Project 2 - TMA4300 Computer Intensive Statistical Methods

Jostein Aastebøl Aanes, Frederick Nilsen

25.03.2022

Introduction

For this report, we will be looking at a given portion of the Tokyo rainfall data set. The data set contains data gathered from 1951 to 1989. For this report, the response is how many times amount of rainfall exceeded 1mm on a given day, for each of the years. Let $t$ denote the days in a year, i.e. $t = 1, 2, \dots, 366$, then let $y_t$ be the amount of times that it rained more than 1mm on day $t$. Furthermore, let n_t denote the amount of times day $t$ was recorded, which means how many years day $t$ occurred in the given timespan. This means that $n_t = 39$ for all $t$ except for $t = 60$, where $n_{60} = 10$, since February 29th only occurs every fourth year. Lastly, let $\pi(\tau_t)$ be the probability that it rains more than 1mm on day $t$.

We assume that $y_t$ given τ_t is binomially distributed with n_t trials and $\pi(\tau_t)$ probability of success for each trial. We also assume conditionall independence among the $y_t|\tau_t$ for all $t$. Furthermore, τ_t is the logit probability of exceedence. In other words:

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

Exploration of the data set

To start the data exploration, we plot the response $y_t$ as a function of $t$, which can be seen in Figure 1. There is clearly a pattern. One can see how the different seasons affect the amount of rain. During the winter, which is for $t$ close to zero or $t$ close to 366, there is visibly less rain, compared to the summer which is around $t \in [100, 250]$ in which there is considerably more rain. This makes sense as seasons affect the amount of rain. One interpretation of Figure 1, is that if there is a lot of rain in a given period, then there will be less rain in the following period, and the other way around. Notably, one can see that there is a short period around $t = 225$ where there is noticeably less rain compared to the preceding and following period where there is much rain.

```r
load(file = "rain.rda")  #Loading in the Tokyo rainfall data

#Plotting rain as a function of the day.
ggplot(rain, aes(day,n.rain)) + geom_point() +
  ggtitle(TeX(r'($y_t$ plotted against $t$)')) + xlab("t")+
  ylab(TeX(r'($y_t$)'))
```

Furthermore, it is of interest to get a more quantitative summary of the data. We achieve this by using the built in r function "summary()", and the result can be seen below. From the output, we see that $y_t$ has a mean of approximately 11 and the median is 11. The fact that the mean and median are almost equal indicates well balanced data, without any outliers.

```r
summary(rain)
```

```
##       day            n.years          n.rain
##  Min.   :  1.00   Min.   :10.00   Min.   : 0.00
##  1st Qu.: 92.25   1st Qu.:39.00   1st Qu.: 8.00
##  Median :183.50   Median :39.00   Median :11.00
```

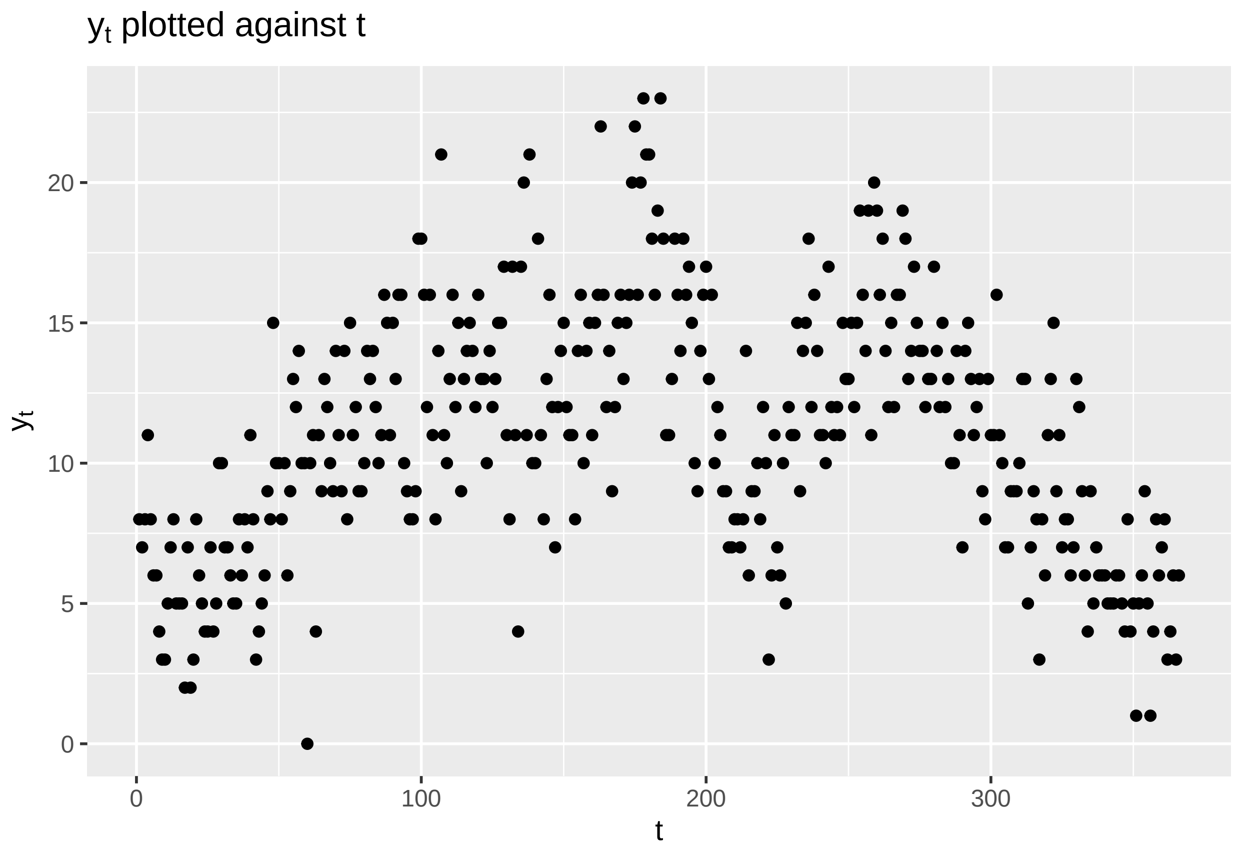


Figure 1: y plotted against t

```
##  Mean   :183.50   Mean   :38.92   Mean   :10.98
##  3rd Qu.:274.75   3rd Qu.:39.00   3rd Qu.:14.00
##  Max.   :366.00   Max.   :39.00   Max.   :23.00
```

Likelihoods and conditionals

Although we now have a general understanding of our data, we would like to further explore some of its statistical properties.

Obtaining likelihood of y_t given $\pi(\tau_t)$

We wish to find $\mathcal{L}(\pi(\tau_t)|y_t)$. By definition

$$\mathcal{L}(\pi(\boldsymbol{\tau})|\boldsymbol{y}) = f(y_1, y_2, ..., y_{366}|\pi(\tau_1), \pi(\tau_2), ..., \pi(\tau_{366}))$$

where f is the joint conditional distribution. Now, we have that $f(y_1, ..., y_{366}|\pi(\tau_1), ..., \pi(\tau_{366})) = f(y_1|\pi(\tau_1), ..., \pi(\tau_{366})) \cdot f(y_2, ..., y_{366}|y_1, \pi(\tau_1), ..., \pi(\tau_{366}))$. Since y_1 is conditionally independent all $\pi(\tau_j)$ for $j \neq 1$, we get that $f(y_1|\pi(\tau_1), ..., \pi(\tau_{366}) = f(y_1|\pi(\tau_1))$. In addition, all y_j where $j \neq 1$ are conditionally independent y_1 and $\pi(\tau_1)$ given $\pi(\tau_j)$, thus $f(y_2, ..., y_{366}|y_1, \pi(\tau_1), ..., \pi(\tau_{366})) = f(y_2, ..., y_{366}|\pi(\tau_2), ..., \pi(\tau_{366}))$. Using the same argument for $t = 2, ..., 366$, one gets that the joint distribution is equal to

$$f(y_1, y_2, ..., y_{366}|\pi(\tau_1), \pi(\tau_2), ..., \pi(\tau_{366})) = \prod_{i=1}^{366} f(y_i|\pi(\tau_i)).$$

Hence

$$\mathcal{L}(\pi(\boldsymbol{\tau})|\boldsymbol{y}) = \prod_{i=1}^{366} \mathcal{L}(\pi(\tau_i)|y_i) = \prod_{i=1}^{366} f(y_i|\pi(\tau_i))$$

which means that

$$\mathcal{L}(\pi(\tau_i)|y_i) = f(y_i|\pi(\tau_i))$$

where $f(y_i|\pi(\tau_i)) = \binom{n_i}{y_i}\pi(\tau_i)^{y_i}(1-\pi(\tau_i))^{n_i-y_i}$.

Establishing the Bayesian hierarchical model

Now, we will apply a Bayesian hierarchical model to the dataset, using a Random walk of order 1 to model the trend as $\tau_t \sim \tau_{t-1} + u_t$ where $u_t \sim \mathcal{N}(0, \sigma_u^2)$ are identically and independently distributed. This in turn implies that $p(\tau_t|\sigma_u^2) \sim \mathcal{N}(\tau_{t-1}, \sigma_u^2)$. We have that

$$p(\boldsymbol{\tau}|\sigma_u^2) = \prod_{t=2}^{T} \frac{1}{\sigma_u^2} e^{-\frac{1}{2\sigma_u^2}(\tau_t - \tau_{t-1})^2}$$

We impose an inverse gamma prior on σ_u^2 as follows

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

We also introduce the notation $\boldsymbol{y} = (y_1, ..., y_{366})^T$, $\boldsymbol{\tau} = (\tau_1, ..., \tau_{366})^T$ and $\boldsymbol{\pi} = (\pi(\tau_1), ..., \pi(\tau_{366}))^T$.

Finding the conditional $p(\sigma_u^2|\boldsymbol{y}, \boldsymbol{\tau})$

We now wish to find the conditional $p(\sigma_u^2|\boldsymbol{y}, \boldsymbol{\tau})$.

Firstly

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

Now, $\boldsymbol{y}$ is conditionally independent σ_u^2 given $\boldsymbol{\tau}$. Thus, $p(\boldsymbol{y}|\sigma_u^2, \boldsymbol{\tau}) = p(\boldsymbol{y}|\boldsymbol{\tau})$. We know that $p(\boldsymbol{y}|\boldsymbol{\tau})$ does not include any terms with σ_u^2. Therefore, we get

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

This becomes in turn

$$p(\sigma_u^2|\boldsymbol{y}, \boldsymbol{\tau}) \propto \prod_{t=2}^{T} \frac{1}{\sigma_u} \exp\left(-\frac{1}{2\sigma_u^2}(\tau_t - \tau_{t-1})^2\right) \times \frac{\beta^\alpha}{\Gamma(\alpha)} \left(\frac{1}{\sigma_u^2}\right)^{\alpha+1} \exp\left(-\frac{\beta}{\sigma_u^2}\right)$$

Disregarding the constant $\beta^\alpha/\Gamma(\alpha)$, this simplifies to

$$\left(\frac{1}{\sigma_u}\right)^{T-1} \exp\left(\sum_{t=2}^{T} \frac{-1}{2\sigma_u^2}(\tau_t - \tau_{t-1})^2 + \frac{-\beta}{\sigma_u^2}\right) \left(\frac{1}{\sigma_u^2}\right)^{\alpha+1}$$

$$= (\sigma_u^2)^{-(\alpha + \frac{T-1}{2})-1} \exp\left(\frac{-(\beta + \frac{1}{2}\sum_{t=2}^{T}(\tau_t - \tau_{t-1})^2)}{\sigma_u^2}\right)$$

Which is the pdf of an inverse gamma distribution with parameters

$$\alpha' = \alpha + \frac{T-1}{2}, \quad \beta' = \beta + \frac{1}{2}\sum_{t=2}^{T}(\tau_t - \tau_{t-1})^2)$$

Sampling with a Monte Carlo Markov Chain (MCMC)

With our relations derived so far, we have almost everything needed to implement a MCMC sampler, using the Metropolis-Hastings algorithm to sample $\boldsymbol{\tau}$ and Gibbs algorithm to sample σ_u^2. All we need before the implementing a MCMC sampler, is an expression for the acceptance probability.

Acceptance probability for MH-step

Let $\mathcal{I} \subseteq \{1, \dots, 366\}$ be a set of time indices, and let $\boldsymbol{\tau}'_{\mathcal{I}}$ be proposed values for $\boldsymbol{\tau}_{\mathcal{I}}$. Furthermore, $\boldsymbol{\tau}_{-\mathcal{I}} = \boldsymbol{\tau}_{\{1,\dots,366\}\setminus\mathcal{I}}$ be a subset of $\boldsymbol{\tau}$ which includes all τ_t except for those with indices in $\mathcal{I}$. We have that the prior proposal distribution is

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

We consider it known that the acceptance probability is given by

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

It is preferable to simplify the last expression as much as possible.

The ratio including the prior proposal distribution $Q(\cdot)$ becomes

$$\frac{p(\boldsymbol{\tau}_{\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}}, \sigma_u^2)}{p(\boldsymbol{\tau}'_{\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}}, \sigma_u^2)}.$$

Considering now the numerator for the first factor,

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

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

where the last relations holds since we assume conditional independence among all $y_t|\tau_t$. Similarly for the denominator we will end up with

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

Taking the divison allows for several cancellations

$$\frac{p(\boldsymbol{y}_{\mathcal{I}}|\boldsymbol{\tau}'_{\mathcal{I}})\cdot p(\boldsymbol{y}_{-\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}})\cdot p(\boldsymbol{\tau}'_{\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}},\sigma_u^2)}{p(\boldsymbol{y}|\boldsymbol{\tau}_{-\mathcal{I}},\sigma_u^2)}\quad\frac{p(\boldsymbol{y}|\boldsymbol{\tau}_{-\mathcal{I}},\sigma_u^2)}{p(\boldsymbol{y}_I|\boldsymbol{\tau}_{\mathcal{I}})\cdot p(\boldsymbol{y}_{-\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}})\cdot p(\boldsymbol{\tau}_{\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}},\sigma_u^2)} = \frac{p(\boldsymbol{y}_I|\boldsymbol{\tau}'_{\mathcal{I}})\cdot p(\boldsymbol{y}_{-\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}})\cdot p(\boldsymbol{y}|\boldsymbol{\tau}_{-\mathcal{I}},\sigma_u^2)\cdot p(\boldsymbol{\tau}'_{\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}},\sigma_u^2)}{p(\boldsymbol{y}_I|\tau_{\mathcal{I}})\cdot p(\boldsymbol{y}_{-\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}})\cdot p(\boldsymbol{y}|\boldsymbol{\tau}_{-\mathcal{I}},\sigma_u^2)\cdot p(\boldsymbol{\tau}_{\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}},\sigma_u^2)}$$

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

Multiplying this with $\frac{Q(\boldsymbol{\tau}_{\mathcal{I}}|\dots)}{Q(\boldsymbol{\tau}'_{\mathcal{I}}|\dots)}$, the final factors cancel out:

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

Thus, we get that the acceptance probability is given by

$$\alpha = \min\left(1,\frac{p(\boldsymbol{y}_{\mathcal{I}}|\boldsymbol{\tau}'_I)}{p(\boldsymbol{y}_{\mathcal{I}}|\boldsymbol{\tau}_I)}\right)$$

Now, we find an expression for the acceptance probability through logscaling.

$$\alpha = \min\left\{1,\frac{\prod_{i\in I}\pi(\tau'_i)^{y_i}(1-\pi(\tau'_i))^{n_i-y_i}}{\prod_{i\in I}\pi(\tau_i)^{y_i}(1-\pi(\tau_i))^{n_i-y_i}}\right\}$$

We have that

$$\ln\left(\prod_{i\in I}\pi(\tau'_i)^{y_i}(1-\pi(\tau'_i))^{n_i-y_i}\right) = \sum_{i\in I} y_i\tau'_i + n_i\ln(1-\pi(\tau'_i)).$$

Thus, we get that

$$\alpha = \min\left(1,\exp\left(\sum_{i\in I} y_i(\tau'_i-\tau_i)+n_i(\ln(1-\pi(\tau'_i))-\ln(1-\pi(\tau_i)))\right)\right)$$

Which reduces to

$$\alpha = \min\left(1,\exp\left(\sum_{i\in\mathcal{I}} y_i(\tau'_i-\tau_i)+n_i\log\left(\frac{1+e^{\tau_i}}{1+e^{\tau'_i}}\right)\right)\right)$$

This will be our expression for the acceptance probability. The expression is used since it is less computationally intensive than calculating two binomial densities, and finding the ratio between them. A comparison between the newly obtained expression and the initial expression is shown in Figure 2, where the optimized expression clearly outperforms the standard expression.

Deriving precision matrix $Q = Prec(\boldsymbol{\tau}|\sigma_u^2)$

We have that

$$p(\boldsymbol{\tau}|\sigma_u^2) = \prod_{t=2}^{T}\frac{1}{\sigma_u}\exp\left(-\frac{1}{2\sigma_u^2}(\tau_t-\tau_{t-1})\right)$$

$$= \left(\frac{1}{\sigma_u}\right)^{T-1}\exp\left(-\frac{1}{2\sigma_u^2}\sum_{t=2}^{T}(\tau_t-\tau_{t-1})\right) = \left(\frac{1}{\sigma_u}\right)^{T-1}\exp\left(-\frac{1}{2}\boldsymbol{\tau}Q\boldsymbol{\tau}^t\right)$$

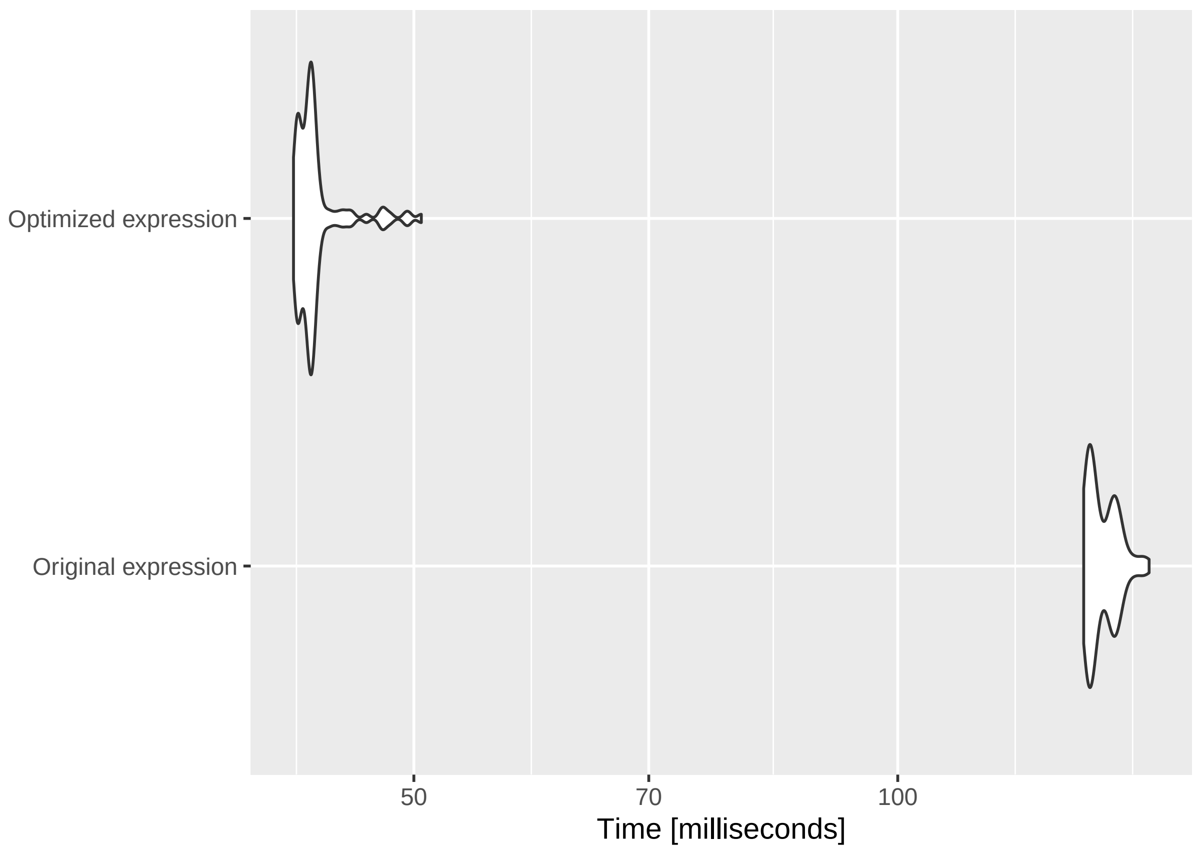


Figure 2: Distribution plot of the computational time for the original expression compared to the optimized expression. Plot is made using the R-package microbenchmark.

where $\boldsymbol{\tau}^t$ means the transposed of $\boldsymbol{\tau}$. Here, one has that

$$Q = \frac{1}{\sigma_u^2}\begin{bmatrix} 1 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \dots & 0 \\ 0 & -1 & 2 & -1 & \vdots \\ \vdots & & \ddots & \ddots & \\ 0 & \dots & 0 & -1 & 1 \end{bmatrix}$$

For a single-site MCMC algorithm one then gets that the mean value μ_t for $\tau_t|\sigma_u^2$ is given by

$$\mu_t = \begin{cases} \tau_2, \quad t=1 \\ \frac{1}{2}(\tau_{t-1}+\tau_{t+2}), \quad t \neq 1 \text{ and } t \neq T \\ \tau_{T-1}, \quad t = T \end{cases}$$

and that the variance is

$$\begin{cases} \sigma_u^2, \quad t=1 \\ \frac{1}{2}\sigma_u^2, \quad t \neq 1 \text{ and } t \neq T \\ \sigma_u^2, \quad t = T. \end{cases}$$

We can now start by determining initial values for $\boldsymbol{\tau}$. Since our Bayesian hierarchical model uses a random walk of order 1, all $\tau_i \sim \mathcal{N}(\tau_{i-1}, \sigma_u^2)$. We initialize all τ_i as $\mathcal{N}(0, \widehat{\sigma_u^2})$, where $\widehat{\sigma_u^2}$ is estimated from the data. We find be empirical variance of the logit of $\widehat{\pi(y/n)} =$ `n.rain/n.years` as follows

```r
empirical_pi = rain$n.rain/rain$n.years
sigma_0_est = var(log(empirical_pi)/(1-log(empirical_pi)), na.rm = TRUE)
```

Running the code above results in $\widehat{\sigma_u^2} = 0.0067$, so that a reasonable initial value is 0.007 for the variance. Thus, we initialize $\boldsymbol{\tau}_t^{(0)} \sim \mathcal{N}(0, 0.007), \forall\, t$ for iteration 0.

Single-site MCMC implementation

Before implementing the MCMC algorithm, we define some helper functions shown below.

```r
rgamma_inv <- function(n, alpha, beta){
  # Inverse gamma sampler. Simply samples from Gamma-distribution
  # and returns its reciprocal
  return(1/rgamma(n, shape=alpha, rate=beta))
}

expit <- function(tau){
  # Expit / inverse logit function
  return( 1 / (1+exp(-tau )))
}
```

One detail to keep in mind is that we specify the `rate` as β when sampling from the gamma distribution. This is since the reciprocal of a gamma distribution with *scale* β corresponds to an inverse gamma distribution with *rate* β. We want the inverse gamma prior to have about 95% of its density between 0.01 and 0.25. We can determine the correct sampling method by plotting these histograms as shown in Figure 3.

```r
par(mfrow=c(1,2))
samps_rate = 1/rgamma(5000, shape=2, rate=0.05)
samps_scale = 1/rgamma(5000, shape=2, scale=0.05)
hist(samps_scale, breaks=1000, xlim=c(0,60), main=TeX("$\\beta$ as scale"))
hist(samps_rate, breaks=500, xlim=c(0,0.5), main=TeX("$\\beta$ as rate"))
```

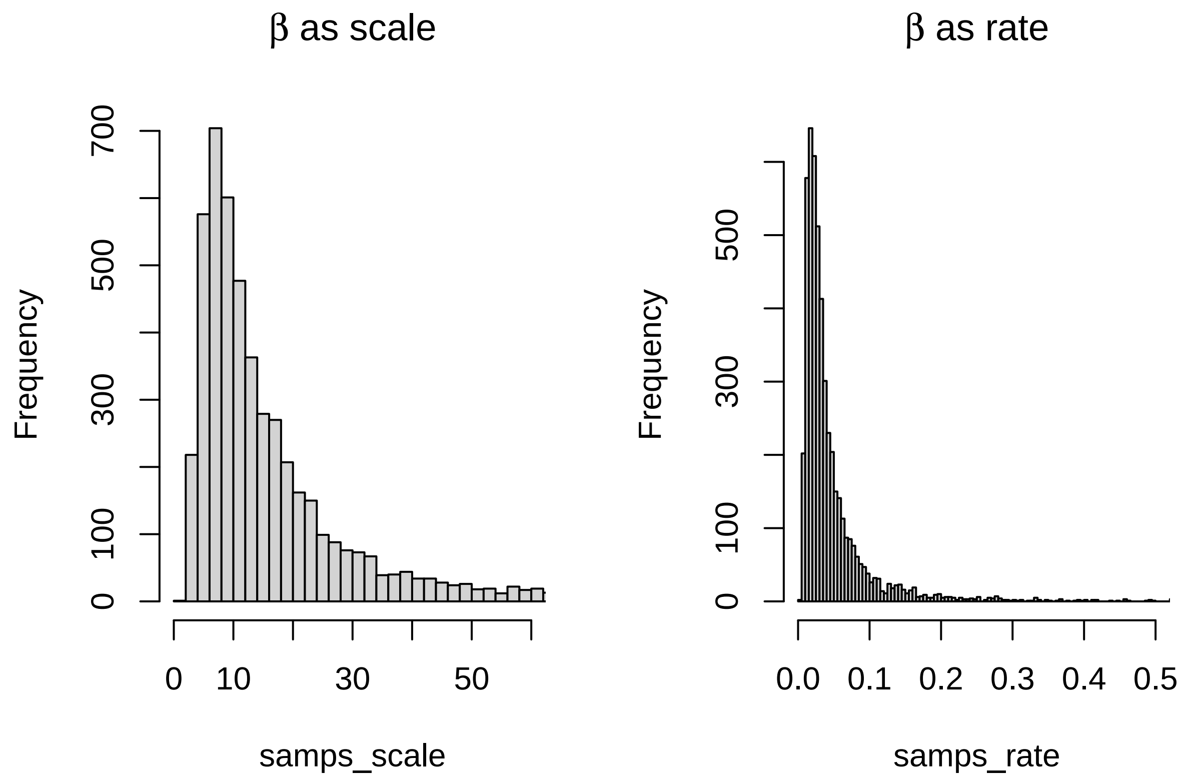


Figure 3: Comparison of histograms for inverse gamma using either rate or scale as β. From the x-axis, we clearly see that the rate parameter produces the wanted results.

We may now implement the Metropolis-Hastings step. This step consists of, for each t, sampling the proposal τ_t' with mean and variance as previously described. If this τ_t' is such that a sampled $u \sim \text{Unif}(0,1)$ is *less than* the acceptance probability $\alpha(\tau_t'|\tau_t, \sigma_u^2, \boldsymbol{y})$, we accept the step, i.e. $\tau_t \leftarrow \tau_t'$. We also store the acceptance rate for later analysis.

```r
mh_step <- function(curr_tau, sigma_u, k){
  # Metropolis-Hastings step for MCMC sampler
  # tau     : Matrix storing each sample value for each iteration
  # sigma_u : Standard deviation of the prior
  # k       : Iteration number, 1 <= k <= 50 000
  # returns : list of new tau and acceptance rate for this iteration

  tau_size = length(curr_tau) # = 366
  accepted = 0

  normal_samples = rnorm(tau_size)
  uniform_samples = runif(tau_size)

  # First tau (t=1)
  first_proposal = curr_tau[2] + sigma_u * normal_samples[1]
  # Get acceptance probability
  acceptance_probability = exp(rain$n.rain[1] * (first_proposal - curr_tau[1])+
                               rain$n.years[1] * log((1+exp(curr_tau[1]))/(
                                 1+exp(first_proposal))))
  # Update tau if acceptance criteria is satisfied
  curr_tau[1] = ifelse(uniform_samples[1] < acceptance_probability,
                       first_proposal, curr_tau[1])
  accepted = accepted + ifelse(uniform_samples[1] < acceptance_probability, 1,0)

  # Do the same for all t except first and last (t != 1, t != 366)
  for (k in 2:(tau_size - 1)) {
    proposal = 1/2 *(curr_tau[k - 1] + curr_tau[k + 1]) +
      normal_samples[k] * sigma_u/sqrt(2)
    acceptance_probability = exp(rain$n.rain[k] * (proposal - curr_tau[k]) +
                                 rain$n.years[k] * log((1+exp(curr_tau[k]))/(
                                   1+exp(proposal))))
    curr_tau[k] = ifelse(uniform_samples[k] < acceptance_probability,
                         proposal, curr_tau[k])
    accepted = accepted + ifelse(uniform_samples[k] < acceptance_probability,
                                 1, 0)
  }

  # Same procedure for last tau (t=366)
  last_proposal = curr_tau[tau_size - 1] + sigma_u * normal_samples[tau_size]
  acceptance_probability = exp(rain$n.rain[tau_size] * (last_proposal -
                                                        curr_tau[tau_size]) +
                               rain$n.years[tau_size] * log(
                                 (1+exp(curr_tau[k]))/(1+exp(last_proposal))
                                 )
                             )
  curr_tau[tau_size] = ifelse(uniform_samples[tau_size]< acceptance_probability,
                              last_proposal, curr_tau[tau_size])
  accepted = accepted +ifelse(uniform_samples[tau_size]< acceptance_probability,
                              1, 0)
  return(list(tau = curr_tau, accept = (accepted/tau_size)))
}
```

We can now apply all the defined functions into a complete single-site MCMC sampler:

```r
sample_mcmc <- function(tau_0, alpha, beta, nsteps=50000){
  # MCMC Sampling algorithm for Tokyo rainfall dataset
  # tau_0   : 366-length vector of initial tau values
  # alpha   : Shape parameter for sigma_u^2
  # beta    : Scale parameter for sigma_u^2
  # nsteps  : Number of MCMC iterations, defaults to 50.000

  # Intializations
  start = proc.time()[3] # To measure time
  tau = array(c(0), dim=c(length(tau_0), nsteps))
  current_tau = tau_0 # Fill initial values
  sigma_u = sqrt(rgamma_inv(1, alpha=alpha, beta=beta))
  accepted = 0
  variances = c()
  # Progress bar initialization
  pb = txtProgressBar(min = 0, max = nsteps, initial = 1)

  # Iterate nstep times
  for(k in 1:nsteps){
    setTxtProgressBar(pb,k) # Update progress bar

    # MH step:
    mh_res = mh_step(current_tau, sigma_u, k)
    current_tau = mh_res$tau # Updates main sample matrix
    tau[, k] = current_tau
    accepted = accepted + mh_res$accept

    # Gibbs step:
    variances[k] = rgamma_inv(1, alpha=alpha + (length(current_tau)-1)/2,
                              beta=beta + 0.5 * sum(diff(current_tau)^2))
    sigma_u = sqrt(variances[k])
    # Note: Variances are stored to list in each iteration
    #       so we can do traceplots on variance

  }
  close(pb)

  # Return tau samples, variance samples, runtime in seconds and acceptance rate
  res = list(
    tau        = tau,
    runtime    = proc.time()[3]-start,
    sigma_u    = variances,
    acceptance = accepted/nsteps
    )

  return(res)
}
result = sample_mcmc(tau_0=rnorm(366, 0, sqrt(0.007)),
                     alpha=2, beta=0.05, nsteps=50000)
```

```
## ================================================================================
```

This run had a runtime of 112.26 seconds with acceptance rate 0.917.

We now wish to determine the performance of the sampler, and introduce some evaluation functions. In particular, we would like to plot trace plots of certain days and the sampled variance, i.e. the evolution of the samples for each iteration. We also implement plots of the histograms with 95 % credible intervals, in addition to an autocorrelation plot.

These functions are implemented below.

```r
# Methods to remove burn-in from results
remove_burnin_tau <- function(tau){
  burnin = 500
  return(tail(tau[, ], n=50000-burnin))
}
remove_burnin_sigma <- function(sigma){
  burnin = 500
  return(tail(sigma, n=50000-burnin))
}

traceplot <- function(result, day=NA, xlim=NULL){
  # Plots trace plot of MCMC results
  # result : Large list containing values for both tau and sigma_u^2
  # day    : int specifying day (when plotting tau) or NA (plotting sigma_u^2)
  # xlim   : Specifies limits to plot trace plot within, defaults to [0,50000]

  if(is.na(day)){ # Sigma_u^2
    plot(result$sigma_u, type="l", xlab="Iteration number", xlim=xlim,
         ylab="", main=TeX(r'(Trace plot of $\sigma_u^2$)'))
  }
  else{ # tau(t=day)
    plot(result$tau[day,], type="l", xlab="Iteration number", xlim=xlim,
         ylab="", main=TeX(sprintf(
           r'(Trace plot of $\tau$ at $t = %d$)', day)))
  }
}

plot_hist <- function(result, day=NA){
  # Plots histogram of MCMC results
  # result : Large list containing values for both tau and sigma_u^2
  # day    : int specifying day (when plotting tau) or NA (plotting sigma_u^2)
  # returns: ggplot object. Returns so it can be passed to subplot method

  # First, remove burn-in
  result$tau = remove_burnin_tau(result$tau)
  remove_burnin_sigma = remove_burnin_sigma(result$sigma_u)

  # Compute credible interval
  ci_res = if(is.na(day)) ci(result$sigma_u, ci=0.95) else
    ci(result$tau[day, ], ci=0.95)
  # Plotting parameters used by ggplot, common for all histograms
  plot_params <- list(
    geom_vline(aes(xintercept = ci_res$CI_low, color="95% credible interval")),
    geom_vline(xintercept = ci_res$CI_high, color="red"),
    theme(legend.position = "top")
  )

  if(is.na(day)){ # Sigma_u^2
    df = as.data.frame(result$sigma_u)
    return(ggplot(data=df, aes(x=result$sigma_u)) + geom_histogram(
      color="black", alpha=0.4, binwidth = 1e-3) + plot_params +
      labs(x=TeX("$\\sigma_u^2$"), color=""))
  }
  else{ # tau(t=day)
    df = as.data.frame(result$tau[day, ])
    return(ggplot(data=df, aes(x=result$tau[day, ])) + geom_histogram(
      color="black", alpha=0.4, binwidth = 0.04) + plot_params +
      labs(x=TeX(paste("$\\tau_t$, $t=", day)), color=""))
  }
}

compare_to_data <- function(result, plotMean=TRUE){
  # Compares pi(tau_t) to empirical pi(tau_t) for t = 1, ..., 366
  # result   : Large list as in other plotting functions
  # plotMean : bool determining if we want the mean (default) or just
  #            the last iteration
  #

  # First, remove burn-in
  result$tau = remove_burnin_tau(result$tau)
  remove_burnin_sigma = remove_burnin_sigma(result$sigma_u)

  # Plots the probability of rain alongside the sampled values for each day.
  pi_data = rain$n.rain/rain$n.years
  pi_mcmc = if(plotMean) expit(rowMeans(result$tau)) else expit(
    tail(t(result$tau), n=1)[,]
    )
  ci_low = c()
  ci_high = c()
  for(i in 1:length(result$tau[, 1])){
    ci_all = ci(expit(result$tau[i,]), ci=0.95)
    ci_low[i] = ci_all$CI_low
    ci_high[i] = ci_all$CI_high
  }
  plot(pi_data, type="l", col="grey", ylim=c(0,1))
  lines(pi_mcmc, type="l", col="red")
  lines(ci_low, type="l", col="blue")
  lines(ci_high, type="l", col="blue")
  legend('topright', lty=1, cex=0.8, col=c("grey", "red", "blue"),
        legend=c("True observations","MCMC predictions","95% credible interval")
  )
}
```

We choose to look at the results for σ_u^2, and τ_t for the days $1, 201$, and 366. Figure 4 shows the trace plots of σ_u^2 and each of the three τ for all 50000 samples. We see some burn-in from most plot at the very start, but other than that we observe stationary noise for all cases.

If we look at the first 1000 iterations, we are able to easier determine around what values we can expect burn-in. This is done in Figure 5. The burn-in seem to only be in around $[0, 200]$. Thus, for good measure, a

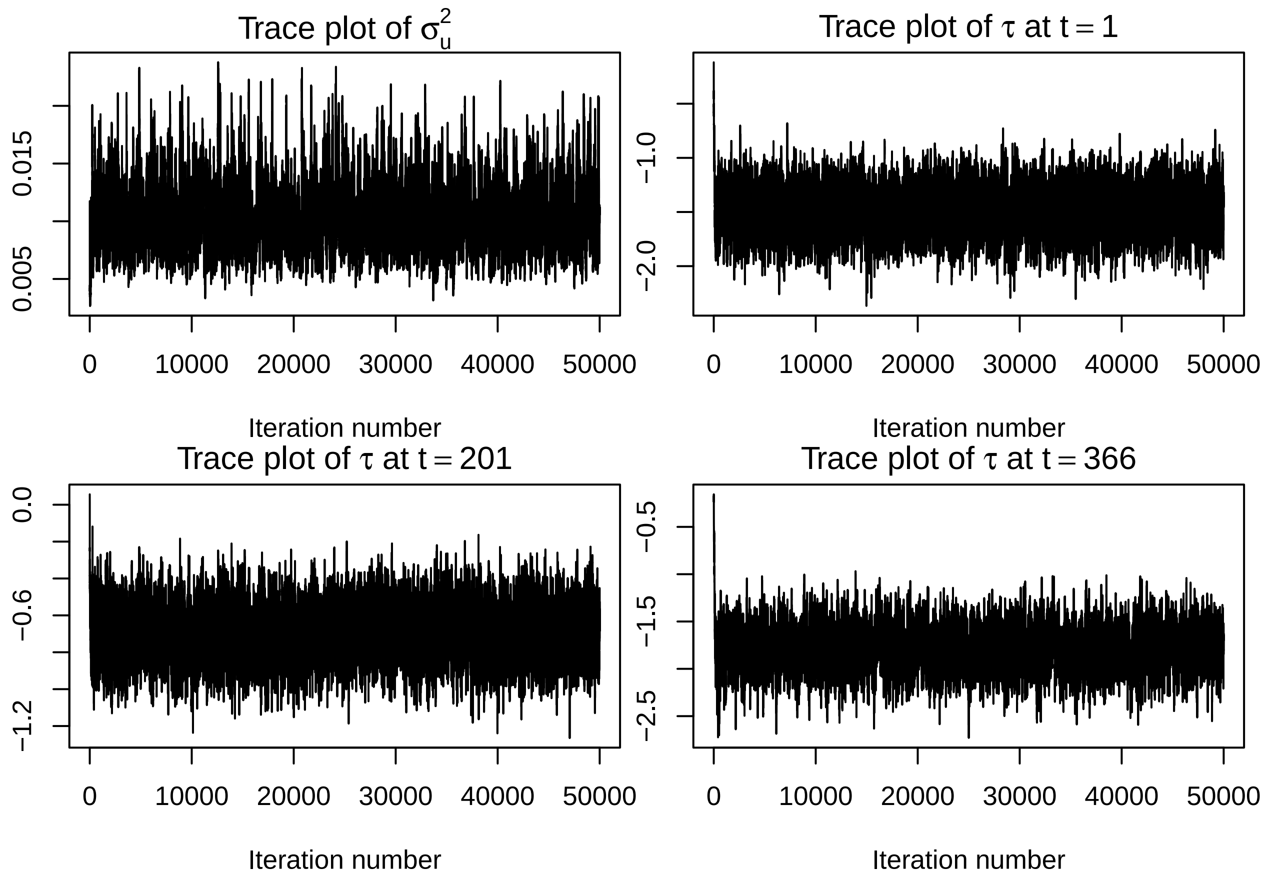


Figure 4: Trace plots of σ_u^2 and τ_t for $t = 1, 201, 366$ for all iterations. Although there is some burn-in deviations for the earliest iterations, they all generally look like white noise, which is the intended behavior.

burn-in of about 500 should be more than enough. For the other plots and analyses, we exclude the first 500 iterations.

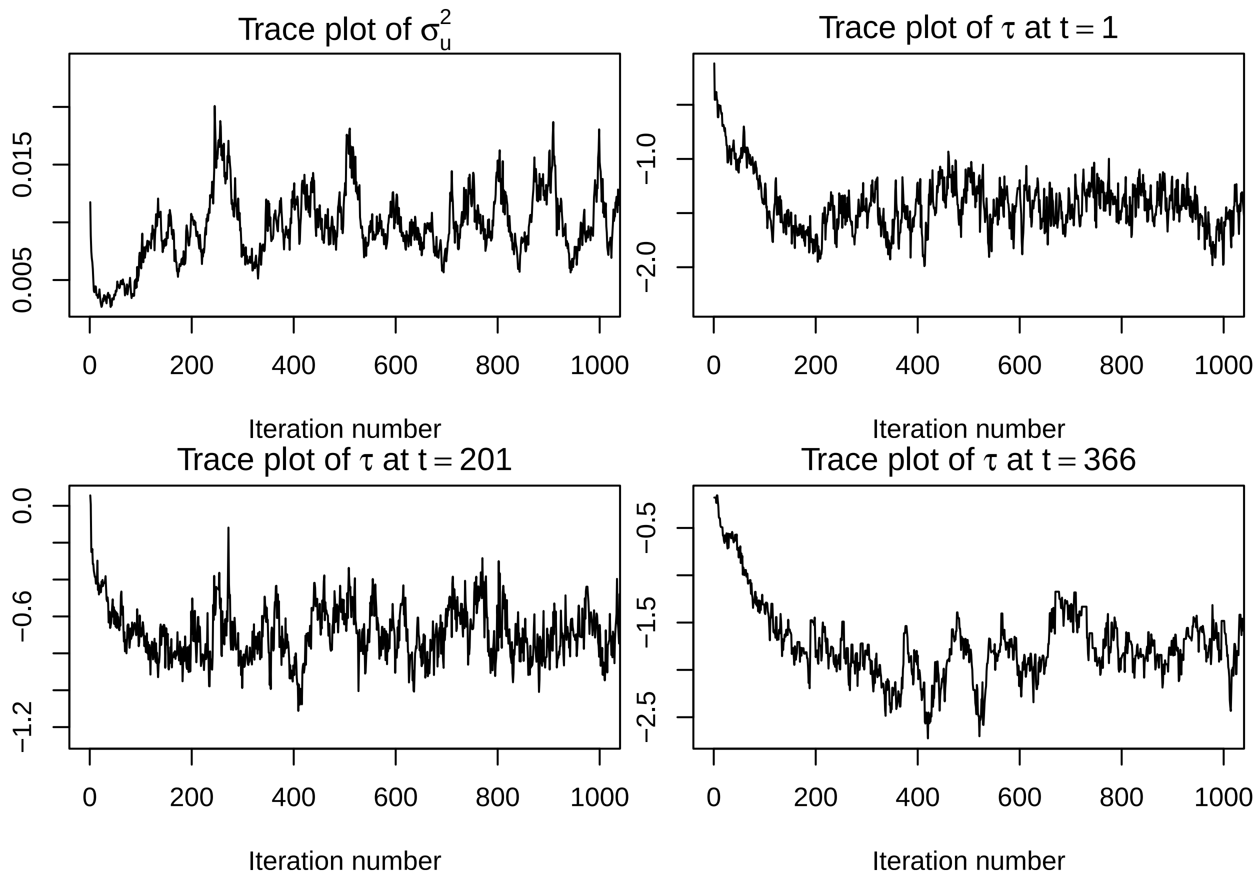


Figure 5: Trace plots of the same parameters for the 1000 first values. We see clear burn-in around the 100 first iterations, but the rest seem like noise.

The histogram plots for the same four values are given in Figure 6. σ_u^2 seem to follow the density of an inverse gamma distribution, and the τ_t follow a normal distribution as expected. We can also compute the expit of the centers from the τ densities and compare them to Figure 1. Doing so reveals smaller probabilities for exceedence at $t = 1, 366$ and a higher probability for exceedence at $t = 201$. This coincides well with Figure 1, seeing as day 201 experiences more rain than the start and end of the year.

We would like to also compare the MCMC predictions to the empirical ratio `n.rain/n.years`, which is done in Figure 7. The resulting plot shows that the 95% credible interval captures the true values for most days with few exceptions. Generally, the estimated distribution from the MCMC is considerably less jagged than the empirical data, which indicates that it captures the trend and can be less prone to noise.

Lastly, we also plot the autocorrelation of the parameters as a function of lag in Figure 8.

These plots show that there is strong correlation in the iterates for both σ_u^2 and all three values of τ_t.

MCMC blocking preliminaries

Although the single-site MCMC provided promising results, it could be of interest to see if we can create a blocking algorithm to further improve performance and/or the predictions.

Consider the partionining of $\boldsymbol{\tau} = [\boldsymbol{\tau}_{\mathcal{I}}, \boldsymbol{\tau}_{-\mathcal{I}}]^T$, where $\mathcal{I}$ represents the index set as previously described. This is

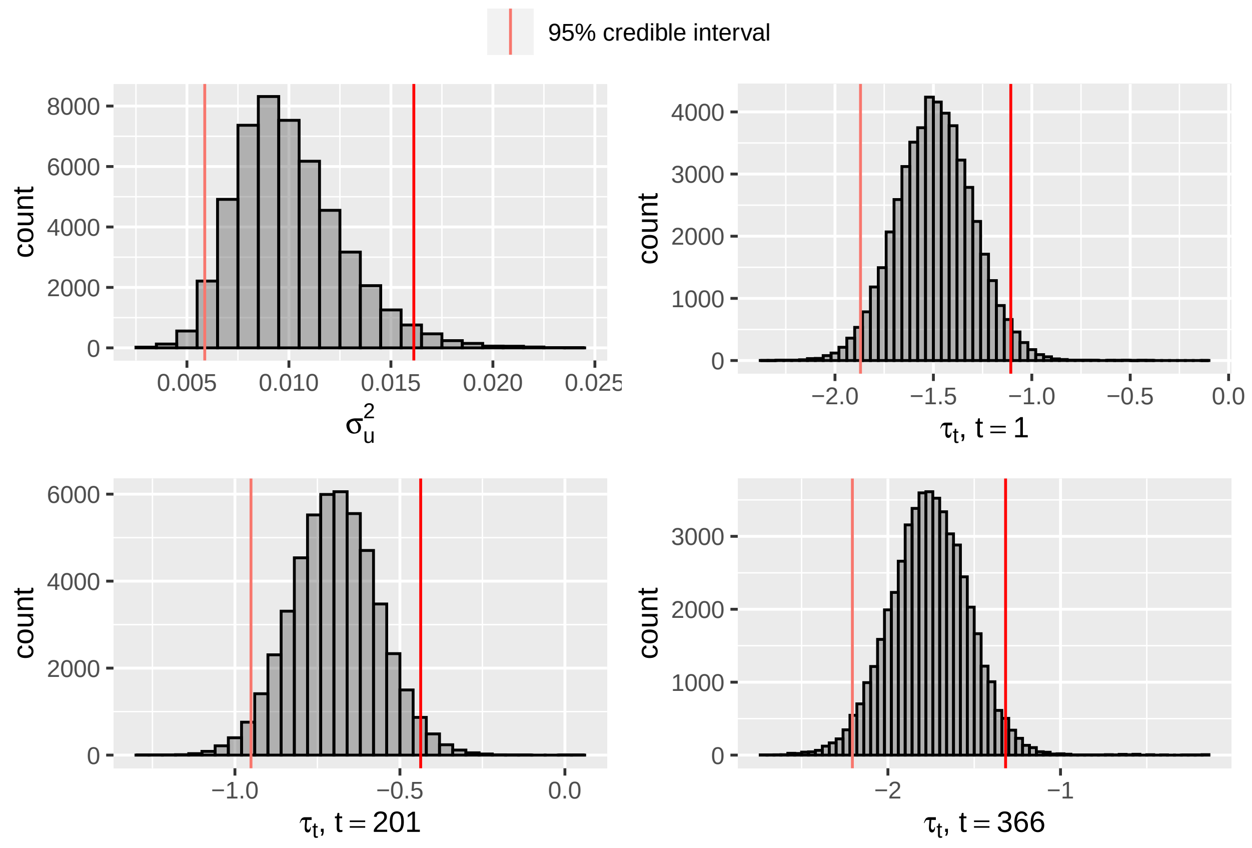


Figure 6: Histograms of σ_u^2 and τ_t for $t = 1, 201, 366$ with their corresponding 95% credible intervals.

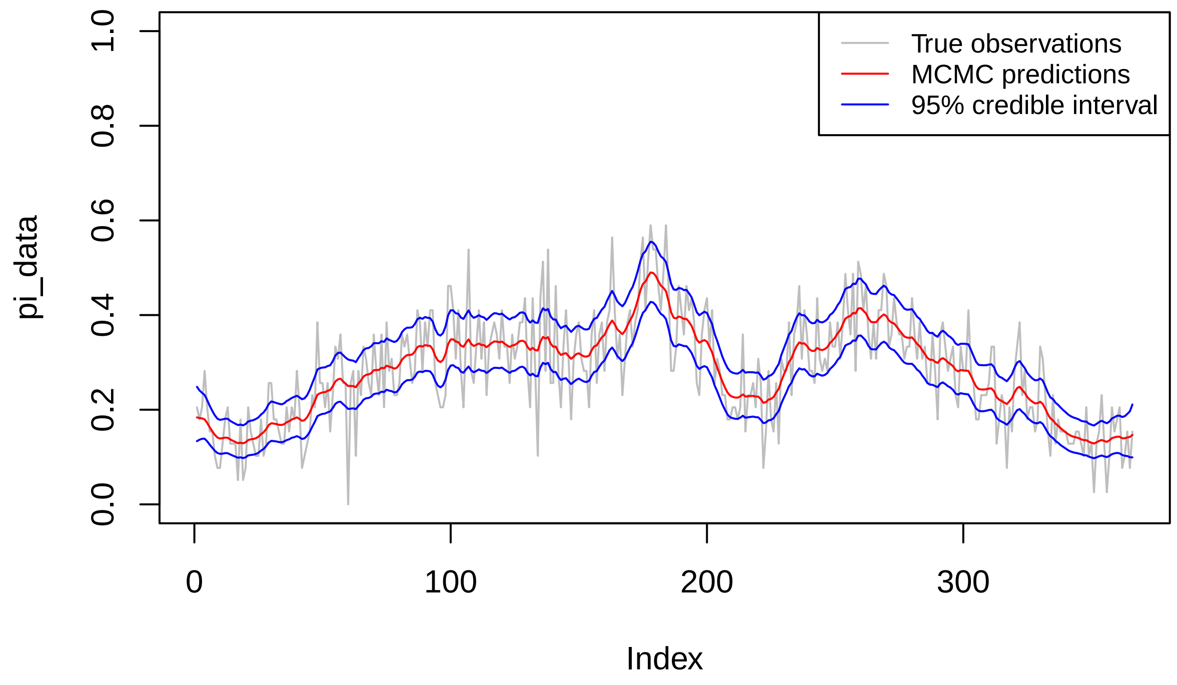


Figure 7: Comparison between of true probabilities versus the sampled $\pi(\tau_t)$ for each day. The samples represent the mean of all iterations, excluding burn-in.

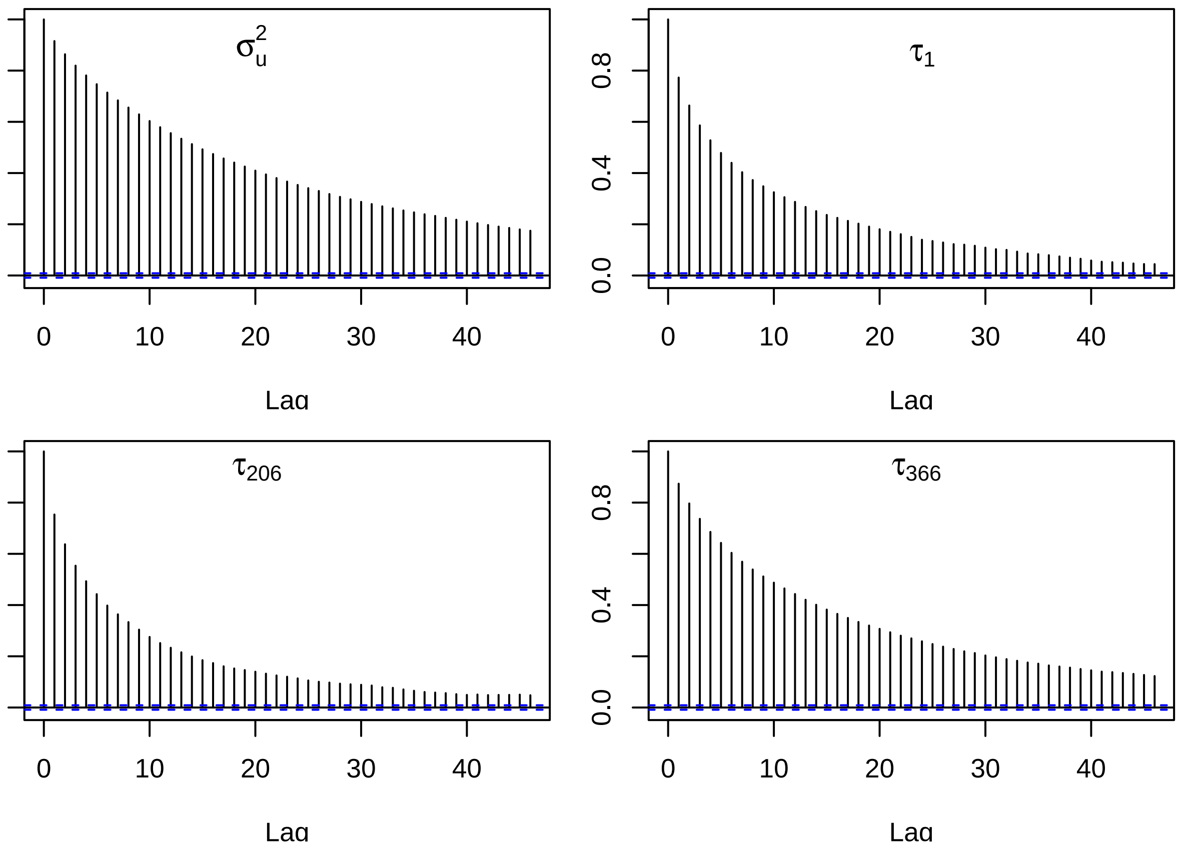


Figure 8: Autocorrelation plot of the four parameters as a function of lag.

multivariate normally distributed as

$$\boldsymbol{\tau} = [\boldsymbol{\tau}_{\mathcal{I}}, \boldsymbol{\tau}_{-\mathcal{I}}]^T \sim MVN\left([\boldsymbol{\mu}_I, \boldsymbol{\mu}_{-I}]^T, \left(\text{Prec}(\boldsymbol{\tau}|\sigma_u^2)\right)^{-1}\right),$$

where $\text{Prec}(\boldsymbol{\tau}|\sigma_u^2)$ is the already defined precision matrix. We have that

$$\boldsymbol{\tau}'_{\mathcal{I}} \sim f(\boldsymbol{\tau}_{\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}})$$

where $\boldsymbol{\tau}'_{\mathcal{I}}$ is the proposed value for $\boldsymbol{\tau}_{\mathcal{I}}$. Since the Bayesian hierarchical model uses a random walk of order one, if $\mathcal{I} = \{a, a+1, \dots, b-1, b\}$, then $\boldsymbol{\tau}'_{\mathcal{I}} \sim f(\boldsymbol{\tau}_{\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}})$ becomes $\boldsymbol{\tau}'_{\mathcal{I}} \sim f(\boldsymbol{\tau}_{\mathcal{I}}|\tau_{a-1}, \tau_{b+1})$. Let M be the length of $\mathcal{I}$.

Furthermore:

$$\boldsymbol{\mu}_{\mathcal{I}|-\mathcal{I}} = -Q_{\mathcal{I},\mathcal{I}}^{-1} Q_{\mathcal{I},-\mathcal{I}} \boldsymbol{\tau}_{-\mathcal{I}}.$$

$$Q_{\mathcal{I}|-\mathcal{I}} = Q_{\mathcal{I},\mathcal{I}}$$

We have three different cases to consider: $a = 1$, $a \neq 1$, $b \neq 366$ and $b = 366$. When $a = 1$ we have that

$$Q_{\mathcal{I}\mathcal{I}} = \frac{1}{\sigma_u^2} \begin{bmatrix} 1 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \dots & 0 \\ 0 & -1 & 2 & -1 & \vdots \\ \vdots & & \ddots & \ddots & \end{bmatrix}$$

Then $Q_{\mathcal{I},-\mathcal{I}}$ has dimensions $M \times 366 - M$. In addition, $Q_{\mathcal{I},-\mathcal{I}}$ is zero everywhere except for the element at its M'th row and first column, which is element $q_{M,M+1} = -1$, which is at row M and column $M+1$ in the full precision matrix Q. Let $Q_{\mathcal{I}\mathcal{I}}^{-1}[, M]$ be column M of $Q_{\mathcal{I}\mathcal{I}}^{-1}$, we then have that

$$\boldsymbol{\mu}_{\mathcal{I}|-\mathcal{I}} = Q_{\mathcal{I}\mathcal{I}}^{-1}[, M]\tau_{(b+1)}$$

When neither $a = 1$ or $b = 366$, the precision matrix becomes

$$Q_{\mathcal{I}\mathcal{I}} = \frac{1}{\sigma_u^2} \begin{bmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \dots & 0 \\ 0 & -1 & 2 & -1 & \vdots \\ \vdots & & \ddots & \ddots & \end{bmatrix}$$

Here $Q_{\mathcal{I},-\mathcal{I}}$ will be zero everywhere except for at row 1, column $a-1$ where it is -1 and row b, column $a+1$ where it is -1. This in turn implies that

$$\boldsymbol{\mu}_{\mathcal{I}|-\mathcal{I}} = (Q_{\mathcal{I}\mathcal{I}}^{-1}[, 1], Q_{\mathcal{I}\mathcal{I}}^{-1}[, M]) \cdot (\tau_{a-1}, \tau_{b+1})^T$$

Lastly, when $b = 366$

$$Q_{\mathcal{I}\mathcal{I}} = \frac{1}{\sigma_u^2} \begin{bmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \dots & 0 \\ 0 & -1 & 2 & -1 & \vdots \\ \vdots & & \ddots & \ddots & -1 \\ 0 & \dots & 0 & -1 & 1 \end{bmatrix}$$

Here $Q_{\mathcal{I},-\mathcal{I}}$ is zero everywhere except for at row 1, column $a-1$ where it is -1. Hence

$$\boldsymbol{\mu}_{\mathcal{I}|-\mathcal{I}} = Q_{\mathcal{I}\mathcal{I}}^{-1}[, 1]\tau_{a-1}$$

Precomputiations

Before the actual MCMC blocking algorithm is run, we perform some precomputations to save time. First of all, we precompute $\boldsymbol{\mu}_{\mathcal{I}|-\mathcal{I}}$ for the three different cases shown above. Also, we compute the Cholesky decompositions of the three different $Q_{\mathcal{I}\mathcal{I}}^{-1}$ matrices.

We use this when generating the samples $\boldsymbol{\tau}'_{\mathcal{I}}$. First, we draw samples from a standard normal distribution. Then we utilize the fact that if $x \sim \mathcal{N}(0, I)$ then $y = \mu + Lx \sim \mathcal{N}(\mu, LL^T)$. For our case, μ will be the three different $\boldsymbol{\mu}_{\mathcal{I}|-\mathcal{I}}$ described above, while L is the lower triangular Cholesky decomposition of $Q_{\mathcal{I}\mathcal{I}}^{-1}$ for the three different cases, but we have to remember to multiply L by σ_u.

MCMC Blocking implementation

```r
#----------------------------------------------------------------------
get_Q <- function(n){
  #Function to create precision matrix that is not scaled
  library(Matrix)
  m <- diag(2,n)
  m[abs(row(m) - col(m)) == 1] <- -1
  m[1,1] = 1
  m[n,n] = 1
    return(Matrix(m, sparse = F))
}

#----------------------------------------------------------------------


MCMC_block_sampler <- function(M, tau_0, nsteps = 50000, alpha = 2, beta = 0.05, verbose=F){
  # MCMC block Sampling algorithm for Tokyo rainfall dataset
  # M       : Length of block intervals
  # tau_0   : 366-length vector of initial tau values
  # alpha   : Shape parameter for sigma_u^2
  # beta    : Scale parameter for sigma_u^2
  # nsteps  : Number of MCMC iterations, defaults to 50.000

  start = proc.time()[3] # To measure time

  # Intializations
  n.rain = rain$n.rain
  n.years = rain$n.years
  accepted = 0

  current_tau = tau_0
  tau = matrix(0, nrow = length(current_tau), ncol = nsteps)
  sigmas = rep(0, nsteps)
  tau_proposed = matrix(0, nrow = length(current_tau))

  # Progress bar initialization
  pb = txtProgressBar(min = 0, max = nsteps, initial = 2)

  # Calculations for the blocks
  leftovers = ifelse(366%%M==0, M, 366%%M) # Leftovers after all "full" blocks
  nr_blocks = ceiling(366/M) # Number of blocks

  # Initializing matrices
  #----------------------------------------------------------------------
```

```r
Q = get_Q(366) #The full precision matrix, not scaled

# First block matrices, a=1
Q_AA1 = Q[1:M,1:M]
inv_Q_AA1 = solve(Q_AA1)
mu_factor_1 = inv_Q_AA1[, M]

# Midsection matrices, a != 1,  b != 366
Q_AA2 = Q[2:(M+1), 2:(M+1)]
inv_Q_AA2 = solve(Q_AA2)
mu_factor_2 = cbind(inv_Q_AA2[,1], inv_Q_AA2[, M])

# Last block matrices, b=366
Q_AA3 = Q[(367-M):366, (367-M):366]
inv_Q_AA3 = solve(Q_AA3)
mu_factor_3 = inv_Q_AA3[,1]


# Cholesky decompositions of inverses of block matrices
#     Transposed to get lower triangular
chol_inv_Q_AA1 = t(chol(inv_Q_AA1))
chol_inv_Q_AA2 = t(chol(inv_Q_AA2))
chol_inv_Q_AA3 = t(chol(inv_Q_AA3))


sigma_u = sqrt(1/rgamma(1, shape = alpha, rate = beta))

# #For-loop with every iteration
for (k in 1:nsteps) {
  if(verbose){setTxtProgressBar(pb,k)} # Update progress bar

  #MH-step
  #---------------------------------------------------------------------------

  #Pre-draw normal and uniform samples
  normal_samples = matrix(rnorm(nr_blocks*M), nrow=nr_blocks)
  uniform_samples = runif(nr_blocks)

  ##### First block, a = 1 #####
  a = 1; b = M
  tau_proposed = mu_factor_1*current_tau[b+1] +
    sigma_u*chol_inv_Q_AA1%*%normal_samples[1,]

  # Calculate the acceptance probability for the proposed samples
  # We don't have to use min(1, .) since if larger than 1, we accept anyways
  acceptence_probability = exp(sum(n.rain[a:b]*(tau_proposed-tau[a:b])+
                                   n.years[a:b]*log((1+exp(tau[a:b]))/(
                                     1+exp(tau_proposed)))))
  # Check if we accept the proposed samples. If we do, update current tau
  if(uniform_samples[1] < acceptence_probability){
    current_tau[a:b] = tau_proposed
    accepted = accepted + M/366
  }
  ##### Midsection, a != 1, b != 366 #####
  for (j in 2:(nr_blocks-1)) {
    a = 1+(j-1)*M; b = j*M
    tau_proposed = as.array(mu_factor_2%*%current_tau[c(a-1,b+1)])+
      as.array(sigma_u*chol_inv_Q_AA2%*%normal_samples[j,])
    #Calculate the acceptance probability for the proposed samples
    acceptence_probability = exp(sum(n.rain[a:b]*(tau_proposed-current_tau[a:b])+n.years[a:b]*log((1+
    #Check if we accept the proposed samples, if we do, update current tau
    if(uniform_samples[j] < acceptence_probability){
      current_tau[a:b] = tau_proposed
      accepted = accepted + M/366
    }
  }

  ##### Last block, b = 366 #####
  a = 366-M+1; b = 366; a_leftover = 366 - leftovers + 1
  lower_index = M - leftovers + 1 # Need this to make it work
  tau_proposed = mu_factor_3*current_tau[a-1]+
    sigma_u*chol_inv_Q_AA3%*%normal_samples[nr_blocks,]

  acceptence_probability = exp(sum(n.rain[a_leftover:b]*(
    tau_proposed[lower_index:M]-current_tau[a_leftover:b])+
      n.years[a_leftover:b]*log((1+exp(current_tau[a_leftover:b]))/(
        1+exp(tau_proposed[lower_index:M])))))

  if(uniform_samples[nr_blocks] < acceptence_probability){
    current_tau[a_leftover:b] = tau_proposed[lower_index:M]
    accepted = accepted + leftovers/366
  }


  tau[, k] = current_tau

  # Gibbs step
  #----------------------------------------------------------------------------
  sigmas[k] = 1/rgamma(10, shape = (alpha + (366-1)/2),
                          rate = beta + 0.5 * sum(diff(current_tau)^2))

  sigma_u = sqrt(sigmas[k])

  #----------------------------------------------------------------------------


}

close(pb)
# Return tau samples, variance samples, runtime in seconds and acceptance rate
res = list(
  tau        = tau,
  runtime    = proc.time()[3]-start,
  sigma_u    = sigmas,
  acceptance = accepted/(nsteps)
```

```
    )

  return(res)
}
```

Figure **??** shows a plot of the acceptance rates as a function of M for a total of 2000 iterations, and similarly, the plot below shows the runtime of the algorithm as a function of M. Both plots show a trade-off between acceptance rate and runtime. We would like to choose an M with as high acceptance rate as possible while simultaneously giving low computational time. Increasing M yields lower accept rates and lower runtime. From Figure **??**, we choose $M = 10$, as it has a relatively low runtime while also preserving a high acceptance rate.

It is expected that the runtime will be lower for higher values of M, as there are fewer computations. However, the acceptance rate is also expected to diminish for increasing M, because the algorithm accepts or rejects entire blocks.

```
plot_acceptance_and_runtime <- function(verbose=F){
  M = c(1, 3, 5, 10, 15, 20, 50, 100)
  acceptance_list = c()
  runtime_list = c()
  steps = 2000

  ind_count = 1
  for(m in M){
    if(verbose){print(paste("Fitting for M", m, "..."))}
    fit = MCMC_block_sampler(m, rnorm(366, 0, sqrt(0.07)), nsteps = steps)
    acceptance_list[ind_count] = fit$acceptance
    runtime_list[ind_count] = fit$runtime
    ind_count = ind_count + 1
  }
  return(list(M=M, acceptance=acceptance_list, runtime=runtime_list))
}
M_plotting_list = plot_acceptance_and_runtime()
```

We now apply the algorithm with 50 000 iterations and $M = 10$.

```
result_block = MCMC_block_sampler(10,rnorm(366, 0, sqrt(0.007)),
                                   nsteps = 50000)
```

```
## ================================================================================
```

This run had a runtime of 293.9 seconds with acceptance rate 0.461. Although we expected the blocking algorithm to be faster than the single-site algorithm, it was not. This is mainly due to code optimizations such as the `as.array` convertions, which had to be called due to unexpected behavior. With additional time, one could get rid of such time-costly operations and achieve a faster blocking algorithm.

We produce the same plots as for the single-site MCMC, namely trace plots, histograms and autocorrelation plots. In addition, we compare the predictions from the single-site algorithm to our blocking algorithm.

The trace plots in Figures 10 and 11 follow the same trends as the single-site results. Thus, we choose the same burn-in period of 500.

The histograms, as shown in Figure 12, also follow the same pattern as the single-site case, with corresponding means. This is also to be expected, since both algorithms are MCMC samplers.

The plots that differ significantly from the single-site case is however the autocorrelation plots in Figure 13. Here, we see a rapid decay of the autocorrelation for τ_1 and τ_{366}. τ_{206} does not seem to have any significant

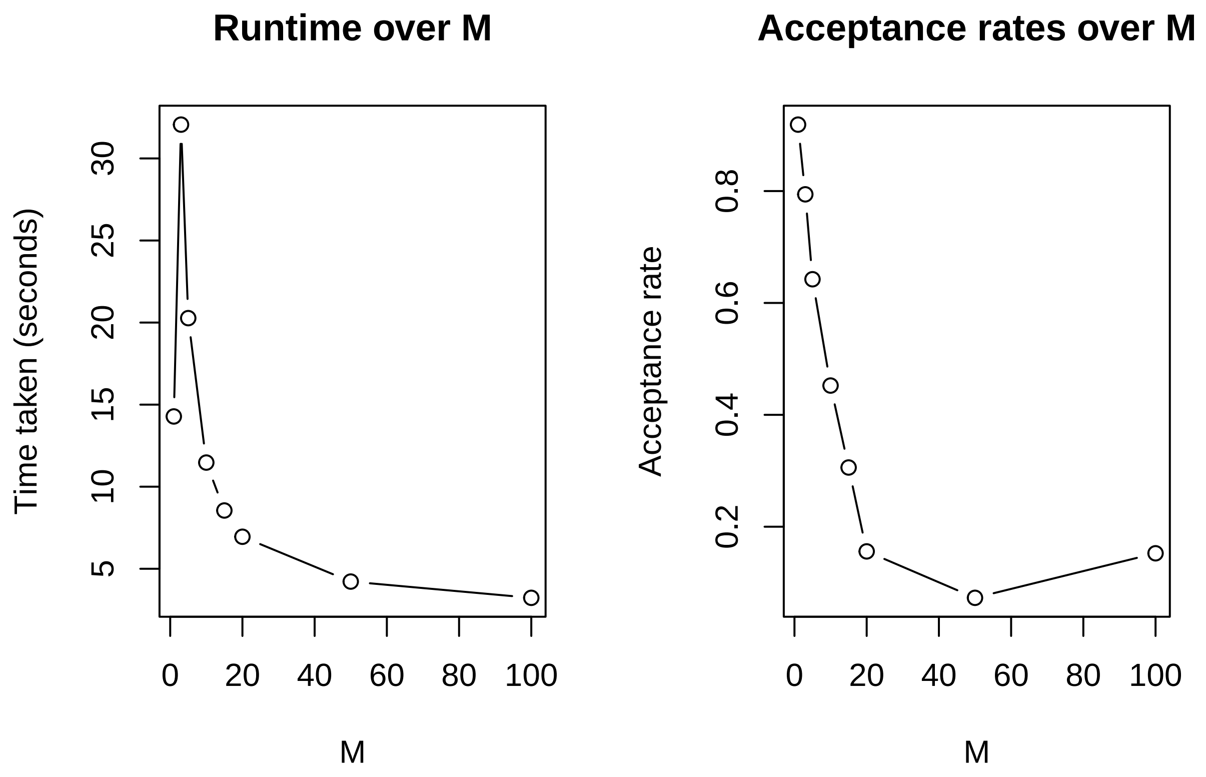


Figure 9: Plot of runtime over different M.

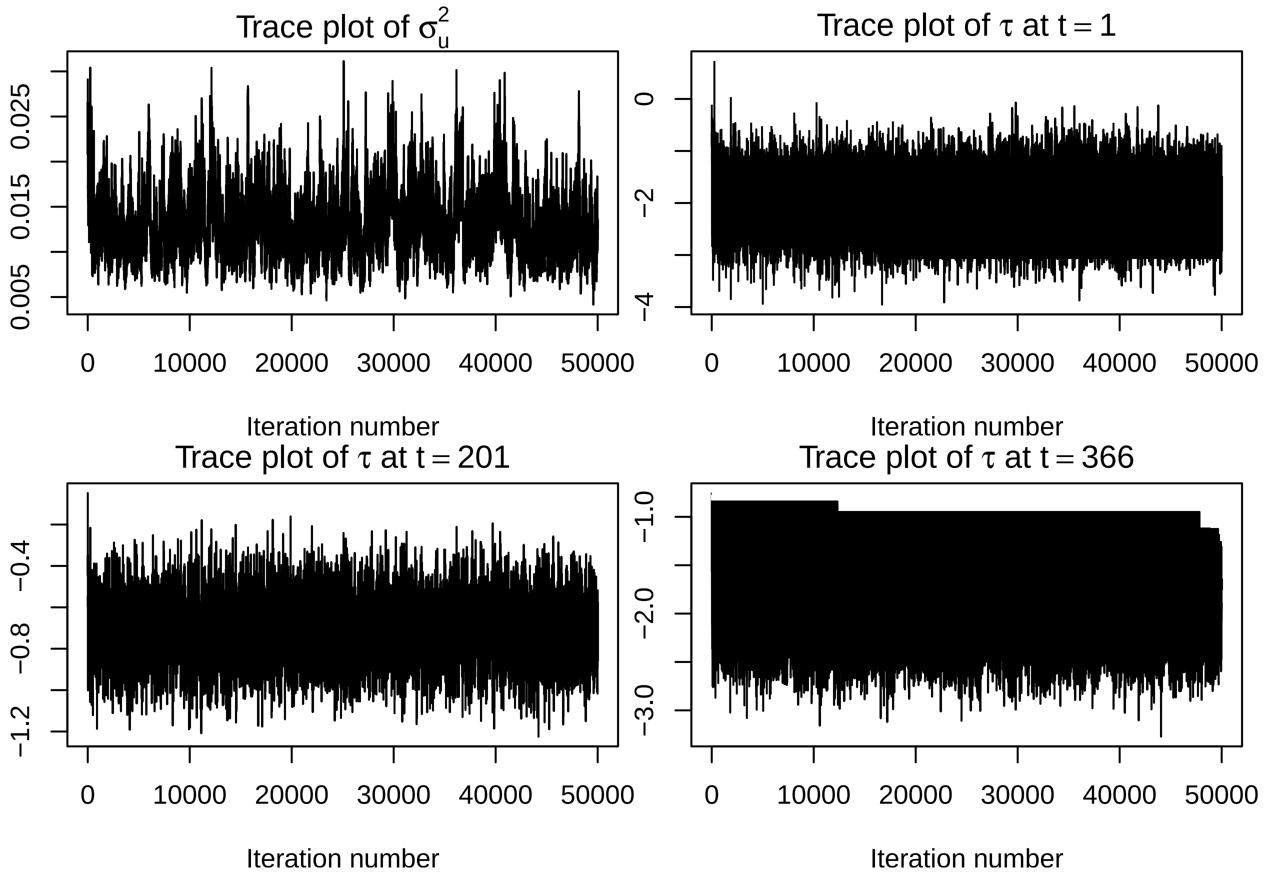


Figure 10: Trace plots of σ_u^2 and τ_t for $t = 1, 201, 366$ for all iterations of blocking MCMC.

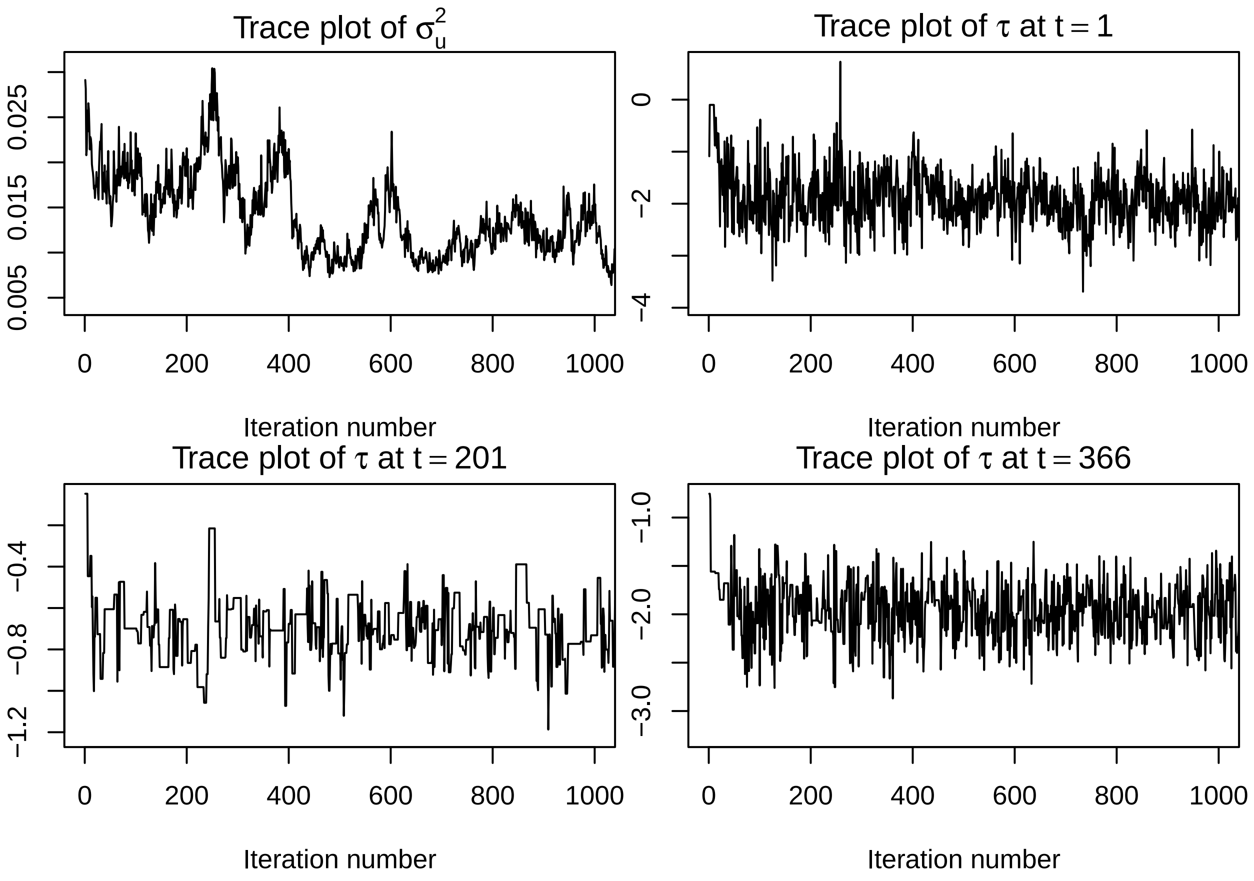


Figure 11: Trace plots of the same parameters for the 1000 first values with blocking MCMC.

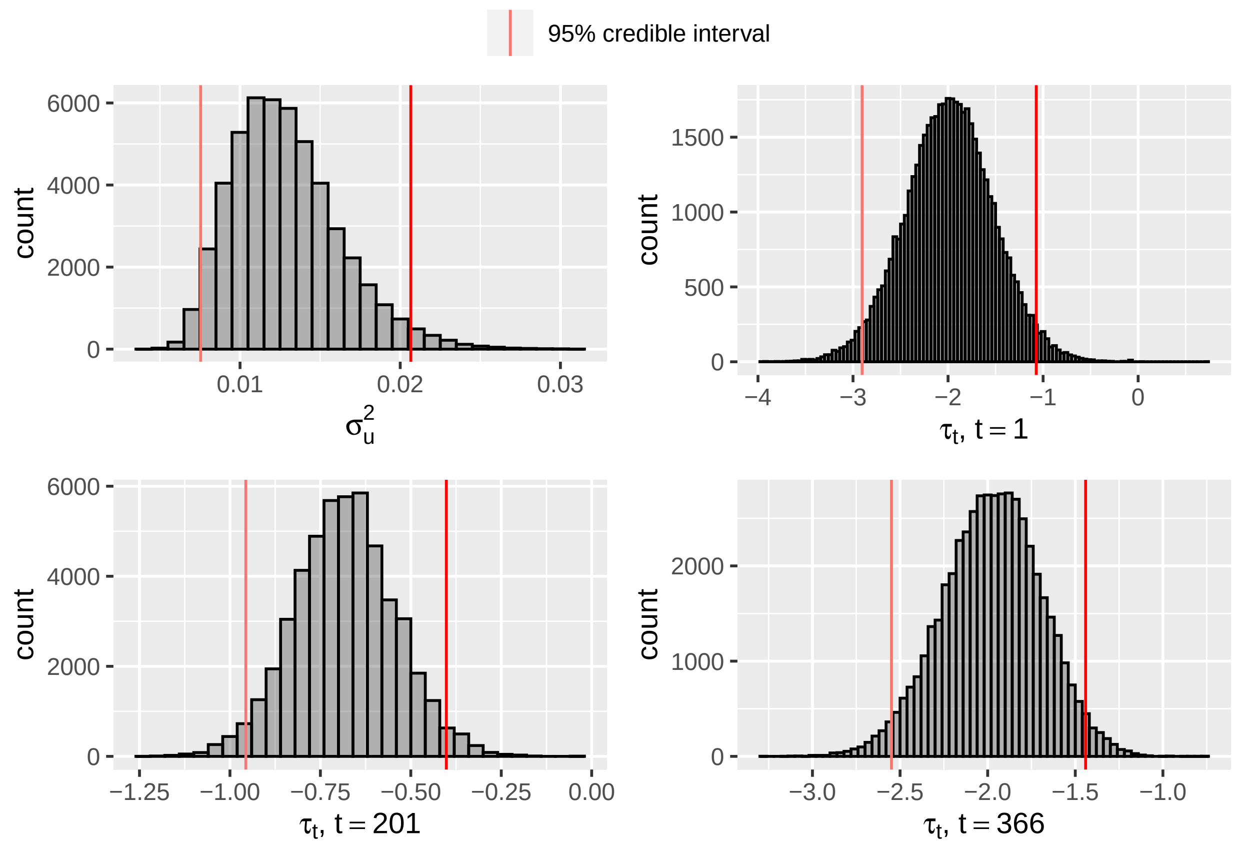


Figure 12: Histograms of σ_u^2 and τ_t for $t = 1, 201, 366$ with their corresponding 95% credible intervals - for blocking MCMC.

changes, and σ_u^2 seem to increase a bit compared to the autocorrelation of the single-site case. This can however be attributed to the monte carlo randomness.

The fact that the values are less correlated in the blocking algorithm, may be since the correlation effect between τ_i and τ_j is reduced when they are jointly updated in the same block.

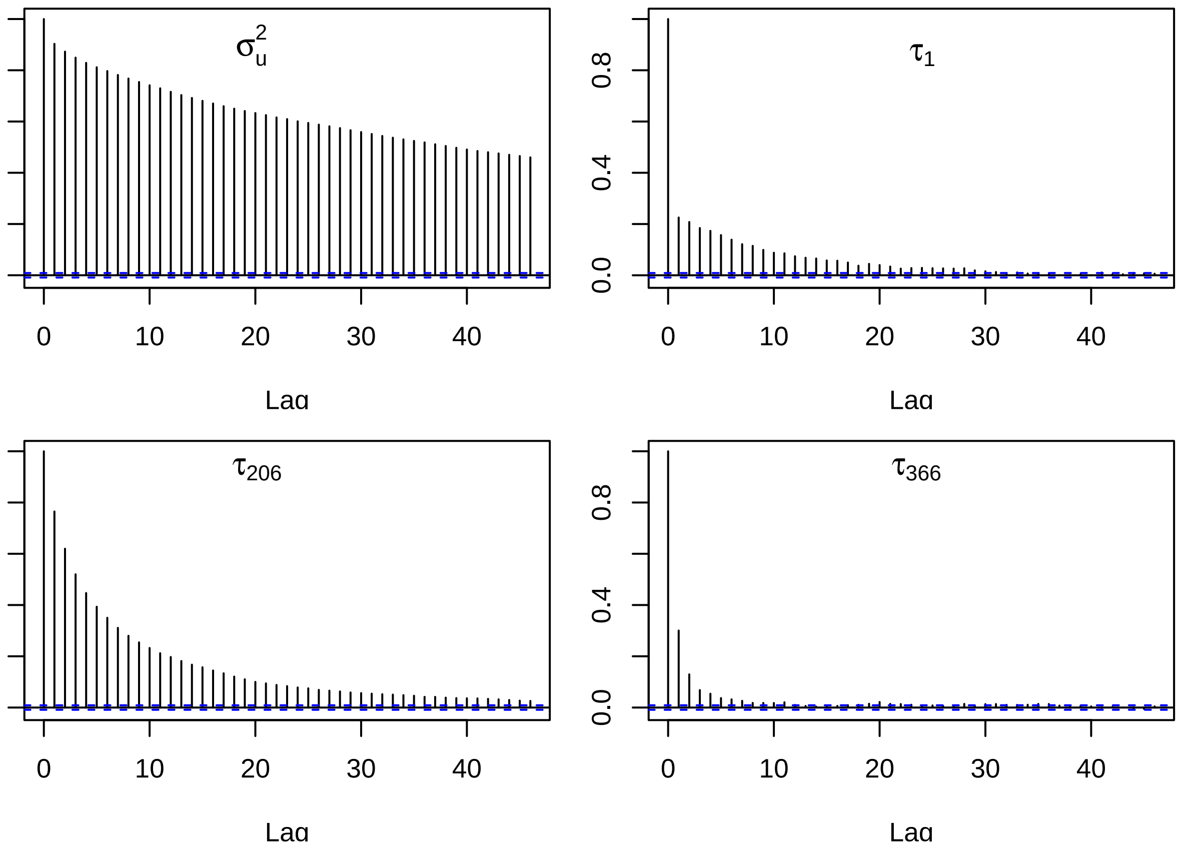


Figure 13: Autocorrelation plot of the four parameters as a function of lag, for blocking MCMC

Lastly, we plot a comparison between the single-site results and the blocking results in Figure 14. They seem to coincide well as expected.

Analyzing rainfall dataset using INLA

INLA is an abbreviation for integrated nested Laplace approximation and a widely used package in R for Bayesian inference.

We will now attempt to get the same predictions using the INLA-library in R. Note that we also place a prior on σ_u^2 in the INLA-model, since this is what we have done in the MCMC sampling algorithm. The prior is written in the hyper-argument for the function `f()` in the `inla` function call, and with keyword `prec`, we use the *log precision*. Thus, instead of using inverse gamma, we will be using log-gamma distribution on the prior - with the same parameters for α and β as before, i.e. 2 and 0.05 respectively. By including `-1` in the function call, we remove the intercept and thus we don't place any priors on the intercept. We also use random walk of order 1 on the INLA model, since this is done in the MCMC algorithm. The model is computed below:

```
fit_and_plot <- function(control, intercept=FALSE, constr=FALSE,
                         compare_MCMC=FALSE){
  # Main method for fitting and plotting INLA models.
  #
```

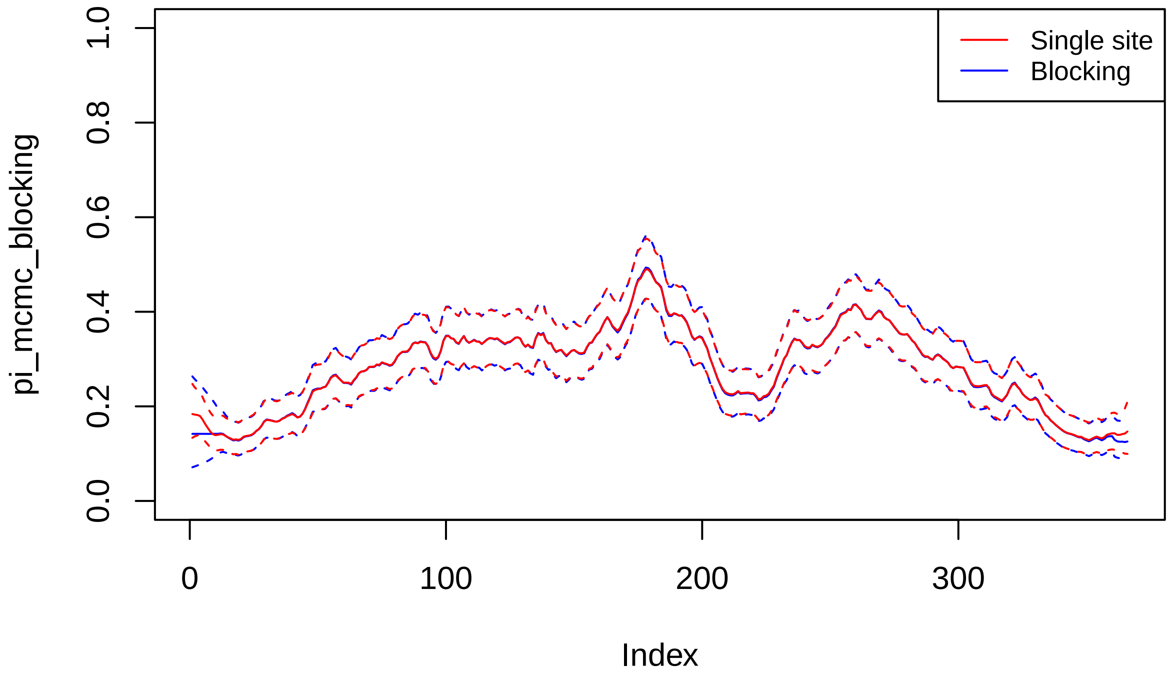
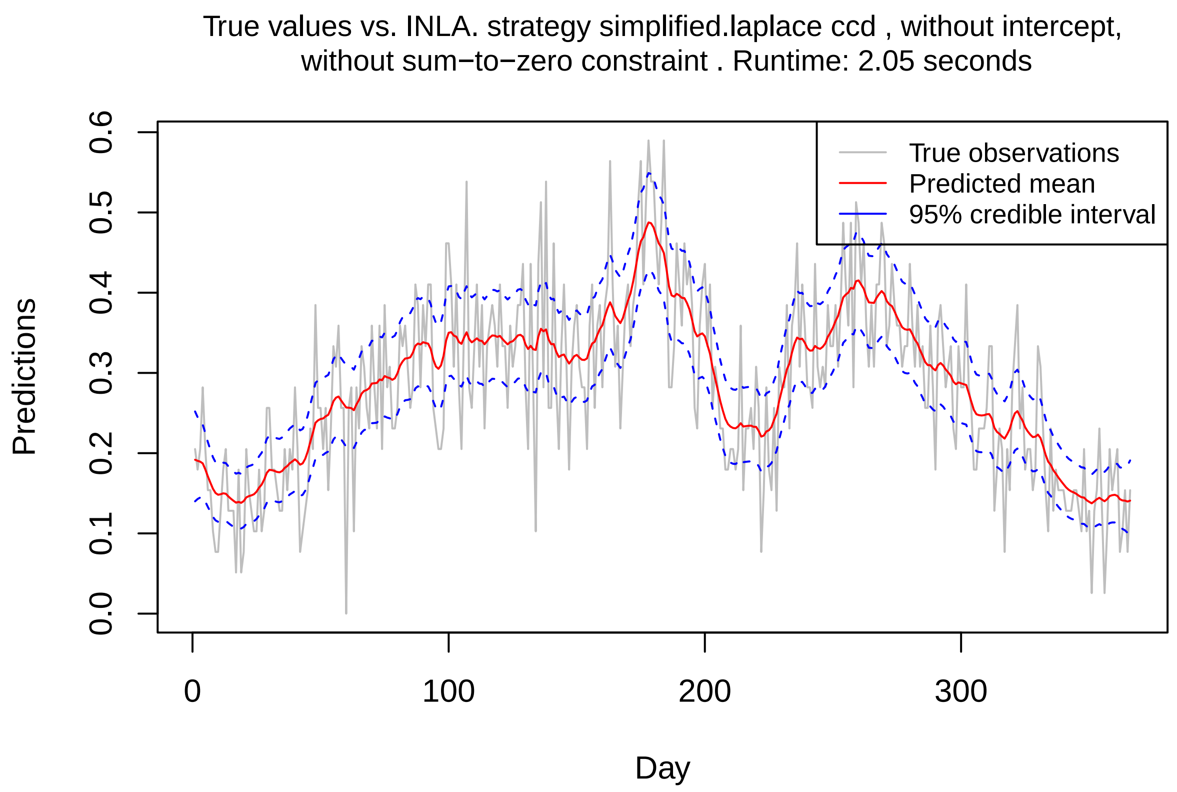


Figure 14: Comparison between single-site and blocking MCMC. Here, M=10.

```r
  # control      : list containing parameters for inla.control
  # intercept    : bool determining if intercept should be included or not
  # constr       : whether or not sum-to-zero constraint should be imposed
  # compare_MCMC : bool determining if plot should focus on INLA result
  #                (with credible interval), or by comparing the MCMC
  #                results withthe an INLA model.

  # Prepare plot title based on parameters/settings
  intercept_str = if(intercept) "with" else "without"
  constr_str = if(constr) "with" else "without"
  plot_text <- if(!compare_MCMC) paste("True values vs. INLA. strategy",
                    control$strategy, control$int.strategy, ",",
                    intercept_str, "intercept,\n",
                    constr_str, "sum-to-zero constraint") else
                      "INLA results vs. MCMC. INLA"
  # Fit INLA model
  time_start = proc.time()[3]
  mod <- inla(n.rain ~ -1*(1-intercept) +
                f(day, model="rw1", constr=constr, hyper=list(prec=list(
                  prior="loggamma",
                  param=c(2,0.05)
                ))),
              data=rain, Ntrials=n.years, control.compute=list(config=TRUE),
              family="binomial", verbose=FALSE, control.inla= control)
  runtime = proc.time()[3] - time_start

  # Plotting
  plot(rain$n.rain/rain$n.years, type="l", col="grey", xlab="Day",
       ylab="Predictions", main=paste(plot_text, ". Runtime:",
         round(runtime,3), "seconds"), cex.main=0.9, font.main = 1
       )

  lines(mod$summary.fitted.values$mean, type="l", col="red")
  legend_txt = c("True observations", "Predicted mean", "95% credible interval")
  if(compare_MCMC){
    pi_mcmc = expit(rowMeans(remove_burnin_tau(result$tau)))
    lines(pi_mcmc, type="l", col="blue")
    legend_txt[3] = "MCMC predictions"
  }
  else{
    lines(mod$summary.fitted.values$`0.025quant`, lty="dashed", col="blue")
    lines(mod$summary.fitted.values$`0.975quant`, lty="dashed", col="blue")
  }
  legend("topright", legend=legend_txt, col=c("grey", "red", "blue"),
         lty=1, cex=0.8)
  return(runtime)
}
```

We can compare the predicted values, stored in `mod$summary.fitted.values`, to their true observations for each day. We plot the predicted mean with a 95% credible interval in Figure .

\begin{figure}



{

}

\caption{Predicted values from INLA model compared with a 95% credible interval plotted alongside the true observations. We observe that the predicted mean closely follows the true observations, and the true observations are rarely outside the 95% credible interval.} \end{figure} Note also that the runtime is 2.05, which is significantly faster than any MCMC algorithm. The results also look similar to that of the MCMC predictions. We expect them to look practically identical since both use the same random walk model and the same prior on σ_u^2. This can be confirmed by comparing the results as done in Figure 15, where we use a single-site MCMC result to compare. We only use single-site since the blocking algorithm produces virtually the same plots. Although the predictions are very similar for INLA and MCMC, they are not exactly the same. This could be the result of the randomness, or Monte Carlo error, introduced in the MCMC algorithm.

```
## elapsed
##     1.7
```

INLA for different control settings

Another part of the INLA model used, is the `control.inla` parameter, where we specify the optimization strategy and integration strategy. For the optimization strategy, we use `simplified.laplace`, which is the default parameter and works well for most cases. It's based on a series expansion of the standard Laplace approximation around $\mu_i(\theta)$.

For the integration strategy, we use`ccd`, which is an abbreviation for a *central composite design.* This improves performance compared to integrating over an entire equidistant grid, since CDD uses fewer, well-placed points. CCD is however less accurate than a grid approach, so it is only beneficial to use when the dimensions are too large to fit a model using a grid within reasonable runtime. However, since our dimensions are 1, we could get use of using the grid approach instead.

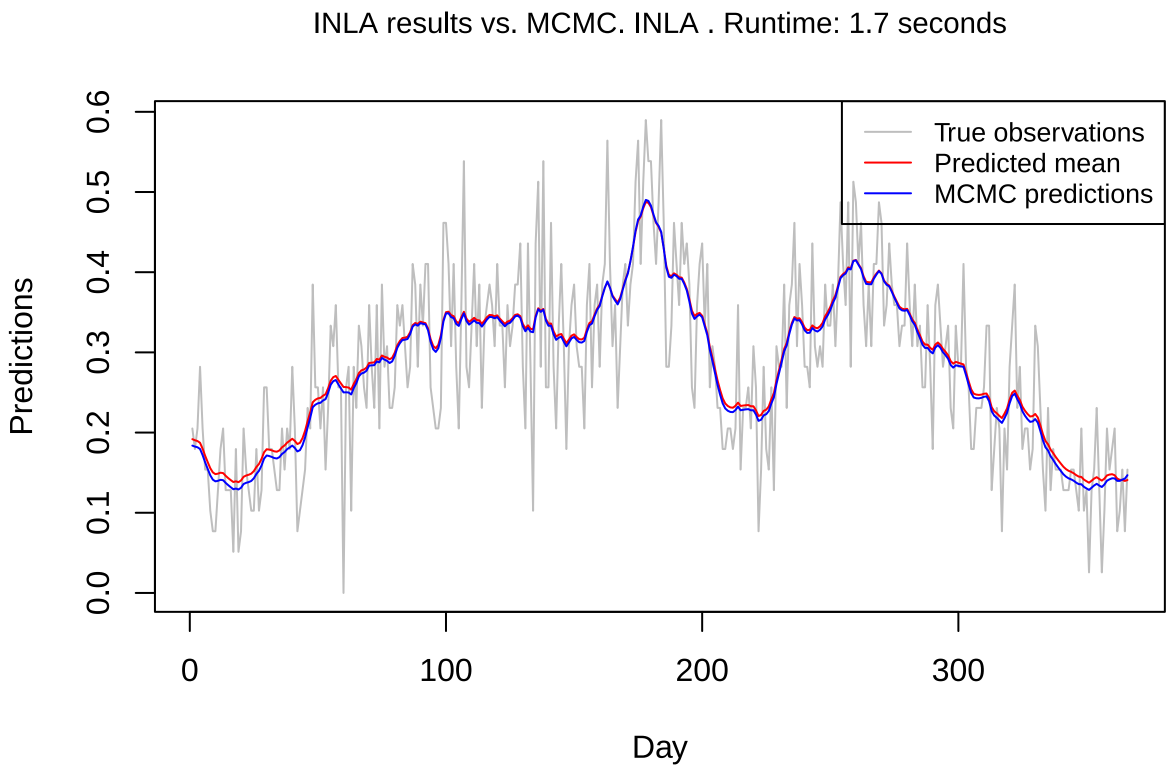


Figure 15: Comparison of the results from INLA and the two MCMC methods

The results from the grid approach compared to the CCD approach is shown in Figure 16 for completeness, but is identical to `cdd`. In addition, we also plot a comparison between `laplace` and `simplified.laplace` in Figure 16, which also seems to be identical. This indicates that the results are robust to the choice of control settings.

```r
compare_inla_models <- function(control1, control2, main){
  # Helper to compare two INLA models
  mod_1 <- inla(n.rain ~ -1 +
                f(day, model="rw1", constr=FALSE, hyper=list(prec=list(
                  prior="loggamma",
                  param=c(2,0.05)))),
              data=rain, Ntrials=n.years, control.compute=list(config = TRUE),
              family="binomial", verbose=FALSE, control.inla=control1
              )
  mod_2 <- inla(n.rain ~ -1 +
                f(day, model="rw1", constr=FALSE, hyper=list(prec=list(
                  prior="loggamma",
                  param=c(2,0.05)))),
              data=rain, Ntrials=n.years, control.compute=list(config = TRUE),
              family="binomial", verbose=FALSE, control.inla=control2
              )
  plot(mod_1$summary.fitted.values$mean, type="l", col="red", main=main,
       xlab="Days", ylab="Predicted mean")
  lines(mod_2$summary.fitted.values$mean, type="l", lty="dashed", col="blue")
  legend("topright", col=c("red", "blue"), lty=1, legend=c(
    paste(control1$strategy, control1$int.strategy),
    paste(control2$strategy, control2$int.strategy)))
}
par(mfrow=c(1,2))
compare_inla_models(list(strategy="simplified.laplace", int.strategy="grid"),
                    list(strategy="simplified.laplace", int.strategy="ccd"),
                    main="CCD vs Grid method"
                    )
compare_inla_models(list(strategy="laplace", int.strategy="grid"),
                    list(strategy="simplified.laplace", int.strategy="grid"),
                    main="Simplified Laplace vs. Laplace"
                    )
```

Intercept and constraints on INLA model

The models shown so far did not include any intercept, nor did they impose any sum-to-zero constraint. We will now explore how this may change our predictions.

Instead of modeling $\pi(\tau_t)$, by introducing an intercept term, we introduce the term $\pi(\eta_t)$ in which $\eta_t := \tau_t + \beta_0$, where β_0 is the intercept. In addition, we impose a sum-to-zero constraint, meaning that $\sum_t \tau_t = 0$.

We can start by plotting the mean of the fitted values for all variations, i.e. without intercept nor constraint, with intercept and constraint, without intercept with constraint and without intercept, with constraint. The four plots are shown in the resulting Figure 17. In summary, having an intercept term but also imposing a sum-to-zero constraint seem to be exactly the same as having neither an intercept, nor the constraint. If we don't impose the constraint on a model with an intercept, we will see slight deviations from the other two models. Lastly, if we impose the constraint on a model *without* any intercept, we get very large deviations, but the fitted values seem to be scaled along the y-axis. Looking at the y-axis, it makes sense that the last model is shifted upwards, in order to attain the sum-to-zero constraint.

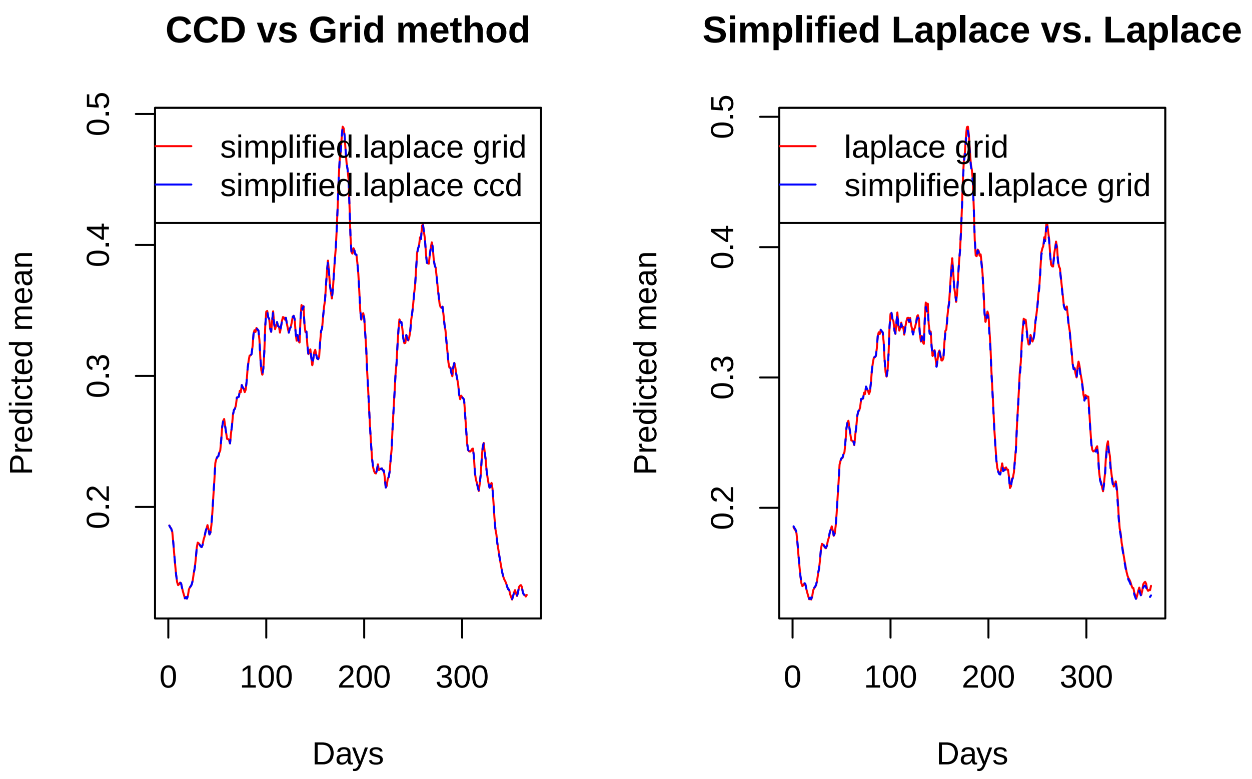


Figure 16: Plot comparing predictions using simplified laplace with CCD strategy as red, and grid strategy as blue to the left Similarly, the plot to the right compares using Laplace and simplified laplace strategy. Both plots seem to coincide.

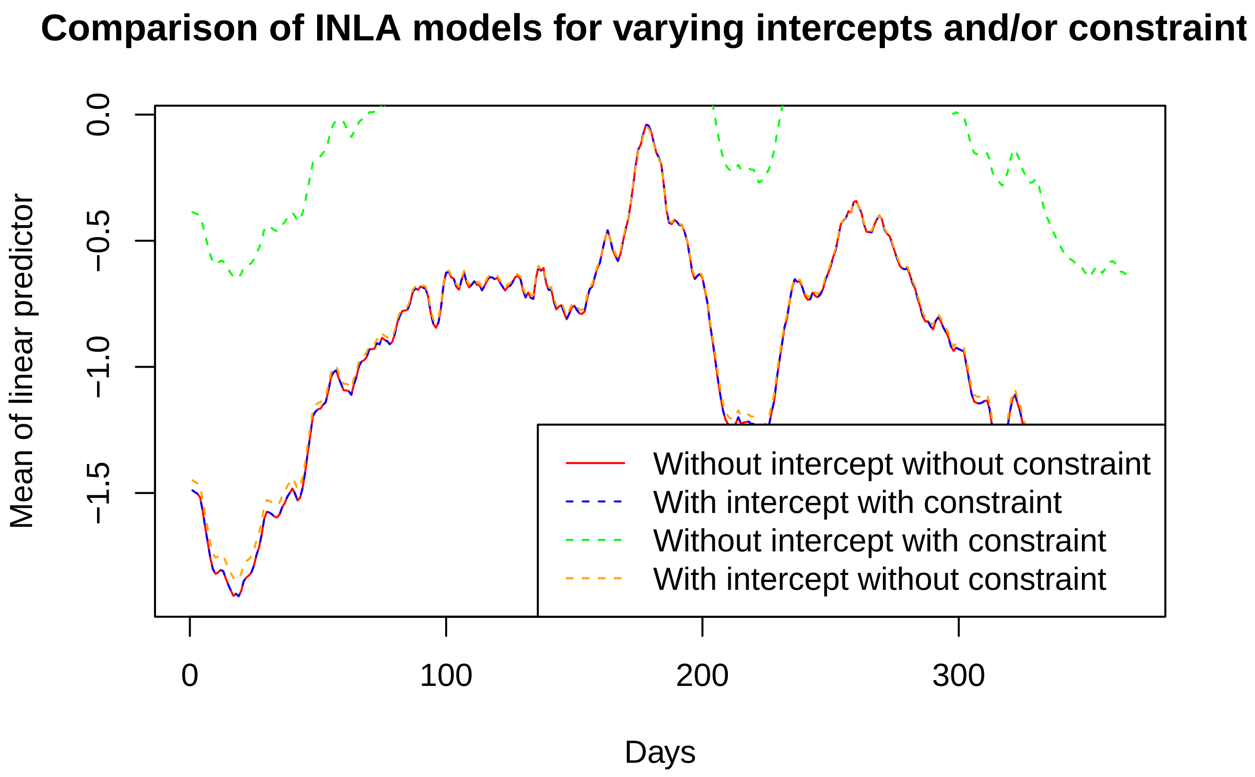


Figure 17: Comparison between models varying on intercept and sum-to-zero constraints.