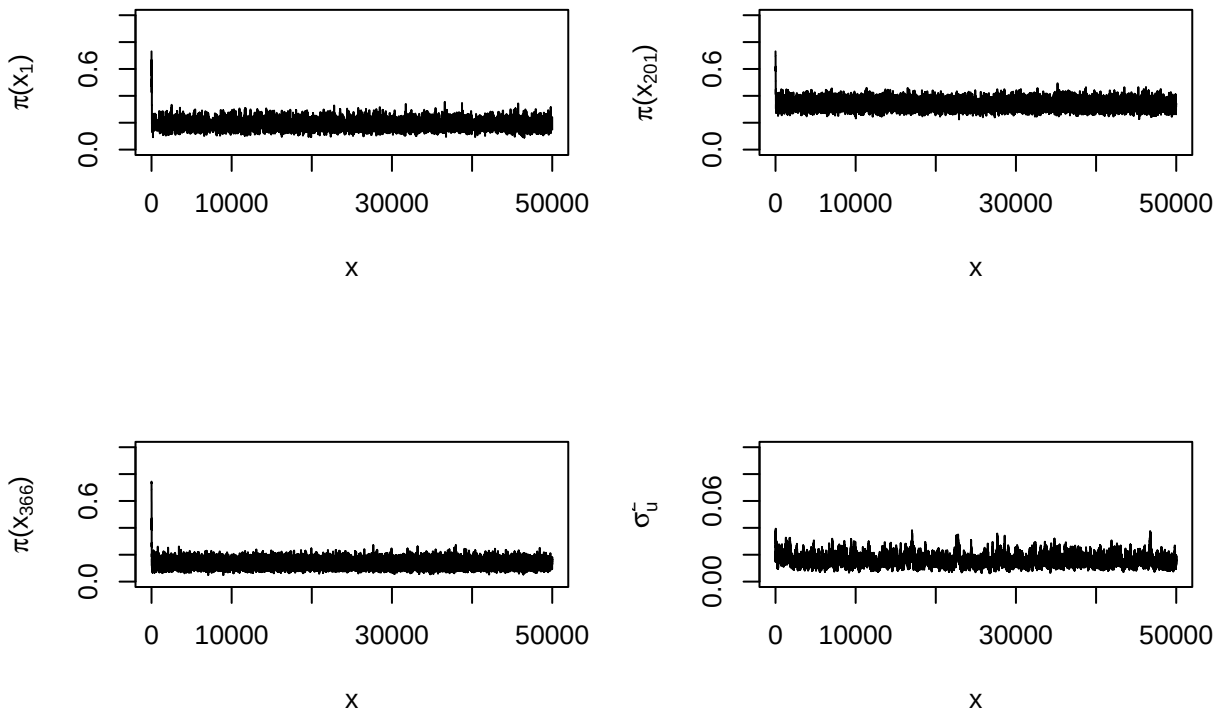
```
set.seed(1337)
MCMC_block_run = MCMC_block(M = 10)
MCMC_block_run$total_time
```

```
## elapsed
##   36.84
```

```
MCMC_block_run$accept_rate
```

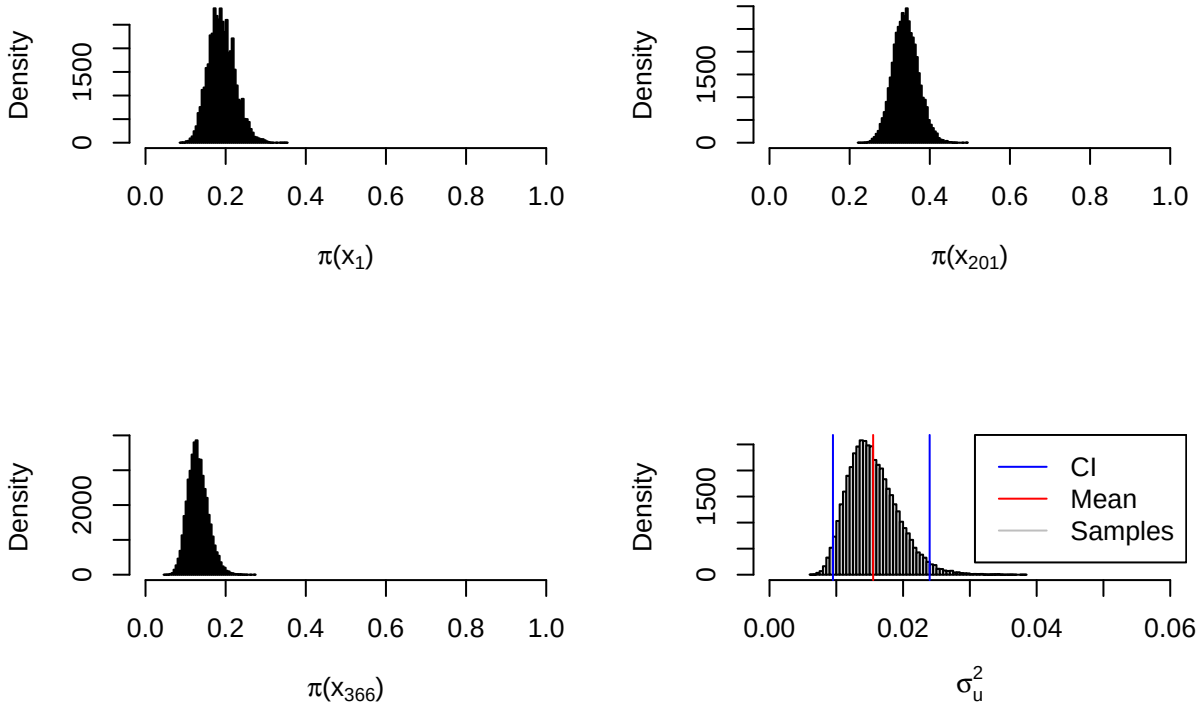
```
## [1] 0.3777578
```

```
MCMC_block_time <- MCMC_block_run$total_time
burnin <- ceiling(N/10)
interval <- burnin:N
x_vec = seq(1,N)

par(mfrow = c(2, 2), mar = c(5,4,4,2))
plot(x_vec, expit(MCMC_block_run$x[,1]), ylim = c(0,1), ylab = ylab1, xlab = "x", type = "l")
plot(x_vec, expit(MCMC_block_run$x[,201]), ylim = c(0,1), ylab = ylab2, xlab = "x", type = "l")
plot(x_vec, expit(MCMC_block_run$x[,366]), ylim = c(0,1), ylab = ylab3, xlab = "x", type = "l")
plot(x_vec, MCMC_block_run$sigma2, ylim = c(0,0.1), ylab = ylab4, xlab = "x", type = "l")
```

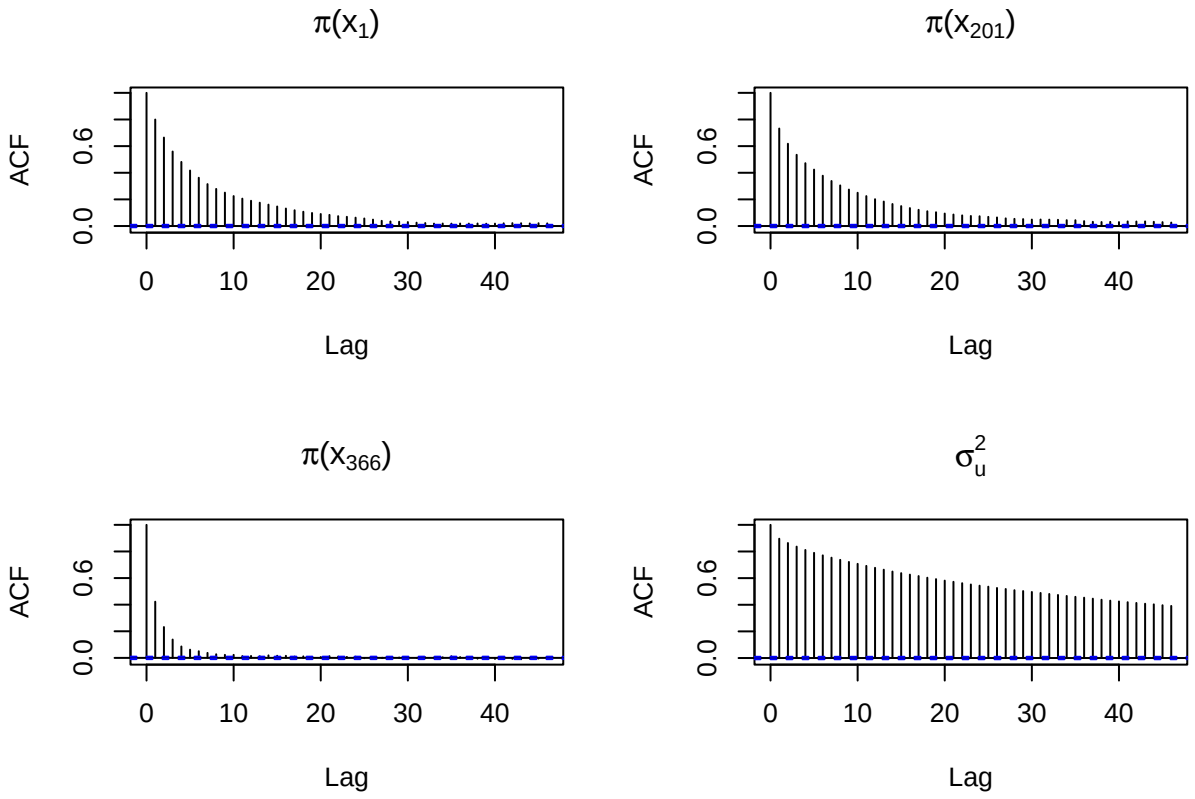



```
# Plotting histograms
# General histogram function
plot_hist <- function(x, xlab = "x", ylab = "Density", breaks = 50, xlim = c(0,1), alpha = 0.05){
  # Plots general histogram with quantiles and mean
  quantiles <- quantile(x, probs = c(alpha/2, 1 - alpha/2))
  hist(x, breaks = breaks, xlab = xlab, ylab = ylab, xlim = xlim, main = NULL)
  abline(v = quantiles[1], col = "blue")
  abline(v = quantiles[2], col = "blue")
  abline(v = mean(x), col = "red")
```

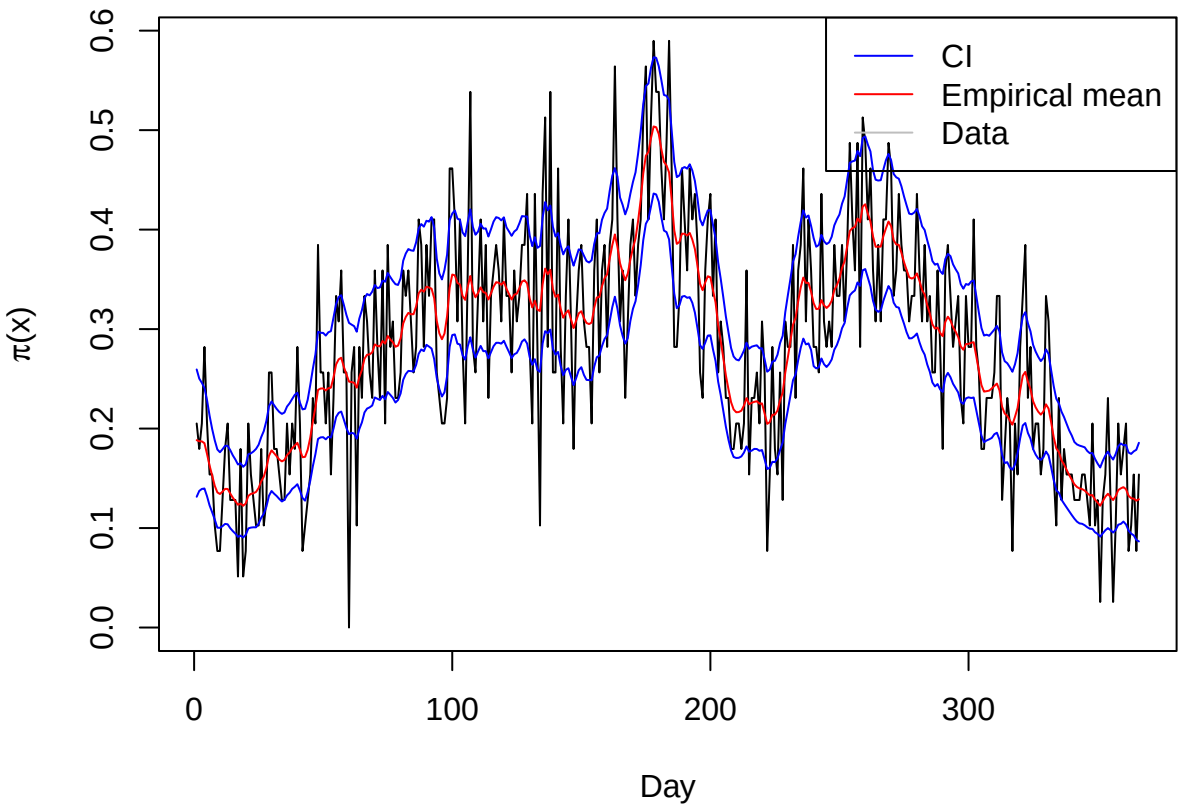
```
  legend("topright",legend = c("CI", "Mean", "Samples"), col = c("blue", "red", "grey"), lty = 1)
}
# Plots
xlab1 = expression(paste(pi, "(x"[1],")"))
xlab2 = expression(paste(pi, "(x"[201],")"))
xlab3 = expression(paste(pi, "(x"[366],")"))
xlab4 = expression(paste(sigma[u]^2))
par(mfrow = c(2, 2), mar = c(5,4,4,2))
hist(expit(MCMC_block_run$x[,1])[interval], breaks = 45, xlab = xlab1,ylab = "Density", xlim = c(0,1), m
hist(expit(MCMC_block_run$x[,201])[interval], breaks = 45, xlab = xlab2,ylab = "Density", xlim = c(0,1)
hist(expit(MCMC_block_run$x[,366])[interval], breaks = 45, xlab = xlab3,ylab = "Density", xlim = c(0,1)
plot_hist(MCMC_block_run$sigma2[interval],xlim = c(0,0.06), xlab = xlab4)
```




```
par(mfrow = c(1, 1))


# ACFs:
par(mfrow = c(2, 2), mar = c(5,4,4,2))
acf(expit(MCMC_block_run$x[,1])[interval], main = xlab1)
acf(expit(MCMC_block_run$x[,201])[interval], main = xlab2)
acf(expit(MCMC_block_run$x[,366])[interval], main = xlab3)
acf(MCMC_block_run$sigma2[interval], main = xlab4)
```




```
par(mfrow = c(1, 1), mar = c(5,5,1,1))

mean_x = colMeans(MCMC_block_run$x[interval,])
pi_data = rain$n.rain/rain$n.years

x_quantiles <- colQuantiles(MCMC_block_run$x[interval,], probs = c(0.025, 0.975))
par(mfrow = c(1, 1))
plot(1:366,pi_data, type = "l",xlab = "Day", ylab = expression(paste(pi, "(x)")))
lines(expit(mean_x), type = "l", col = "red")
lines(expit(x_quantiles[,1]), type = "l", col = "blue")
lines(expit(x_quantiles[,2]), type = "l", col = "blue")
legend("topright",legend = c("CI", "Empirical mean", "Data"), col = c("blue", "red", "grey"), lty = 1)
```



We see that the trace plots show that the samples converge quickly, similar to the run in the previous section for $M = 1$. We see that the acceptance rate has decreased a lot compared to the run in the previous section, while the run time has decreased a lot. By eye test, the plots of the histograms seem similar to the histograms given in the previous section, however we can tell that the acceptance rate has decreased given that the trace plots appear less dense. Looking at the ACF-plots, we see that the ACF of $\pi(x_1)$ and $\pi(x_{201})$ decays slowly, while the ACF of σ_u^2 decays even slower. The ACF of $\pi(x_{366})$ seems to decay much faster, and the auto correlation is approximately 0 at lag 10. In general, all the ACF's decay slower than in the MCMC-run with $M = 1$ in the previous section. This should be expected given the decreased acceptance rate, which makes the samples more correlated for increasing lag. Finally, looking at the plot of the posterior mean $\pi(\boldsymbol{x})$, with 95% credible limits, for $t = 1, ..., 366$, we see that the mean fits the data very well, and the credible limits are closer to the mean than in the case of $M = 1$, which indicates a decrease in variance.

Next we want to analyze the effects of varying the block size/interval length M. We will choose the M of our choice based on the diagnostics run time/acceptance rate and ESS. ESS, or effective sample size, is a diagnostic tool to test how reliable the estimates of the posterior quantities are. ESS is given by

$$ESS = \frac{n}{1 + 2\sum_{k=1}^{\infty} \rho(i)}$$

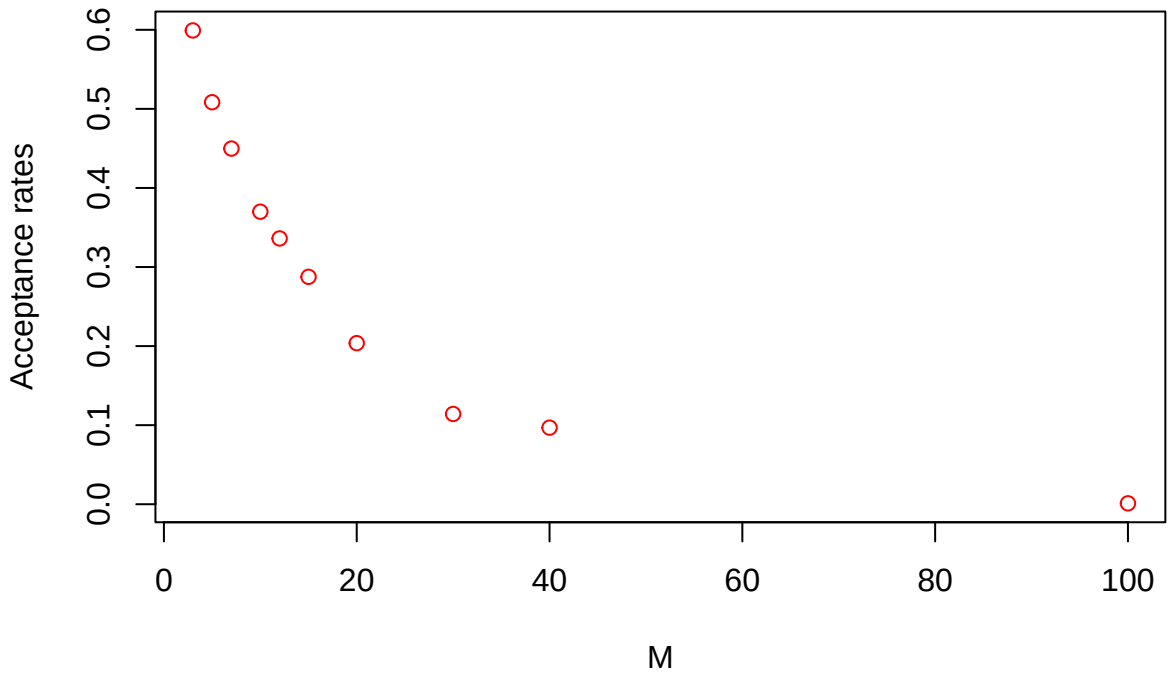
where pluggin in the estimate of correlation $\hat{\rho}$ given by

$$\hat{\rho}(k) = \frac{\sum_{i=1}^{n-k}(x_i - \bar{x})(x_{i+k} - \bar{x})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

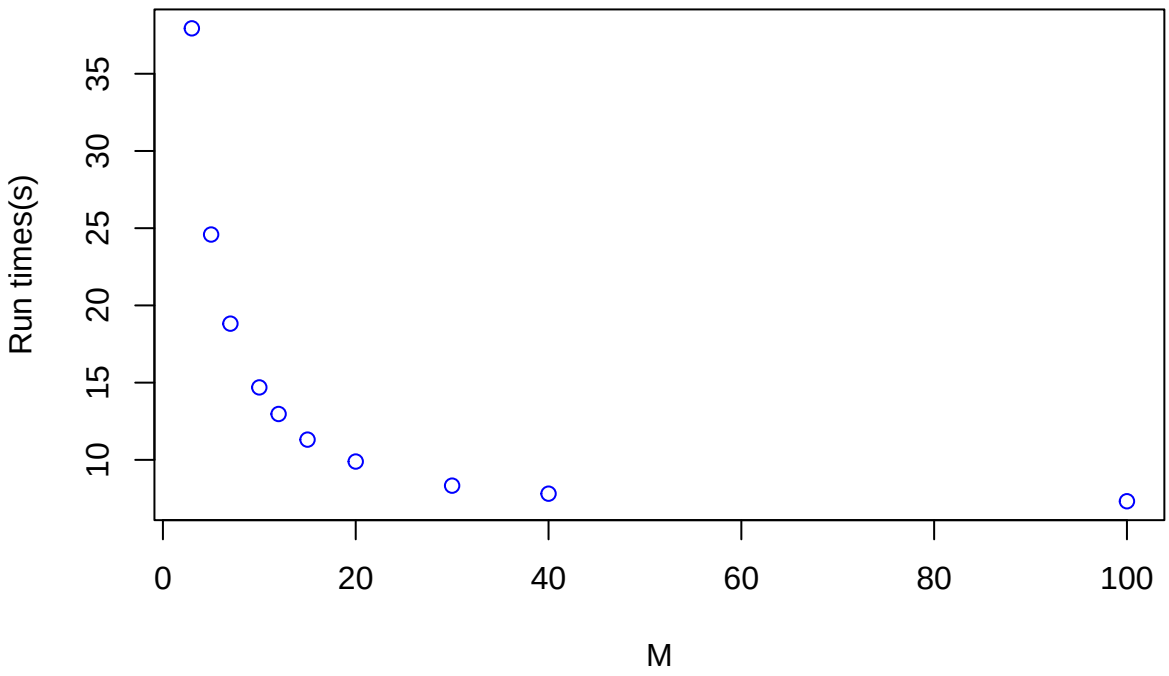
gives a "time series SE" of $\hat{\mu}$. Higher ESS indicates that the samples are more informative and less correlated, and that the samples have explored more of the state space. Thus, we want to maximize ESS. We remove the burn-in period before we test the ESS for the different values of M. Below we test the run time, acceptance

rate and ESS for M in $\{3, 5, 7, 10, 12, 15, 20, 30, 40, 100\}$. Note that we reduce the iterations of the MCMC to $N = 20000$ given the long run time for this code section.

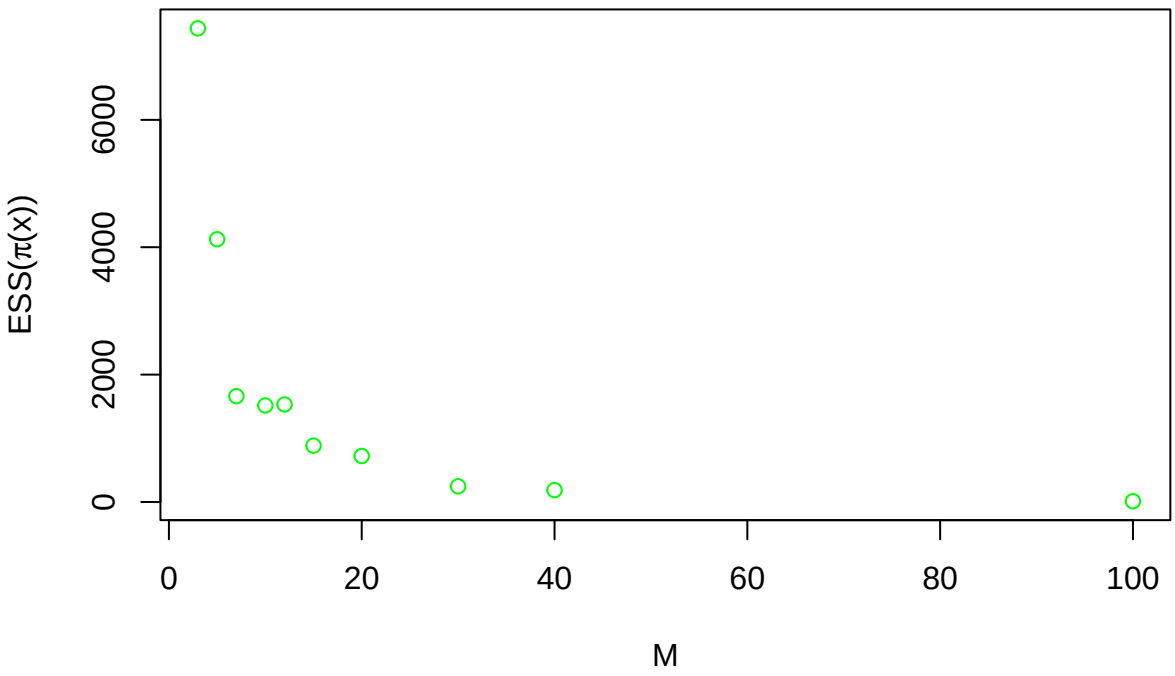
```
# Testing for different M:
M_tests = c(3,5,7,10,12,15,20,30,40,100)
N_test = 20000
burnin <- ceiling(N_test/10)
interval <- burnin:N_test
# Initialization
times = rep(0,length(M_tests))
accept_rates = rep(0, length(M_tests))
ESS_vals = rep(0, length(M_tests))

set.seed(1337)
for (i in 1:length(M_tests)){
  object = MCMC_block(n = N_test, M = M_tests[i])
  times[i] <- object$total_time
  accept_rates[i] <- object$accept_rate
  ESS_vals[i] <- mean(effectiveSize(expit(object$x)[interval]))
}
# Plotting the results
plot(M_tests,accept_rates, col = "red", xlab = "M", ylab = "Acceptance rates")
```




```
plot(M_tests,times, col = "blue", ylab = "Run times(s)", xlab = "M")
```




```
plot(M_tests, ESS_vals, col = "green", xlab = "M", ylab = expression(paste("ESS(",pi,"(x))")))
```



We see that the acceptance rates and the run time follows the same pattern, namely that they both decrease for increasing M. When comparing the run time of the algorithm with the length of interval M, we need to consider the cost of the preconditioning and the cost of running the iterations in the MCMC-block-algorithm. The preconditioning is a fixed cost for a given length M, in the sense that the time it takes to do the calculations in the preconditioning is independent on N. Thus, the higher number of iterations N, the more benefit we gain from the preconditioning. However, the run time for the preconditioning is increasing with M, since the operations involved in the preconditioning (i.e. the inversion of matrices, Cholesky-decomposition etc.) increase with M. We conclude that the benefit of preconditioning depends on how many iterations we run in the MCMC-run. Given that we have $N = 20000$ iterations in this run, which is a relatively big amount, we see that the benefit in terms of run time clearly outweighs the cost of preconditioning. When looking at the plot of the ESS, we see that it decreases with increasing M. This is consistent with the decrease in acceptance rate for increasing M. When the acceptance rate decreases, we accept fewer proposals, and thus our samples will be more correlated. Then the samples will explore less of the state space, and thus decreasing the ESS. When choosing optimal M, we should also consider the bias/variance trade of. Choosing a small M will increase the variance while decreasing the bias. Finally, we note that while increasing M decreases the ESS, it still gives a reasonable estimate when compared to the real data. In some instances, decreasing the run time can be more valuable than increased precision. There is no definite right answer to which M to choose, however we find that $M = 10$ strikes a nice balance considering the run time and the bias/variance trade off.

2 Problem 2

2.1 a)

In this section we will again look at the Tokyo dataset, but only now we will use the INLA package rather than the MCMC function implemented previously. We note that we use the simplified Laplace approximation and 'ccd' integration rather than Laplace approximation and grid integration respectively. We remove the intercept with the -1 option, and use a RW(1) by passing model = "rw1" option to the f() function. When comparing the INLA function with our own MCMC function we need to note that the prior in σ_u^2 in INLA is placed on the log precision rather than the variance. INLA assumes that the random walk is generated using $\Delta x = x_i - x_{i-1} \sim (0, \tau^{-1})$, where the precision parameter τ is represented as $\theta = \log \tau$ and the prior is defined on θ. We have

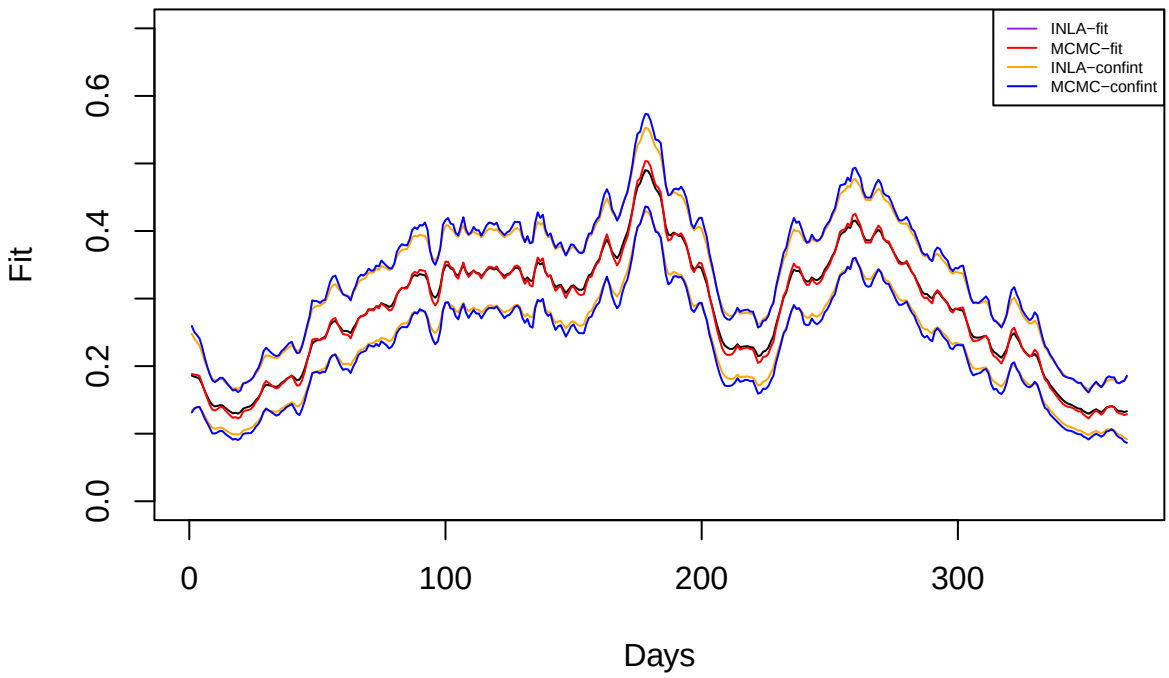
$$\log(\tau) = \log\left(\frac{1}{\sigma_u^2}\right) \sim \log\left(\frac{1}{\text{InvGamma}(\text{shape} = \alpha, \text{scale} = \beta)}\right) = \text{LogGamma}(\text{shape} = \alpha, \text{rate} = \beta).$$

Therefore we are able produce comparable results if we use a loggamma prior in INLA, since our prior is an inverse gamma, the loggamma prior produces a prior which is comparable to the one we use in the MCMC algorithm.

```
t <- proc.time()[3]
control.inla = list(strategy="simplified.laplace", int.strategy="ccd")
mod_no_intercept <- inla(n.rain ~ -1 + f(day, model="rw1", constr=FALSE, hyper = list(prec = list(prior
data=rain, Ntrials=n.years, control.compute=list(config = TRUE),
family="binomial", verbose=FALSE, control.inla=control.inla)

run_time <- proc.time()[3]-t
run_time
```

```
## elapsed
##    0.76
```

```
INLA_fit <- mod_no_intercept$summary.fitted.values$mean
lower <- mod_no_intercept$summary.fitted.values$`0.025quant`
upper <- mod_no_intercept$summary.fitted.values$`0.975quant`
plot(seq(1,366, by = 1), INLA_fit,type = "l", col = "black", xlab = "Days", ylab = "Fit", ylim = c(0, 0
lines(expit(mean_x), type = "l", col = "red")
lines(seq(1,366, by = 1), lower, type = "l", col  = "orange",
     xlab = "Days", ylab = "Fit")
lines(seq(1,366, by = 1), upper, type = "l", col  ="orange")
lines(expit(x_quantiles[,1]), type = "l", col = "blue")
lines(expit(x_quantiles[,2]), type = "l", col = "blue")
legend("topright",legend = c("INLA-fit", "MCMC-fit","INLA-confint", "MCMC-confint"), col = c("purple","
```

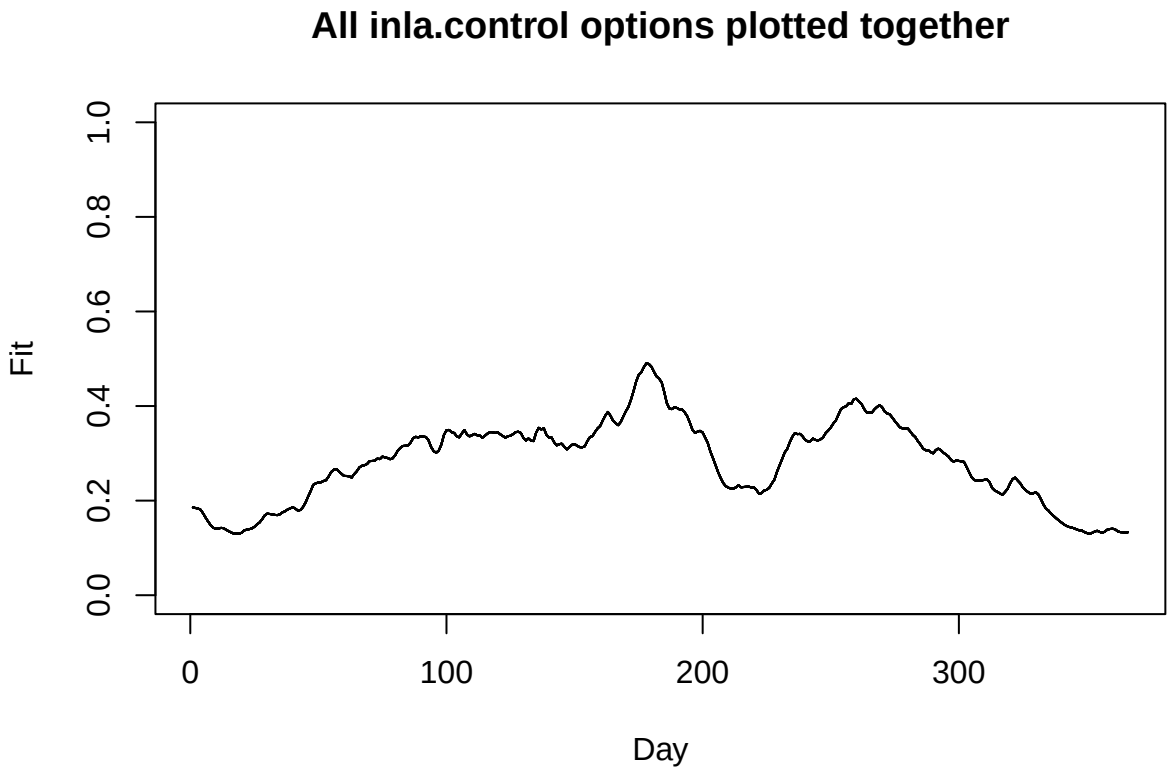


Looking at the predictions for π we see that the predictions from the MCMC algorithm is very similar to the predictions made by the INLA function. One difference is that the INLA package seems to produce "smoother" predictions, meaning that it allows less variation between each consecutive day. Where the INLA package is better is with the computation time, as the computation time of the INLA package is 0.76 while the computation time for the block MCMC implementation is 36.84.

2.2 b)

```
strategy = c('gaussian', 'simplified.laplace', 'laplace', 'adaptive')
int.strategy = c('ccd', 'grid', 'eb')
INLA_fits <- matrix(rep(0, length(int.strategy)*length(strategy)), ncol = 366, nrow = 4*3)
iter <- 1
for (i in 1:length(strategy)){
  for (j in 1:length(int.strategy)){
    control.inla = list(strategy = strategy[i], int.strategy = int.strategy[j])
    fit <- inla(n.rain ~ -1 + f(day, model="rw1", constr=FALSE, hyper = list(prec = list(prior = "loggam
data=rain, Ntrials=n.years, control.compute=list(config = TRUE),
family="binomial", verbose=F, control.inla=control.inla)
    INLA_fits[iter,] <- fit$summary.fitted.values$mean
    iter <- iter + 1
  }
}
plot(INLA_fits[1,], type = "l", ylim = c(0,1), xlab = "Day", ylab ="Fit", main = "All inla.control optio
for (i in 2:12){
```

```
  lines(INLA_fits[i,])
}
```

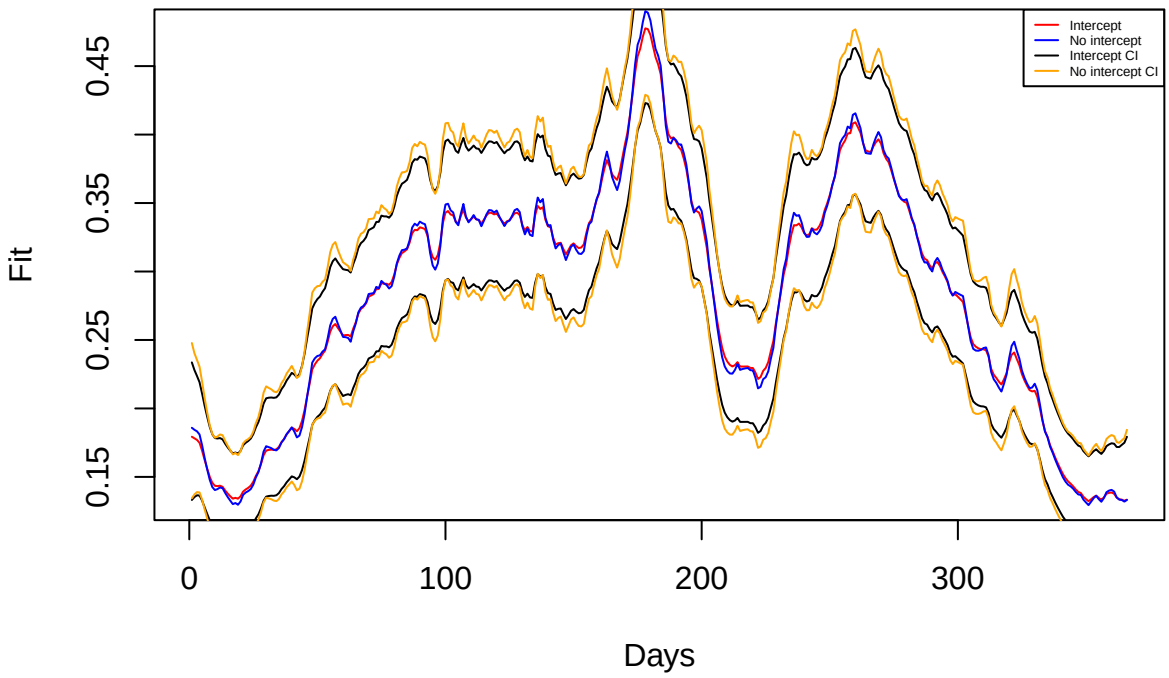


From the plot we can see that the plots for all the options to control.inla look identical, meaning that the results are very robust.

2.3 c)

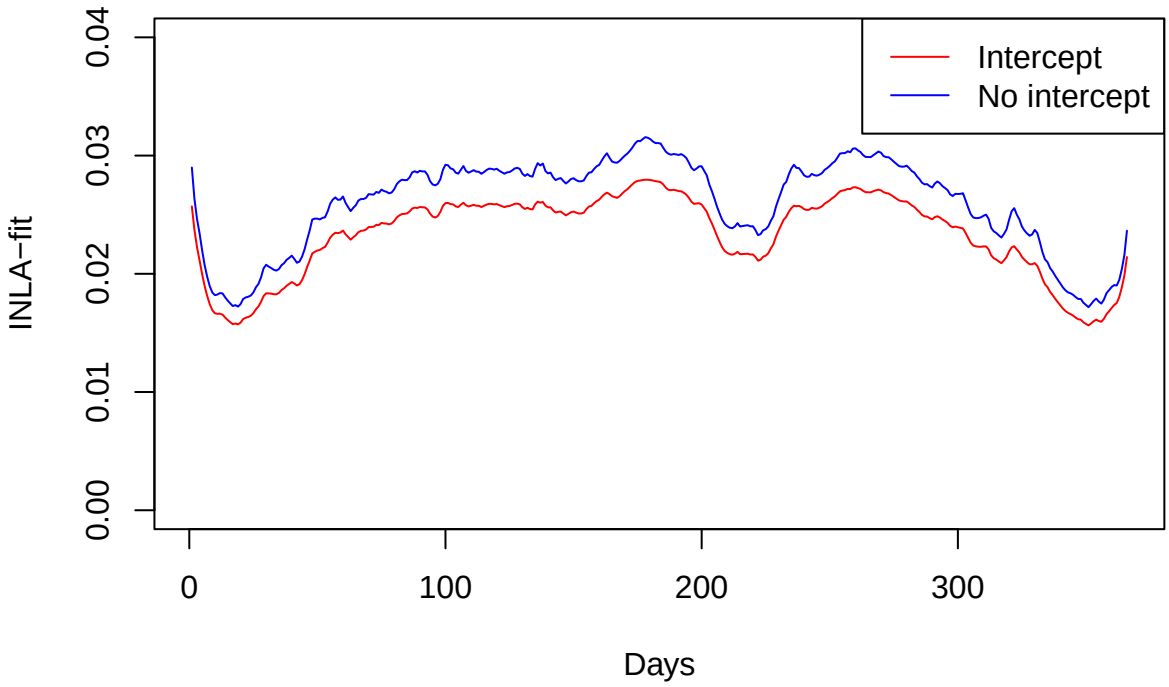
We consider an INLA model with an intercept but where we have the option 'constr = TRUE'.

```
mod_intercept <- inla(n.rain ~ f(day, model="rw1", constr=TRUE),
data=rain, Ntrials=n.years, control.compute=list(config = TRUE),
family="binomial", verbose=F, control.inla=control.inla)
plot(mod_intercept$summary.fitted.values$mean, type = "l", col = "red", xlab = "Days", ylab = "Fit")
lines(mod_intercept$summary.fitted.values$`0.025quant`, type = "l", col = "black")
lines(mod_intercept$summary.fitted.values$`0.975quant`, type = "l", col = "black")
lines(mod_no_intercept$summary.fitted.values$mean, type = "l", col = "blue")
lines(mod_no_intercept$summary.fitted.values$`0.025quant`, type = "l", col = "orange")
lines(mod_no_intercept$summary.fitted.values$`0.975quant`, type = "l", col = "orange")
legend("topright",legend = c("Intercept", "No intercept", "Intercept CI", "No intercept CI"), col = c("
```



From the plot we can see that the two models seems to be very similar. When we do not include an intercept we assume model goes through the origin at $t = 0$, but we have no constraint on the random walk. The variability in this model is from the random walk itself. In the second model, where we include an intercept, we constrain the random walk such that $\sum_{t=1}^{T} u_t = 0$. The random walk for the no intercept model is given as $x_t = x_{t-1} + u_t$, where $x_0 = 0$. Thus we have that $x_t = \sum_{i=1}^{t} u_i$. For the model with an intercept, we have $x_t = a + x_{t-1} + u_t$, $x_0 = a$, for some constant a. In this model we have that $\bar{x}_t = \frac{1}{T}\sum_{t=1}^{T}(a + x_t) = a$. Thus the difference in estimation between the models is that we need to estimate a in the intercept model, while in the no intercept model we estimate the sum of u_t, $1 < t < T$. The difference in the estimated random effects between the intercept and no intercept model is constant equal to a, which indicates that the models are in fact equivalent. We also observe that for the no intercept model, we have $\frac{1}{T}\sum_{t=1}^{T}(x_t) = \alpha$, shown below. This is because the intercept is the empirical mean of the observed values for x_t. We observe that the no intercept has a slightly larger variance than the model including an intercept. This is because we do not put a constraint on the random walk in the no intercept model.

```
plot(mod_intercept$summary.fitted.values$sd, type = "l", ylim = c(0,0.04),col = "red",
     xlab = "Days", ylab = "INLA-fit")
lines(mod_no_intercept$summary.fitted.values$sd, col = "blue")
legend("topright",legend = c("Intercept", "No intercept"), col = c("red", "blue"), lty = 1)
```




```
intercept_fit <- mod_intercept$summary.fitted.values$mean
intercept <- mod_intercept$summary.fixed$mean
x_intercept <- intercept
mean(mod_intercept$summary.random$day$mean+intercept)
```

```
## [1] -0.9855926
```

```
mean(mod_no_intercept$summary.random$day$mean)
```

```
## [1] -0.9860119
```

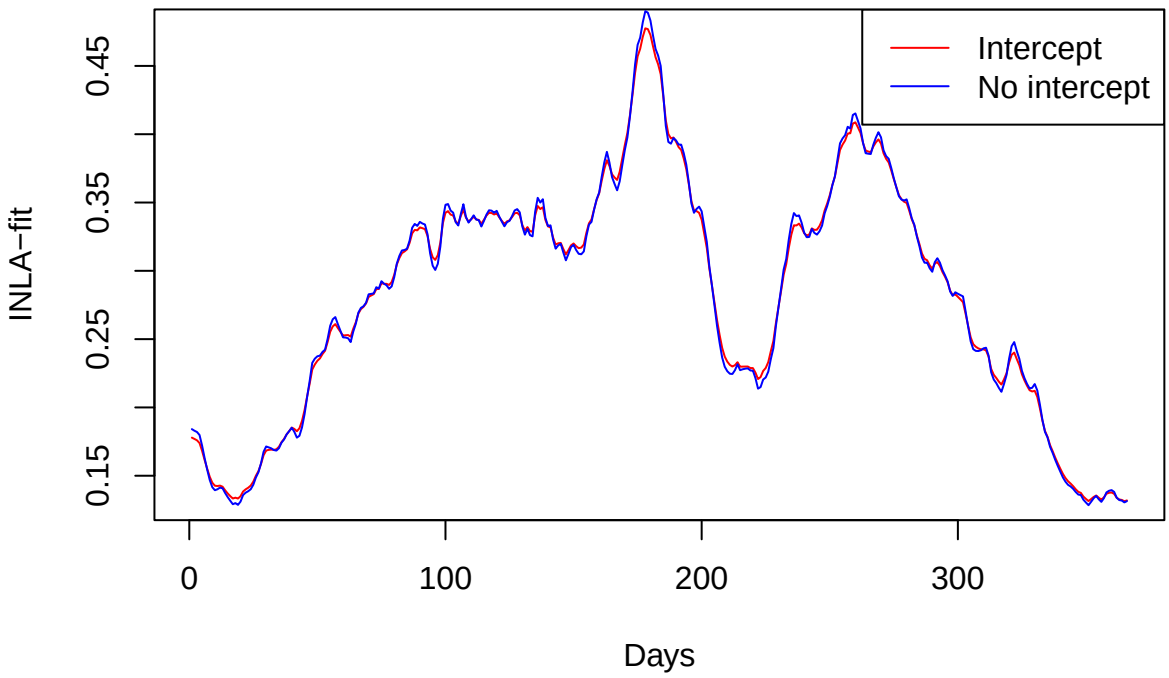
```
#alpha
mean(colMeans(MCMC_block_run$x[interval,]))
```

```
## [1] -0.9909631
```

```
intercept
```

```
## [1] -0.9855926
```

```
plot(expit(mod_intercept$summary.random$day$mean+intercept), type = "l", col = "red",
     xlab = "Days", ylab = "INLA-fit")
lines(expit(mod_no_intercept$summary.random$day$mean), col = "blue")
legend("topright",legend = c("Intercept", "No intercept"), col = c("red", "blue"), lty = 1)
```



3 Problem 3

We will now implement the same model as in Problem 1 and Problem 2, using the R-package RTMB. In order to use RTMB we will first need to compute the negative log-likelihood the joint likelihood $\pi(\boldsymbol{y}, \boldsymbol{x}|\boldsymbol{\theta})$. In this case the model parameter is given by σ_u^2, and we have that

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

We know the distributions for $\boldsymbol{x}|\sigma_u^2$ and $\boldsymbol{y}|\boldsymbol{x}$ from before, and use the R-functions dnorm and dbinom to obtain the negative log likelihoods with model parameters specified from before. MakeADFun produces a new R-function that computes the negative log of the Laplace approximation of the marginal likelihood $\pi(\boldsymbol{y}|\boldsymbol{\theta}) = \int \pi(\boldsymbol{y}, \boldsymbol{x}|\boldsymbol{\theta})\mathrm{d}\boldsymbol{x}$ using automatic differentiation.

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

setwd("C:/Users/marcu/OneDrive/Dokumenter/8. semester/Beregningskrevende statistiske metoder/Project2Be
load("~/8. semester/Beregningskrevende statistiske metoder/Project2Beregningskrevende/rain.rda")
n <- length(rain$n.rain)
alpha <- 2
beta <- 0.05
y <- rain$n.rain
parameters <- list(
  sigma = 1, # standard deviation of white noise variance
```

```
  x = rep(0, n)
)

f <- function(parms) {
  # Make all variables contained in df and parms directly
  # visible.  Note also that the data contained in y is (and must be) passed to
  # f as a global variable
  getAll(parms, warn = FALSE)
  # tell RTMB that y is observed (needed for simulations)
  #y <- OBS(rain$n.rain)
  y <- OBS(y)
  n <- length(y)
  n_t <- rain$n.years

  # Compute negative log of joint likelihood pi(y,x|theta)
  nll <- 0
  # density of x_1, x_2, ..., x_n
  #nll <- nll - dnorm(x[1], 0, sigma, log=TRUE)
  # for (t in 2:n)
  #     nll <- nll - dnorm(x[t], x[t-1], sigma, log=TRUE)
  # # data
  nll_sum <- 0
  nll_sum <- nll_sum - sum(-log(sigma)-(1/(2*sigma^2))*diff(x)^2)
  # for (i in 2:366){
  #   nll_sum <- nll_sum + log(sigma) + (1/(2*sigma^{2}))*(x[i]-x[i-1])^{2}
  # }
  # nll_sum <-
  #   nll <- nll - nll_sum
  nll_sum <- nll_sum - sum(dbinom(y, n_t,expit(x), log=TRUE))
  ADREPORT(pi)
  # return negative joint likelihood
  nll_sum
}
# Compute new function computing (the negative log of)
# the Laplace approximation of the marginal likelihood.
obj <- MakeADFun(func = f, parameters = parameters, random = "x") # random = c("x","mu") same as glmmTM
# The $fn-component is the function computing the Laplace approx
# Each evaluation of this involves an inner optimization
obj$fn(c(0,1))
```

```
## iter: 1  Error in iterate(par) :
##   Newton dropout because inner gradient had non-finite components.
```

```
## [1] NaN
```

```
# This function can be minimised with your favorite optimizer, to find
# the MLEs of the model parameters (mu, sigma, phi) and their
# approximate standard errors based on the (observed) Fisher information
opt <- 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: 1.435518e-13
## iter: 1  mgc: 1.435518e-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: 6.542944e-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: 3.730349e-14
## iter: 1  value: NaN mgc: 8.666403 ustep: 0
## mgc: 8.666403
## iter: 1  mgc: 3.730349e-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: 1.927347e-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: 6.714226e-08 ustep: 1
## iter: 4  mgc: 5.595524e-14
## iter: 1  mgc: 5.595524e-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: 1.162227e-10
## iter: 1  mgc: 1.162227e-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: 5.551115e-13
## iter: 1  mgc: 5.551115e-13
## outer mgc:  0.1543926
## iter: 1  value: -32.53171 mgc: 0.00554609 ustep: 1
## iter: 2  mgc: 7.472988e-09
## iter: 1  mgc: 7.472988e-09
## outer mgc:  0.002012072
```

```
## iter: 1  value: -32.53306 mgc: 7.322176e-05 ustep: 1
## iter: 2  mgc: 1.307177e-12
## iter: 1  mgc: 1.307177e-12
## outer mgc:  1.055901e-06
## iter: 1  value: -32.53306 mgc: 3.840577e-08 ustep: 1
## mgc: 4.796163e-14
## iter: 1  mgc: 1.307177e-12
```

```
sigma2 <- opt$par^2 #Estimated variance
sigma2
```

```
##       sigma
## 0.006961482
```

```
result <- t.test(MCMC_run$sigma2[-c(1:1000)])
lower <- result$conf.int[1]
upper <- result$conf.int[2]
lower
```

```
## [1] 0.1131252
```

```
upper
```

```
## [1] 0.1134452
```

```
mean(MCMC_run$sigma2[-c(1:1000)])
```

```
## [1] 0.1132852
```

We see that the estimated variance using the RTMB function in R produces an estimate for the variance to be 0.00696. This estimate is smaller than the estimate we obtained for σ_u^2 from the MCMC sampler which was $0.1131583 \leq \sigma_u^2 \leq 0.1134777$, with $\alpha = 0.05$. The function sdreport() did not work on our setup, so we do not get information about the uncertainty of the estimate for σ_u^2.