Compulsory exercise 1: Group 5

TMA4300 Computational Statistics V2022

Markus Johnsen Aase, Nora Aasen

18 mars, 2022

Contents

1 Problem 1

1.1 a)

We consider a dataset that looks at rainfall in Tokyo over a period of 39 years, from 1951 to 1989. The response is a Bernoulli variable for day each year, indicating 1 if the amount of rainfall exceeded 1mm for that date, and 0 if it did not. Hence, the conditional distribution of our response is

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

where t is the date of each observation and n_t is the number of years rainfall have been registered for that date. In our case,

$$n_t = \begin{cases} 10 & t = 60 \\ 39 & t \neq 60 \end{cases}$$

since the leap day, 29th of February, is observed only each fourth year. We assume that the responses are conditionally independent.

Furthermore, $\pi(\tau_t)$ is an expression for the probability of rainfall exceeding 1mm on the respective day, t, of the year, which is given by

$$\pi(\tau_t) = \frac{\exp(\tau_t)}{1+\exp(\tau_t)} = \frac{1}{1+\exp(-\tau_t)}.$$

We recognize τ_t as the logit probability, and $\pi(\cdot)$ as the inverse logit function.

```
# load the file into RStudio
load(file = 'rain.rda')

# look at the head of the data file
head(rain)
```

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

```
# look at a summary for the response
summary(rain$n.rain)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00    8.00   11.00   10.98   14.00   23.00
```

```
# look at the leap day
rain[60,]
```

```
##     day n.years n.rain
## 366  60      10      0
```

We start by inspecting our dataset. As we can see, the variable we are most interested in is n.rain, so we look at the summary for those data. Additionally, we take a look at day 60, i.e. the 29th of February, as there are less observations for this date. We note that it has never been registered rain exceeding 1mm on this date.

```
# plot days of rain against day of the year
plot1 <- ggplot(data=rain,aes(x=day,y=n.rain))+geom_line()+labs(x = 'Day of year', y = 'Number of days'

plot2 <- ggplot(data=rain,aes(x=day,y=n.rain))+geom_smooth(span=0.5)+labs(x = 'Day of year', y = 'Numbe

plot_grid(plot1, plot2, labels = NULL)
```

As we can see from figure 1, there is a clear trend towards more rainy days during the late spring and early summer. At the end and start of each year, there is fewer days with more than 1mm rain. This corresponds well with the rain season in Tokyo, which is from early June to mid July. Note that 1st of June is day 152 of the year, and 15th of July is the 196th day of the year.

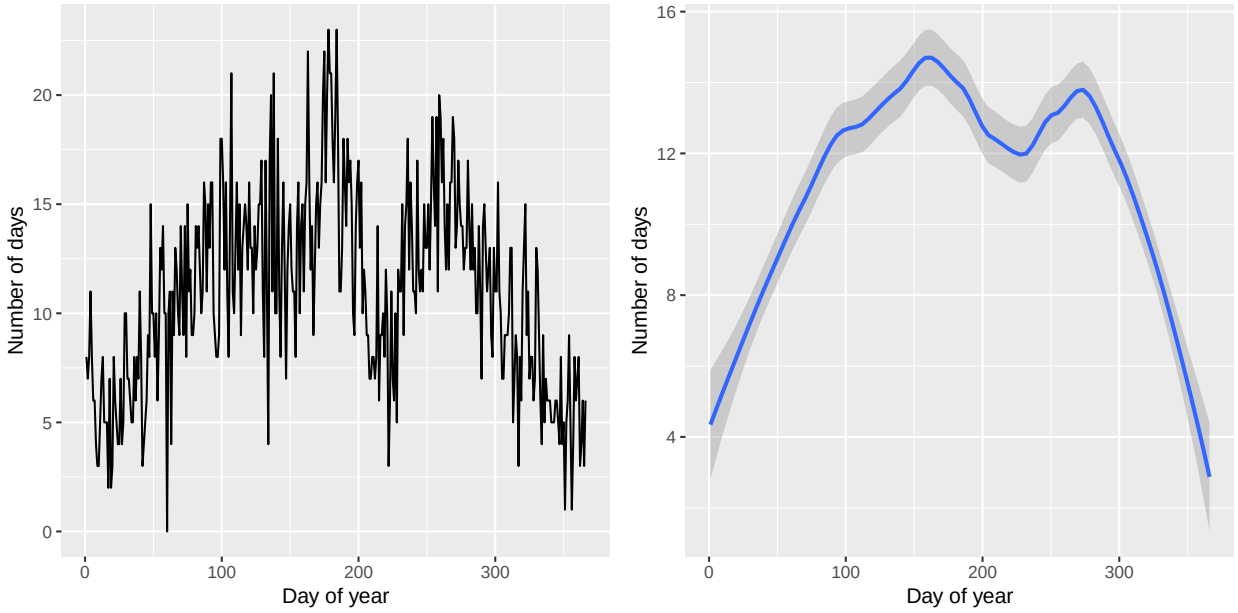


Figure 1: Two plots showing the number of days with more than 1mm rain for the last 39 years, against each day of the year. The right figure show a smoothed trend and a 95% credible interval.

1.2 b)

Next, we find an expression for the likelihood $L(y_t|\pi(\tau_t))$. Since we know that $y|\tau_t \sim Bin(n_t, \pi(\tau))$, and that these observations are conditionally independent, we get

$$L(y_t|\pi(\tau_t)) = \prod_{t=1}^{366} \binom{n_t}{y_t} (\pi(\tau_t))^{y_t}(1-\pi(\tau_t))^{1-y_t} = (1-\pi(\tau_{60})) \prod_{t=1 t\neq 60}^{366} \binom{39}{y_t} (\pi(\tau_t))^{y_t}(1-\pi(\tau_t))^{1-y_t}.$$

1.3 c)

We now consider the trend to be a random walk on logit scale, i.e.

$$\tau_t \sim \tau_{t-1} + u_t, \quad u_t \overset{iid}{\sim} \mathcal{N}(0, \sigma_u^2),$$

where σ_u^2 has an inverse gamma prior, i.e $\frac{1}{\sigma_u^2} \sim Gamma(\alpha, \beta)$.

From this we wish to find an expression for $p(\sigma_u^2|\boldsymbol{y}, \boldsymbol{\tau})$. Using Bayes' theorem, we find that the conditional distribution is given by

$$p(\sigma_u^2|\boldsymbol{y}, \boldsymbol{\tau}) = \frac{p(\boldsymbol{y}, \boldsymbol{\tau}|\sigma_u^2)p(\sigma_u^2)}{p(\boldsymbol{y}, \boldsymbol{\tau})}, \qquad p(\boldsymbol{y}, \boldsymbol{\tau}|\sigma_u^2) = p(\boldsymbol{y}|\boldsymbol{\tau}, \sigma_u^2)p(\boldsymbol{\tau}|\sigma_u^2))$$

In this expression, $p(\boldsymbol{y}, \boldsymbol{\tau})$ is merely a constant with respect to σ_u^2.

Using this, we can calculate the proportional density as

$$p(\sigma_u^2|\boldsymbol{y},\boldsymbol{\tau}) \propto \prod_{i=1}^{366} \binom{n_t}{y_t} \pi(\tau_t)^{y_t}(1-\pi(\tau_t))^{1-y_t} \tag{1}$$

$$\times \prod_{t=2}^{366} \frac{1}{\sigma_u} \exp\left\{-\frac{1}{2\sigma_u^2}(\tau_t - \tau_{t-1})^2\right\} \tag{2}$$

$$\times \frac{\beta^\alpha}{\Gamma(\alpha)} \left(\frac{1}{\sigma_u^2}\right)^{\alpha+1} \exp\left\{-\frac{\beta}{\sigma_u^2}\right\} \tag{3}$$

$$\propto \left(\frac{1}{\sigma_u}\right)^{365} \left(\frac{1}{\sigma_u^2}\right)^{\alpha+1} \exp\left\{-\frac{1}{2\sigma_u^2}\sum_{t=2}^{366}(\tau_t - \tau_{t-1})^2 - \frac{\beta}{\sigma_u^2}\right\} \tag{4}$$

$$= \left(\frac{1}{\sigma_u^2}\right)^{(\alpha+\frac{365}{2})+1} \exp\left\{-\frac{1}{\sigma_u^2}\left(\frac{1}{2}\sum_{t=2}^{366}(\tau_t - \tau_{t-1})^2 + \beta\right)\right\} \tag{5}$$

$$= \left(\frac{1}{\sigma_u^2}\right)^{(\alpha+\frac{365}{2})+1} \exp\left\{-\frac{1}{\sigma_u^2}\left(\frac{1}{2}\tau^T Q\tau + \beta\right)\right\} \tag{6}$$

We recognize this distribution as an inverse gamma distribution, and hence

$$\sigma_u^2|\boldsymbol{y},\boldsymbol{\tau} \sim Gamma^{-1}\left(\alpha + \frac{365}{2}, \frac{1}{2}\tau^T Q\tau + \beta\right).$$

1.4 d)

Here we consider the conditional prior proposal distribution

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

for our Metropolis-Hastings step, and we wish to calculate the resulting acceptance probability.

First, we split the vector of our current values $\boldsymbol{\tau} = [\boldsymbol{\tau}_{\mathcal{I}}, \boldsymbol{\tau}_{-\mathcal{I}}]$ and our proposed values $\boldsymbol{\tau}^* = [\boldsymbol{\tau}_{\mathcal{I}}^*, \boldsymbol{\tau}_{-\mathcal{I}}^*]$ into two sub-vectors. Then,

$$p(\boldsymbol{\tau}|\boldsymbol{y}, \sigma_u^2) = p([\boldsymbol{\tau}_{\mathcal{I}}, \boldsymbol{\tau}_{-\mathcal{I}}]|\boldsymbol{y}, \sigma_u^2) \tag{7}$$

$$\propto p(\boldsymbol{y}|\boldsymbol{\tau}, \sigma_u^2) \times p(\boldsymbol{\tau}|\sigma_u^2) \tag{8}$$

$$= p(\boldsymbol{y}|\boldsymbol{\tau}, \sigma_u^2) \times p(\boldsymbol{\tau}_{\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}}, \sigma_u^2) \times p(\boldsymbol{\tau}_{-\mathcal{I}}|\sigma_u^2) \tag{9}$$

The acceptance probability is given as

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

which in our case is

$$\alpha(\boldsymbol{\tau}^*_{\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}}, \sigma_u^2, \boldsymbol{y}) = \min\left(1, \frac{\pi(\boldsymbol{\tau}^*|\boldsymbol{y}, \sigma_u^2)}{\pi(\boldsymbol{\tau}|\boldsymbol{y}, \sigma_u^2))} \times \frac{Q(\boldsymbol{\tau}_{\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}}, \sigma_u^2, \boldsymbol{y})}{Q(\boldsymbol{\tau}^*_{\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}}, \sigma_u^2, \boldsymbol{y})}\right) \tag{10}$$

$$= \min\left(1, \frac{\pi(\boldsymbol{y}|\boldsymbol{\tau}^*, \sigma_u^2) \times \pi(\boldsymbol{\tau}^*_{\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}}, \sigma_u^2) \times \pi(\boldsymbol{\tau}_{-\mathcal{I}}|\sigma_u^2) \times \pi(\boldsymbol{\tau}_{\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}}, \sigma_u^2)}{\pi(\boldsymbol{y}|\boldsymbol{\tau}, \sigma_u^2) \times \pi(\boldsymbol{\tau}_{\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}}, \sigma_u^2) \times \pi(\boldsymbol{\tau}_{-\mathcal{I}}|\sigma_u^2) \times \pi(\boldsymbol{\tau}^*_{\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}}, \sigma_u^2)}\right) \tag{11}$$

$$= \min\left(1, \frac{\pi(\boldsymbol{y}|\boldsymbol{\tau}^*, \sigma_u^2)}{\pi(\boldsymbol{y}|\boldsymbol{\tau}, \sigma_u^2)}\right) \tag{12}$$

$$= \min\left(1, \frac{\pi(\boldsymbol{y}|\boldsymbol{\tau}^*)}{\pi(\boldsymbol{y}|\boldsymbol{\tau})}\right) \tag{13}$$

Hence, we get the acceptance probability that we expected, which is a ratio of likelihoods.

1.5 e)

In order to make our MCMC algorithm, we first make a function that creates our precision matrix

$$Q = \begin{bmatrix} 1 & -1 & & & & & \\ -1 & 2 & -1 & & & & \\ & -1 & 2 & -1 & & & \\ & & & \ddots & & & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 1 \end{bmatrix}$$

Next, we want to make a function that samples from our conditional distribution for σ_u^2, as found in section 1.3. We use Gibbs sampling on this parameter. In order to sample our τ, we use the fact that this will relate to its neighbouring parameter-values as follows:

- $\tau_1 \sim \mathcal{N}(\tau_2, \sigma_u)$,
- $\tau_t \sim \mathcal{N}(\frac{\tau_{t-1}+\tau_{t+1}}{2}, \sqrt{2}\sigma_u)$ for $t \in [2, 365]$,
- $\tau_{366} \sim \mathcal{N}(\tau_{365}, \sigma_u)$.

This is our conditional prior proposal distribution for τ_t, which we use in our Metropolis-Hastings step. For each iteration we check our proposed value for τ_t against our previous value, and accept with a probability given by our respective acceptance ratio.

```
# Make the precision matrix Q with dimension dxd
make_Q <- function(d) {
    Q = matrix(nrow = d, ncol = d)

    if (d == 2) {
        Q = matrix(c(1, -1, -1, 1), nrow = 2)
    } else if (d == 3) {
        Q = matrix(c(1, -1, 0, -1, 2, -1, 0, -1, 1), nrow = 3)
    } else {
        # Start with defining the two first and two last columns

        Q[, 1] = c(-1, -1, rep(0, d - 2))
        Q[, 2] = c(-1, -1, -1, rep(0, d - 3))
        Q[, d - 1] = c(rep(0, d - 3), -1, -1, -1)
```

```
        Q[, d] = c(rep(0, d - 2), -1, -1)

        # for the middle columns, we use this generic algorithm

        for (i in (3:(d - 2))) {
            Q[, i] = c(rep(0, i - 2), -1, -1, -1, rep(0, d - 1 - i))
        }

        # set the correct diagonal values last

        diag(Q) = c(1, rep(2, d - 2), 1)

    }

    return(Q)
}


# Sample sigma_u^2 using Gibbs sampling and the conditional distribution from
# 1c, which is an inverse gamma distribution
sample_sigma <- function(alpha, beta, tau) {

    # set shape and rate parameter according to what we found in 1c

    a = alpha + (365/2)
    b = (1/2) * sum((diff(tau))^2) + beta
    # diff(tau)
    sigma_sqrt = rgamma(1, shape = a, rate = b)
    return(1/sigma_sqrt)
}


# sample tau from a conditional normal
proposal <- function(t_old, i, sigma) {
    if (i == 1) {
        t_star = rnorm(1, t_old[i + 1], sigma)
    } else if (i == length(t_old)) {
        t_star = rnorm(1, t_old[i - 1], sigma)
    } else {
        t_star = rnorm(1, (t_old[i - 1] + t_old[i + 1])/2, sigma/sqrt(2))
    }
    return(t_star)
}


# the likelihood function of y given tau
binom_ll <- function(y, n, p) {
    return(sum(y * log(p) + (n - y) * log(1 - p)))
}


# Calculate the acceptance probability
acceptance_r <- function(y, n, pi_tau_new, pi_tau_old) {
```

```
    # Calculate p(y\verttau^*) and p(y\verttau)

    p_new = binom_ll(y, n, pi_tau_new)
    p_old = binom_ll(y, n, pi_tau_old)

    return(exp(p_new - p_old))

}


# MCMC algorithm n: number of samples tau: inital guess for tau data.y:
# observed values data.n: number of years sigma_alpha and sigma_beta:
# parameters for the inverse gamma we use to sample sigma

MCMC_samples <- function(n, tau, data.y, data.n, sigma_alpha = 2, sigma_beta = 0.05) {

    # set time-stamp
    t = proc.time()[3]

    yes = rep(0, length(tau))

    # matrix and vector to store each sample
    tau_mat = matrix(nrow = length(tau), ncol = n)

    sigma_vec = numeric(n)

    for (i in (1:n)) {

        # sample sigma^2
        sigma_2 = sample_sigma(sigma_alpha, sigma_beta, tau)

        sigma_vec[i] = sigma_2
        sigma = sqrt(sigma_2)

        # elementwise update of our parameter tau
        for (j in (1:(length(tau)))) {

            # sample a proposed value for tau
            tau_star = proposal(tau, j, sigma)

            # calculate acceptance probability for our new tau
            acc_prob = acceptance_r(data.y[j], data.n[j], pi_tau_new = inv.logit(tau_star),
                pi_tau_old = inv.logit(tau)[j])

            # accept new tau with probability given by the acceptance
            # probability
            if (runif(1) < min(acc_prob, 1)) {
                tau[j] = tau_star
                yes[j] = yes[j] + 1
            }

            tau_mat[j, i] = tau[j]
        }
```

```
    }
    # find the run-time of our algorithm
    time = proc.time()[3] - t

    return(list(sigmas = sigma_vec, taus = tau_mat, run_time = time, accept_ratio = yes/n))
}
```

```
tau_0 = rain$n.rain/rain$n.years
nsamples = 50000
sampled_MCMC = MCMC_samples(nsamples, tau_0, rain$n.rain, rain$n.years)

# save(sampled_MCMC, file = "MCMC_50000.rda")

# load("MCMC_50000.rda")

# We start by applying the inverse logit on our sampled values for tau
sampled_MCMC$taus = inv.logit(sampled_MCMC$taus)
```

We now consider the sampled values for σ_u^2, $\pi(\tau_1)$, $\pi(\tau_{201})$ and $\pi(\tau_{366})$.

```
par(mfrow=c(2,2))

plot(sampled_MCMC$sigmas[c(2000:50000)], type = 'l', xlab = 'Iterations', ylab = expression(sigma[u]^2))
plot(sampled_MCMC$taus[1,c(2000:50000)], type = 'l', xlab = 'Iterations', ylab = expression(paste(pi,'(
plot(sampled_MCMC$taus[201,c(2000:50000)], type = 'l', xlab = 'Iterations', ylab = expression(paste(pi,
plot(sampled_MCMC$taus[366,c(2000:50000)], type = 'l', xlab = 'Iterations', ylab = expression(paste(pi,
```

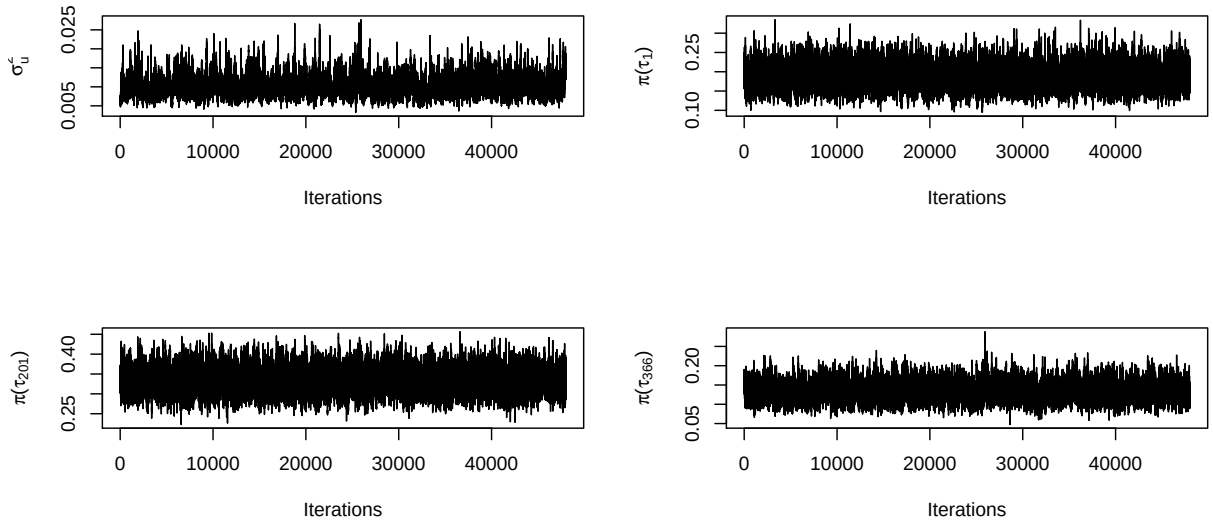


Figure 2: Trace plots of our estimated parameters.

From figure 2 we see that each of our traceplots indicate nice convergence of our Markov chain.

```
par(mfrow=c(2,2), cex.main = 0.7)

acf(sampled_MCMC$sigmas[c(2000:50000)],lag.max = 100, main = expression(paste('ACF for ', sigma[u]^2)))
acf(sampled_MCMC$taus[1,c(2000:50000)],lag.max = 100, main = expression(paste('ACF for ', pi,'(',tau[1]
acf(sampled_MCMC$taus[201,c(2000:50000)],lag.max = 100, main = expression(paste('ACF for ', pi,'(',tau[
acf(sampled_MCMC$taus[366,c(2000:50000)],lag.max = 100, main = expression(paste('ACF for ', pi,'(',tau[
```

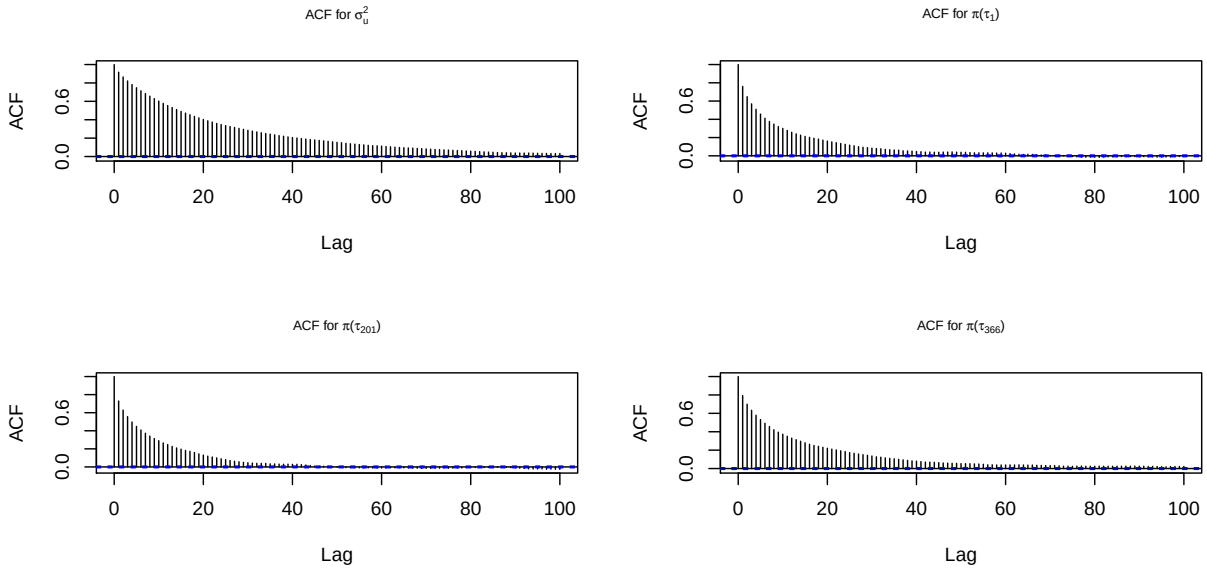


Figure 3: ACF for each of our parameters up to h=100.

From figure 3 we see that the correlation of each sampled value decrease in a nice pace, and that there is low correlation when the samples are distanced.

```
par(mfrow=c(2,2), cex.main = 0.7)

hist(sampled_MCMC$sigmas[c(2000:50000)], freq = F, nclass = 100, xlab = 'x', ylab = 'density',main = ex
hist(sampled_MCMC$taus[1,c(2000:50000)], freq = F, nclass = 100, xlab = 'x', ylab = 'density',main = ex
hist(sampled_MCMC$taus[201,c(2000:50000)], freq = F, nclass = 100, xlab = 'x', ylab = 'density',main =
hist(sampled_MCMC$taus[366,c(2000:50000)], freq = F, nclass = 100, xlab = 'x', ylab = 'density',main =
```

From figure 4 we see that density of our appear reasonable. Keep in mind that $\pi(\tau_t)$ is the inverse logit of a normally distributed variable, so our histogram seem sensible.

```
print(sampled_MCMC$run_time)
```

```
## elapsed
## 1160.61
```

```
print(paste("The MCMC algorithm took ", floor(sampled_MCMC$run_time/60), " minutes and ", round(((sampl
```

```
## [1] "The MCMC algorithm took  19  minutes and  20.6  seconds to produce  50000  number of samples"
```

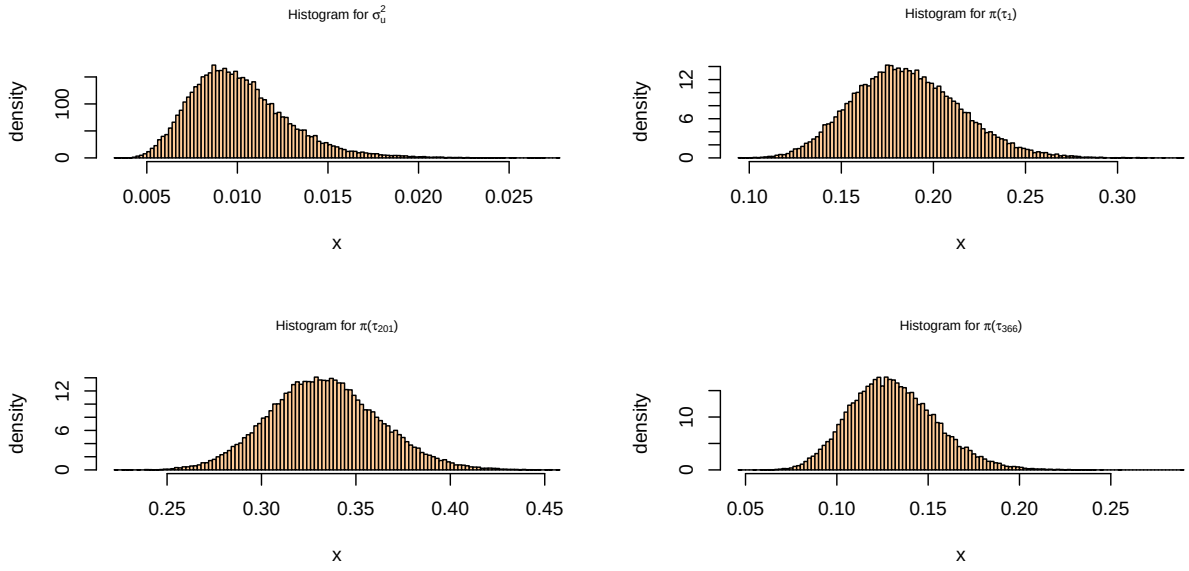


Figure 4: Histogram for each of our sampled parameters.

We make a matrix to get an overview of the estimated means, and 95% credible interval boundaries for our sampled parameters.

```
estimated_mat = matrix(nrow = 4, ncol = 3)
estimated_mat[1,2] = round(mean(sampled_MCMC$sigmas[c(2000:50000)]),3)
estimated_mat[2,2] = round(mean(sampled_MCMC$taus[1,c(2000:50000)]),3)
estimated_mat[3,2] = round(mean(sampled_MCMC$taus[201,c(2000:50000)]),3)
estimated_mat[4,2] = round(mean(sampled_MCMC$taus[366,c(2000:50000)]),3)


estimated_mat[1,c(1,3)] = round(quantile(sampled_MCMC$sigmas[c(2000:50000)], probs = c(0.025,0.975)),3)
estimated_mat[2,c(1,3)] = round(quantile(sampled_MCMC$taus[1,c(2000:50000)], probs = c(0.025,0.975)),3)
estimated_mat[3,c(1,3)] = round(quantile(sampled_MCMC$taus[201,c(2000:50000)], probs = c(0.025,0.975)),3
estimated_mat[4,c(1,3)] = round(quantile(sampled_MCMC$taus[366,c(2000:50000)], probs = c(0.025,0.975)),3

rownames(estimated_mat) = c('sigma_u^2','pi(tau_1)','pi(tau_201)','pi(tau_366)')
colnames(estimated_mat) = c('2.5% CI bound', 'mean','97.5% CI bound')

estimated_mat
```

```
##             2.5% CI bound  mean 97.5% CI bound
## sigma_u^2           0.006 0.010          0.016
## pi(tau_1)           0.134 0.186          0.247
## pi(tau_201)         0.279 0.333          0.393
## pi(tau_366)         0.089 0.131          0.181
```

```
taus_mean = rowMeans(sampled_MCMC$taus[,c(2000:50000)])
taus_quants = apply(sampled_MCMC$taus[,c(2000:50000)],MARGIN = 1, FUN = quantile,probs = c(0.025,0.975))

plot(rain$day, rain$n.rain/rain$n.years, xlab = 'Day', ylab = 'Frequency of rain', type = 'p',  pch = 1
```

```
lines(taus_mean, col = 'red')
lines(taus_quants[1,], col = 'darkblue')
lines(taus_quants[2,], col = 'darkblue')
```

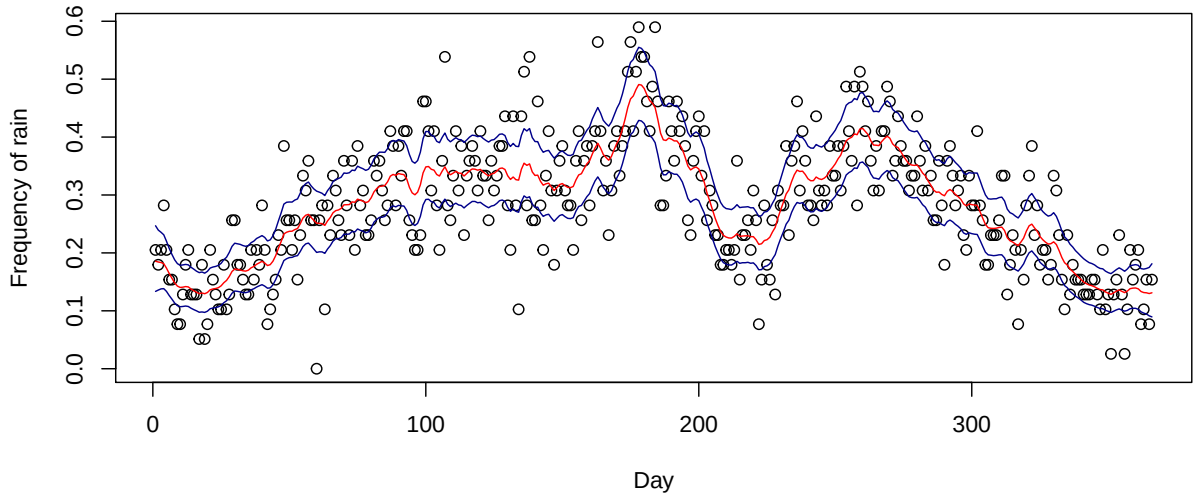


Figure 5: Our original dataset, and the empirical mean and credible interval from our MCMC algorithm.

Lastly, in figure 5 we compare our sampled values with the original data set. The trend of out samples seem to fit well with our original data set.

The acceptance ratio for our MCMC algorithm for the different τ_t can be seen in figure 6. We notice that the acceptance ratio is lower for the middle values of out parameter vector $\boldsymbol{\tau}$, but overall it is close to 1. Typically, we desire an acceptance ratio between 0.2 and 0.25, and hence it might be interesting to reconsider the proposal distribution.

```
accp_r_df = as.data.frame(cbind(rain$day,sampled_MCMC$accept_ratio))

ggplot(data = accp_r_df, aes(x=V1, y= V2)) +
  geom_smooth(span = 0.5) +
  geom_point() + labs(x = 't', y = 'acceptance ratio')
```

1.6 f)

Now, we want to use a block MCMC algorithm instead of a single site MCMC algorithm. First we define three matrices, which are sub-matrices of our precision matrix Q.

$$Q_1 = \begin{bmatrix} 1 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & & \ddots & & & \\ & & -1 & 2 & -1 & \\ & & & -1 & 2 & \end{bmatrix} \quad Q_M = \begin{bmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & & \ddots & & & \\ & & -1 & 2 & -1 & \\ & & & -1 & 2 & \end{bmatrix} \quad Q_{max} = \begin{bmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & & \ddots & & & \\ & & -1 & 2 & -1 & \\ & & & -1 & 1 & \end{bmatrix}$$

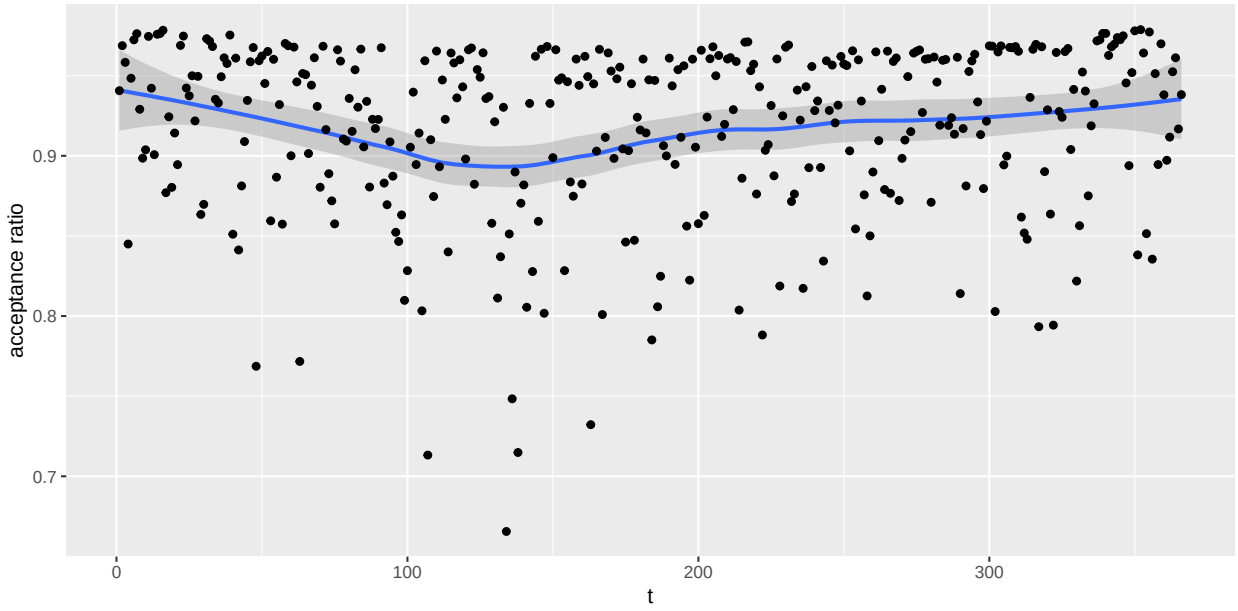


Figure 6: Plot of the acceptance ratio for each tau.

and then we calculate the inverse and Cholesky decomposition of each matrix before we start running the for-loops.

Next, we samples σ_u^2 as before, and start updating the $\tau_{(a,b)}$-vectors. We update $\tau_{(1,M)}$, then $\tau_{(M+1,2M)}$, ... and so on, by sampling a M-dimensional multinormal vector, with

$$\mu = -Q_k^{-1}\begin{bmatrix}\tau_{a-1} & 0 & 0 & \dots & 0 & \tau_{b+1}\end{bmatrix}^T, \quad \Sigma = Q_k(\sigma_u^2)^{-1}$$

where Q_k is one of the three we defined above, and depends on which part of the vector $\boldsymbol{\tau}$ we are currently updating. Then we calculate acceptance probability α as before, and accept the new sample with that probability.

```
# define a function that samples from a multivariate normal distribution
# using an already computed Cholesky decomposition
multinorm <- function(n,mu,cov_mat, choleski_mat){
  d = length(mu)
  M = matrix(NA, nrow = d, ncol = n)

  for (i in (1:n)){
    M[,i] = rnorm(d)
  }
  return(mu+choleski_mat%*%M)
}



block_MCMC_samples <- function(n, tau, data.y, data.n, M, sigma_alpha = 2, sigma_beta = 0.05){

  # set time-stamp
  t = proc.time()[3]
```

```
yes = numeric(ceiling(length(tau)/M))
# matrix and vector to store each sample
tau_mat = matrix(nrow = length(tau), ncol = n)

sigma_vec = numeric(n)

# define the matrix used to get our precision matrix
Q_1 = make_Q(M)
diag(Q_1) = c(1,rep(2,M-1))
Q_max = make_Q(M)
diag(Q_max) = c(rep(2,M-1),1)
Q_M = make_Q(M)
diag(Q_M) = rep(2,M)

# calculate the inverse outside the for-loop
Q_1.inv = solve(Q_1)
Q_max.inv = solve(Q_max)
Q_M.inv = solve(Q_M)

# calculate the Cholesky factorization outside the for-loop
Q_1.chol = t(chol(Q_1.inv))
Q_max.chol = t(chol(Q_max.inv))
Q_M.chol = t(chol(Q_M.inv))

for (i in (1:n)){

  # sample sigma^2
  sigma_2 = sample_sigma(sigma_alpha, sigma_beta, tau)
  sigma = sqrt(sigma_2)
  sigma_vec[i] = sigma_2

  # elementwise update of our parameter tau

  for (j in (1:ceiling(length(tau)/M))){
    a = (j-1)*M + 1
    b = j*M
    if(j == 1){
      tau_old = tau[c(a:b)]
      mu = Q_1.inv %*% c(rep(0,M-1),tau[b+1])
      cov_mat = (sigma_2)*Q_1.inv
      choleski_mat = sigma*Q_1.chol
    }

    else if(j == ceiling(length(tau)/M)){
      a = length(tau)-M+1
      b = length(tau)
      tau_old = tau[c(a:b)]
      mu = Q_max.inv %*% c(tau[a-1],rep(0,M-1))
      cov_mat = (sigma_2)*Q_max.inv
      choleski_mat = sigma*Q_max.chol
    }
```

```
      else{
        tau_old = tau[c(a:b)]
        mu = Q_M.inv %*% c(tau[a-1],rep(0,M-2),tau[b+1])
        cov_mat = (sigma_2)*Q_M.inv
        choleski_mat = sigma*Q_M.chol
      }



      # sample a proposed value for tau
      tau_star = multinorm(1,mu,cov_mat, choleski_mat)


      # calculate acceptance probability for our new tau
      acc_prob = acceptance_r(data.y[c(a:b)],data.n[c(a:b)],
                              pi_tau_new = inv.logit(tau_star),
                              pi_tau_old = inv.logit(tau_old))


      # accept new tau with probability given by the acceptance probability
      if (runif(1) < min(acc_prob,1)){
        tau[c(a:b)]=tau_star
        yes[j] = yes[j]+1
      }

      tau_mat[c(a:b),i] = tau[c(a:b)]
    }

  }
  # find the run-time of our algorithm
  time = proc.time()[3]-t

  return(list(sigmas = sigma_vec,
              taus = tau_mat,
              run_time = time,
              accept_ratio = yes))
}
```

```
tau_0 = rain$n.rain/rain$n.years
nsamples = 50000

sampled_block_MCMC_5 = block_MCMC_samples(nsamples, tau_0, rain$n.rain, rain$n.years, 5)

sampled_block_MCMC_10 = block_MCMC_samples(nsamples, tau_0, rain$n.rain, rain$n.years, 10)

sampled_block_MCMC_20 = block_MCMC_samples(nsamples, tau_0, rain$n.rain, rain$n.years, 20)



# change from tau to pi(tau)
sampled_block_MCMC_5$taus = inv.logit(sampled_block_MCMC_5$taus)

sampled_block_MCMC_10$taus = inv.logit(sampled_block_MCMC_10$taus)
```

```
sampled_block_MCMC_20$taus = inv.logit(sampled_block_MCMC_20$taus)
```

Below we compare the samples drawn using three different choices for the tuning parameter M, which decides the block size. We sampled using $M = 5, 10, 20$ and again we consider the four parameters σ_u^2, $\pi(\tau_1)$, $\pi(\tau_{201})$ and $\pi(\tau_{366})$.

```
par(mfrow = c(3, 4), cex.main = 0.7)

plot(sampled_block_MCMC_5$sigmas[c(2000:50000)], type = "l", xlab = "Iterations",
    ylab = expression(sigma[u]^2), main = "M = 5")
plot(sampled_block_MCMC_5$taus[1, c(2000:50000)], type = "l", xlab = "Iterations",
    ylab = expression(paste(pi, "(", tau[1], ")")), main = "M = 5")
plot(sampled_block_MCMC_5$taus[201, c(2000:50000)], type = "l", xlab = "Iterations",
    ylab = expression(paste(pi, "(", tau[201], ")")), main = "M = 5")
plot(sampled_block_MCMC_5$taus[366, c(2000:50000)], type = "l", xlab = "Iterations",
    ylab = expression(paste(pi, "(", tau[366], ")")), main = "M = 5")

plot(sampled_block_MCMC_10$sigmas[c(2000:50000)], type = "l", xlab = "Iterations",
    ylab = expression(sigma[u]^2), main = "M = 10")
plot(sampled_block_MCMC_10$taus[1, c(2000:50000)], type = "l", xlab = "Iterations",
    ylab = expression(paste(pi, "(", tau[1], ")")), main = "M = 10")
plot(sampled_block_MCMC_10$taus[201, c(2000:50000)], type = "l", xlab = "Iterations",
    ylab = expression(paste(pi, "(", tau[201], ")")), main = "M = 10")
plot(sampled_block_MCMC_10$taus[366, c(2000:50000)], type = "l", xlab = "Iterations",
    ylab = expression(paste(pi, "(", tau[366], ")")), main = "M = 10")

plot(sampled_block_MCMC_20$sigmas[c(2000:50000)], type = "l", xlab = "Iterations",
    ylab = expression(sigma[u]^2), main = "M = 20")
plot(sampled_block_MCMC_20$taus[1, c(2000:50000)], type = "l", xlab = "Iterations",
    ylab = expression(paste(pi, "(", tau[1], ")")), main = "M = 20")
plot(sampled_block_MCMC_20$taus[201, c(2000:50000)], type = "l", xlab = "Iterations",
    ylab = expression(paste(pi, "(", tau[201], ")")), main = "M = 20")
plot(sampled_block_MCMC_20$taus[366, c(2000:50000)], type = "l", xlab = "Iterations",
    ylab = expression(paste(pi, "(", tau[366], ")")), main = "M = 20")
```

```
par(mfrow = c(3, 4), cex.main = 0.7)

acf(sampled_block_MCMC_5$sigmas[c(2000:50000)], lag.max = 100, main = expression(paste("ACF for ",
    sigma[u]^2, " and M = 5")))
acf(sampled_block_MCMC_5$taus[1, c(2000:50000)], lag.max = 100, main = expression(paste("ACF for ",
    pi, "(", tau[1], ") and M = 5")))
acf(sampled_block_MCMC_5$taus[201, c(2000:50000)], lag.max = 100, main = expression(paste("ACF for ",
    pi, "(", tau[201], ") and M = 5")))
acf(sampled_block_MCMC_5$taus[366, c(2000:50000)], lag.max = 100, main = expression(paste("ACF for ",
    pi, "(", tau[366], ") and M = 5")))



acf(sampled_block_MCMC_10$sigmas[c(2000:50000)], lag.max = 100, main = expression(paste("ACF for ",
    sigma[u]^2, " and M = 10")))
acf(sampled_block_MCMC_10$taus[1, c(2000:50000)], lag.max = 100, main = expression(paste("ACF for ",
    pi, "(", tau[1], ") and M = 10")))
```

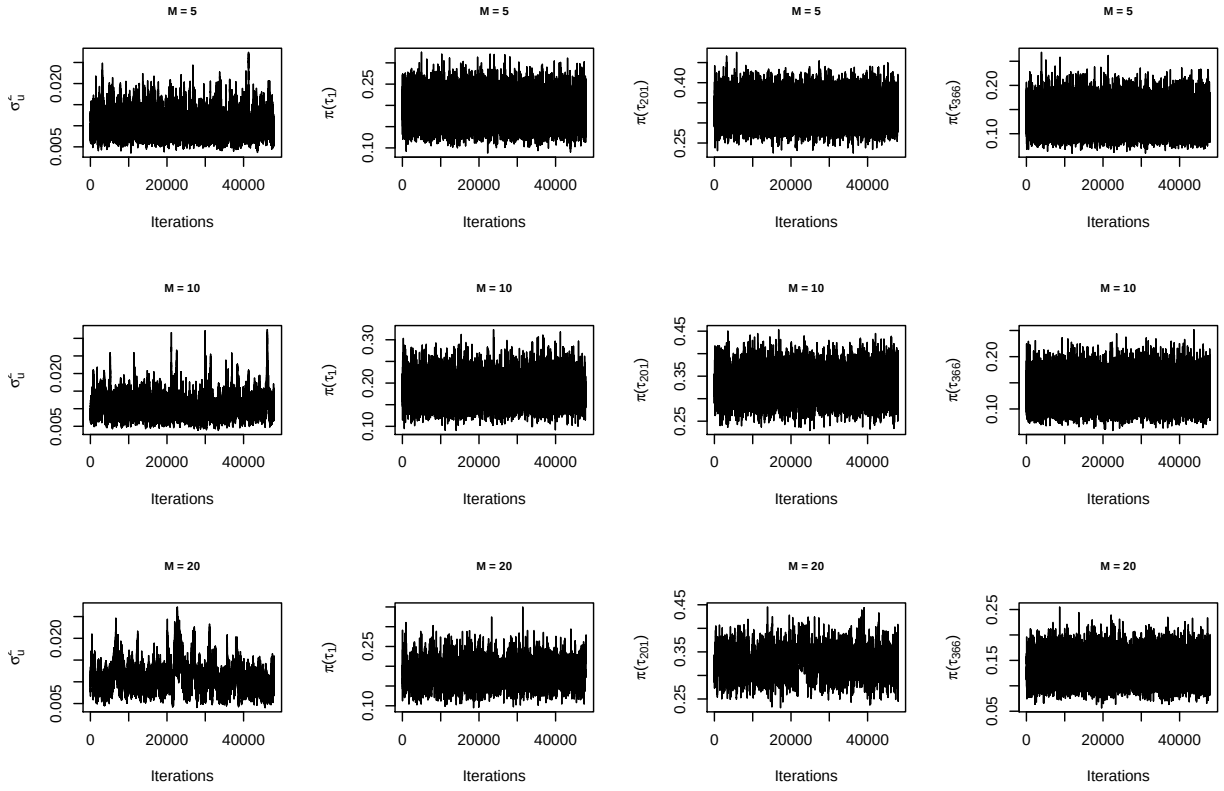


Figure 7: Traceplots for three different values of the tuning parameter M, which decides the block size

```
acf(sampled_block_MCMC_10$taus[201, c(2000:50000)], lag.max = 100, main = expression(paste("ACF for ",
    pi, "(", tau[201], ") and M = 10")))
acf(sampled_block_MCMC_10$taus[366, c(2000:50000)], lag.max = 100, main = expression(paste("ACF for ",
    pi, "(", tau[366], ") and M = 10")))


acf(sampled_block_MCMC_20$sigmas[c(2000:50000)], lag.max = 100, main = expression(paste("ACF for ",
    sigma[u]^2, " and M = 20")))
acf(sampled_block_MCMC_20$taus[1, c(2000:50000)], lag.max = 100, main = expression(paste("ACF for ",
    pi, "(", tau[1], ") and M = 20")))
acf(sampled_block_MCMC_20$taus[201, c(2000:50000)], lag.max = 100, main = expression(paste("ACF for ",
    pi, "(", tau[201], ") and M = 20")))
acf(sampled_block_MCMC_20$taus[366, c(2000:50000)], lag.max = 100, main = expression(paste("ACF for ",
    pi, "(", tau[366], ") and M = 20")))
```

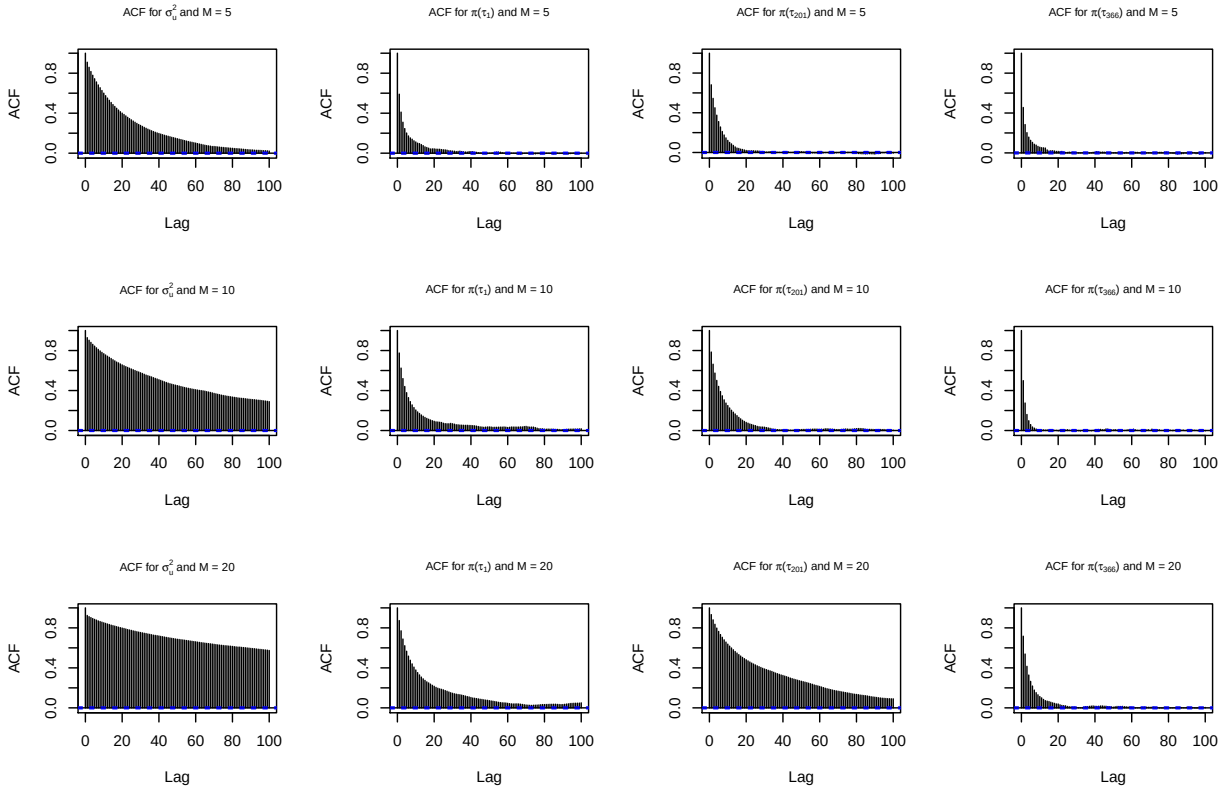


Figure 8: ACF of our parameters up to h=100 for three different values of M.

From figure 7 and figure 8 we see that when the block size $M = 20$, there is too high correlation in our samples and our samples don't converge as well as for lower choices of M. For $M = 5$ and $M = 10$ there is some differences, but both appear to work well and the samples seem to converge and show small correlation. The exception is the parameter σ_u^2, which show some more correlation than what we would like. To compare these samples further, we could look at the time and the acceptance ratio.

```
accp_r_df = as.data.frame(cbind(rain$day, sampled_block_MCMC_5$accept_ratio/nsamples,
    sampled_block_MCMC_10$accept_ratio/nsamples, sampled_block_MCMC_20$accept_ratio/nsamples))
```

```
ggplot(data = accp_r_df) + geom_smooth(aes(x = V1, y = V2), span = 0.5, col = "red") +
    geom_smooth(aes(x = V1, y = V3), span = 0.5) + geom_smooth(aes(x = V1, y = V4),
    span = 0.5, col = "darkgreen") + labs(x = "t", y = "acceptance ratio") + theme_minimal()
```

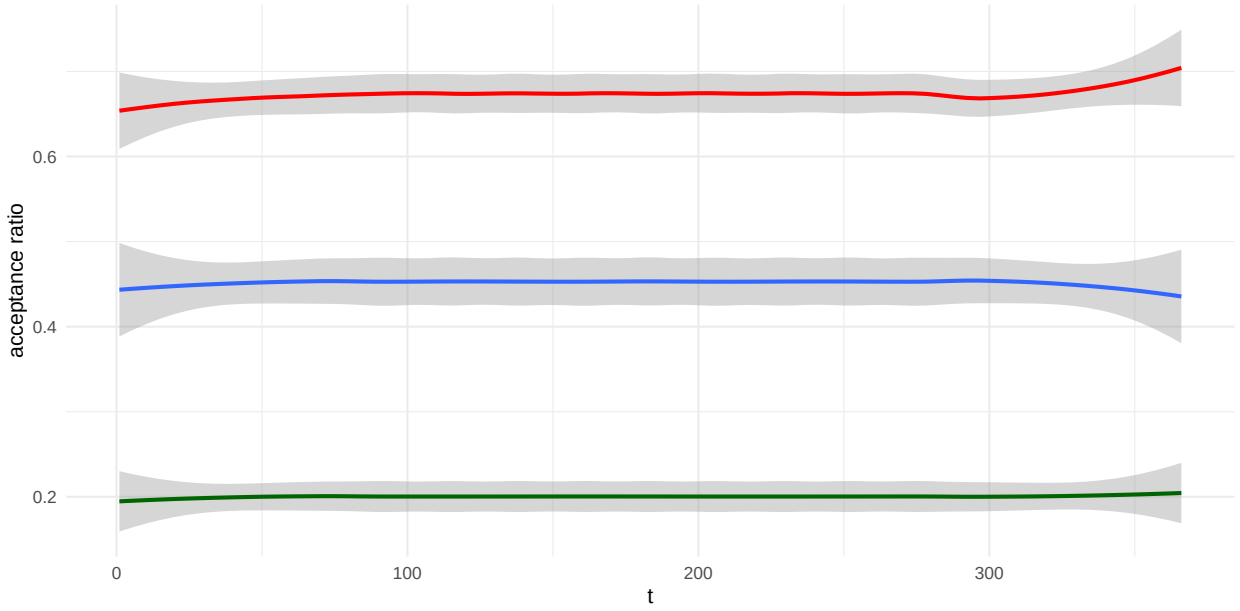


Figure 9: Plot of the acceptance ratio for M=5 (red), 10 (blue), and 20 (green)

```
time_5 = paste(floor(sampled_block_MCMC_5$run_time/60), "min,", round(((sampled_block_MCMC_5$run_time/6

time_10 = paste(floor(sampled_block_MCMC_10$run_time/60), "min,", round(((sampled_block_MCMC_10$run_tim

time_20 = paste(floor(sampled_block_MCMC_20$run_time/60), "min,", round(((sampled_block_MCMC_20$run_tim


block_time = c(time_5, time_10, time_20)

dim(block_time) = c(1,3)

rownames(block_time) = "Runtime"
colnames(block_time) = c("  M=5", "   M=10", "  M=20")

as.table(block_time)
```

```
##           M=5            M=10         M=20
## Runtime 3 min, 47.9 s 1 min, 30.5 s 0 min, 53 s
```

From the table above, and figure 9, we see that as M increase, the acceptance ratio decrease, but the running time decrease. Hence, larger M makes the algorithm run faster and more efficient. However, if the M is too large there is high correlation and we might need more samples to get converged samples, thus decreasing the efficiency.

From all this information, we conclude that if we wanted to tune this block-size parameter further, we would look at the sizes ranging from 3 to 9. In figure @ref(fig:comp_plots), we compare the block samples with $M = 5$ to the samples drawn using the single site algorithm.

```r
par(mfrow = c(6, 4), cex.main = 0.7)


######## traceplots ##########


plot(sampled_MCMC$sigmas[c(2000:50000)], type = "l", xlab = "Iterations", ylab = expression(sigma[u]^2)
    main = "Single site")
plot(sampled_MCMC$taus[1, c(2000:50000)], type = "l", xlab = "Iterations", ylab = expression(paste(pi,
    "(", tau[1], ")")), main = "Single site")
plot(sampled_MCMC$taus[201, c(2000:50000)], type = "l", xlab = "Iterations", ylab = expression(paste(pi
    "(", tau[201], ")")), main = "Single site")
plot(sampled_MCMC$taus[366, c(2000:50000)], type = "l", xlab = "Iterations", ylab = expression(paste(pi
    "(", tau[366], ")")), main = "Single site")

plot(sampled_block_MCMC_5$sigmas[c(2000:50000)], type = "l", xlab = "Iterations",
    ylab = expression(sigma[u]^2), main = "blocking, M = 5")
plot(sampled_block_MCMC_5$taus[1, c(2000:50000)], type = "l", xlab = "Iterations",
    ylab = expression(paste(pi, "(", tau[1], ")")), main = "blocking, M = 5")
plot(sampled_block_MCMC_5$taus[201, c(2000:50000)], type = "l", xlab = "Iterations",
    ylab = expression(paste(pi, "(", tau[201], ")")), main = "blocking, M = 5")
plot(sampled_block_MCMC_5$taus[366, c(2000:50000)], type = "l", xlab = "Iterations",
    ylab = expression(paste(pi, "(", tau[366], ")")), main = "blocking, M = 5")


######## acf ##########

acf(sampled_MCMC$sigmas[c(2000:50000)], lag.max = 100, main = expression(paste("ACF for ",
    sigma[u]^2, ", Single site")))
acf(sampled_MCMC$taus[1, c(2000:50000)], lag.max = 100, main = expression(paste("ACF for ",
    pi, "(", tau[1], "), Single site")))
acf(sampled_MCMC$taus[201, c(2000:50000)], lag.max = 100, main = expression(paste("ACF for ",
    pi, "(", tau[201], "), Single site")))
acf(sampled_MCMC$taus[366, c(2000:50000)], lag.max = 100, main = expression(paste("ACF for ",
    pi, "(", tau[366], "), Single site")))

acf(sampled_block_MCMC_5$sigmas[c(2000:50000)], lag.max = 100, main = expression(paste("ACF for ",
    sigma[u]^2, " and M = 5, blocking")))
acf(sampled_block_MCMC_5$taus[1, c(2000:50000)], lag.max = 100, main = expression(paste("ACF for ",
    pi, "(", tau[1], ") and M = 5, blocking")))
acf(sampled_block_MCMC_5$taus[201, c(2000:50000)], lag.max = 100, main = expression(paste("ACF for ",
    pi, "(", tau[201], ") and M = 5, blocking")))
acf(sampled_block_MCMC_5$taus[366, c(2000:50000)], lag.max = 100, main = expression(paste("ACF for ",
    pi, "(", tau[366], ") and M = 5, blocking")))


######## histograms ##########


hist(sampled_MCMC$sigmas[c(2000:50000)], freq = F, nclass = 100, xlab = "x", ylab = "density",
    main = expression(paste("Histogram for ", sigma[u]^2, ", Single site")), col = "#FFCC99")
hist(sampled_MCMC$taus[1, c(2000:50000)], freq = F, nclass = 100, xlab = "x", ylab = "density",
```

```
    main = expression(paste("Histogram for ", pi, "(", tau[1], "), Single site")),
    col = "#FFCCFF")
hist(sampled_MCMC$taus[201, c(2000:50000)], freq = F, nclass = 100, xlab = "x", ylab = "density",
    main = expression(paste("Histogram for ", pi, "(", tau[201], "), Single site")),
    col = "#CCFFCC")
hist(sampled_MCMC$taus[366, c(2000:50000)], freq = F, nclass = 100, xlab = "x", ylab = "density",
    main = expression(paste("Histogram for ", pi, "(", tau[366], "), Single site")),
    col = "#99CCFF")

hist(sampled_block_MCMC_5$sigmas[c(2000:50000)], freq = F, nclass = 100, xlab = "x",
    ylab = "density", main = expression(paste("Histogram for ", sigma[u]^2, ", blocking")),
    col = "#FFCC99")
hist(sampled_block_MCMC_5$taus[1, c(2000:50000)], freq = F, nclass = 100, xlab = "x",
    ylab = "density", main = expression(paste("Histogram for ", pi, "(", tau[1],
        "), blocking")), col = "#FFCCFF")
hist(sampled_block_MCMC_5$taus[201, c(2000:50000)], freq = F, nclass = 100, xlab = "x",
    ylab = "density", main = expression(paste("Histogram for ", pi, "(", tau[201],
        "), blocking")), col = "#CCFFCC")
hist(sampled_block_MCMC_5$taus[366, c(2000:50000)], freq = F, nclass = 100, xlab = "x",
    ylab = "density", main = expression(paste("Histogram for ", pi, "(", tau[366],
        "), blocking")), col = "#99CCFF")
```

From figure 10 we see that the samples are very similar, but the run time is far lower when using blocking. It takes between 19 and 20 to run the single site algorithm, whereas the blocking algorithm only takes between 3 and 4.

We make a matrix to get an overview of the estimated means, and 95% confidence interval boundaries for our sampled parameters.

```
estimated_mat = matrix(nrow = 8, ncol = 3)
estimated_mat[1,2] = round(mean(sampled_MCMC$sigmas[c(2000:50000)]),3)
estimated_mat[3,2] = round(mean(sampled_MCMC$taus[1,c(2000:50000)]),3)
estimated_mat[5,2] = round(mean(sampled_MCMC$taus[201,c(2000:50000)]),3)
estimated_mat[7,2] = round(mean(sampled_MCMC$taus[366,c(2000:50000)]),3)

estimated_mat[2,2] = round(mean(sampled_block_MCMC_5$sigmas[c(2000:50000)]),3)
estimated_mat[4,2] = round(mean(sampled_block_MCMC_5$taus[1,c(2000:50000)]),3)
estimated_mat[6,2] = round(mean(sampled_block_MCMC_5$taus[201,c(2000:50000)]),3)
estimated_mat[8,2] = round(mean(sampled_block_MCMC_5$taus[366,c(2000:50000)]),3)

estimated_mat[1,c(1,3)] = round(quantile(sampled_MCMC$sigmas[c(2000:50000)], probs = c(0.025,0.975)),3)
estimated_mat[3,c(1,3)] = round(quantile(sampled_MCMC$taus[1,c(2000:50000)], probs = c(0.025,0.975)),3)
estimated_mat[5,c(1,3)] = round(quantile(sampled_MCMC$taus[201,c(2000:50000)], probs = c(0.025,0.975)),3
estimated_mat[7,c(1,3)] = round(quantile(sampled_MCMC$taus[366,c(2000:50000)], probs = c(0.025,0.975)),3

estimated_mat[2,c(1,3)] = round(quantile(sampled_block_MCMC_5$sigmas[c(2000:50000)], probs = c(0.025,0.9
estimated_mat[4,c(1,3)] = round(quantile(sampled_block_MCMC_5$taus[1,c(2000:50000)], probs = c(0.025,0.9
estimated_mat[6,c(1,3)] = round(quantile(sampled_block_MCMC_5$taus[201,c(2000:50000)], probs = c(0.025,0
estimated_mat[8,c(1,3)] = round(quantile(sampled_block_MCMC_5$taus[366,c(2000:50000)], probs = c(0.025,0

rownames(estimated_mat) = c('sigma_u^2, single site','sigma_u^2, blocking','pi(tau_1), single site','pi
colnames(estimated_mat) = c('2.5% CI bound', 'mean','97.5% CI bound')
estimated_mat
```

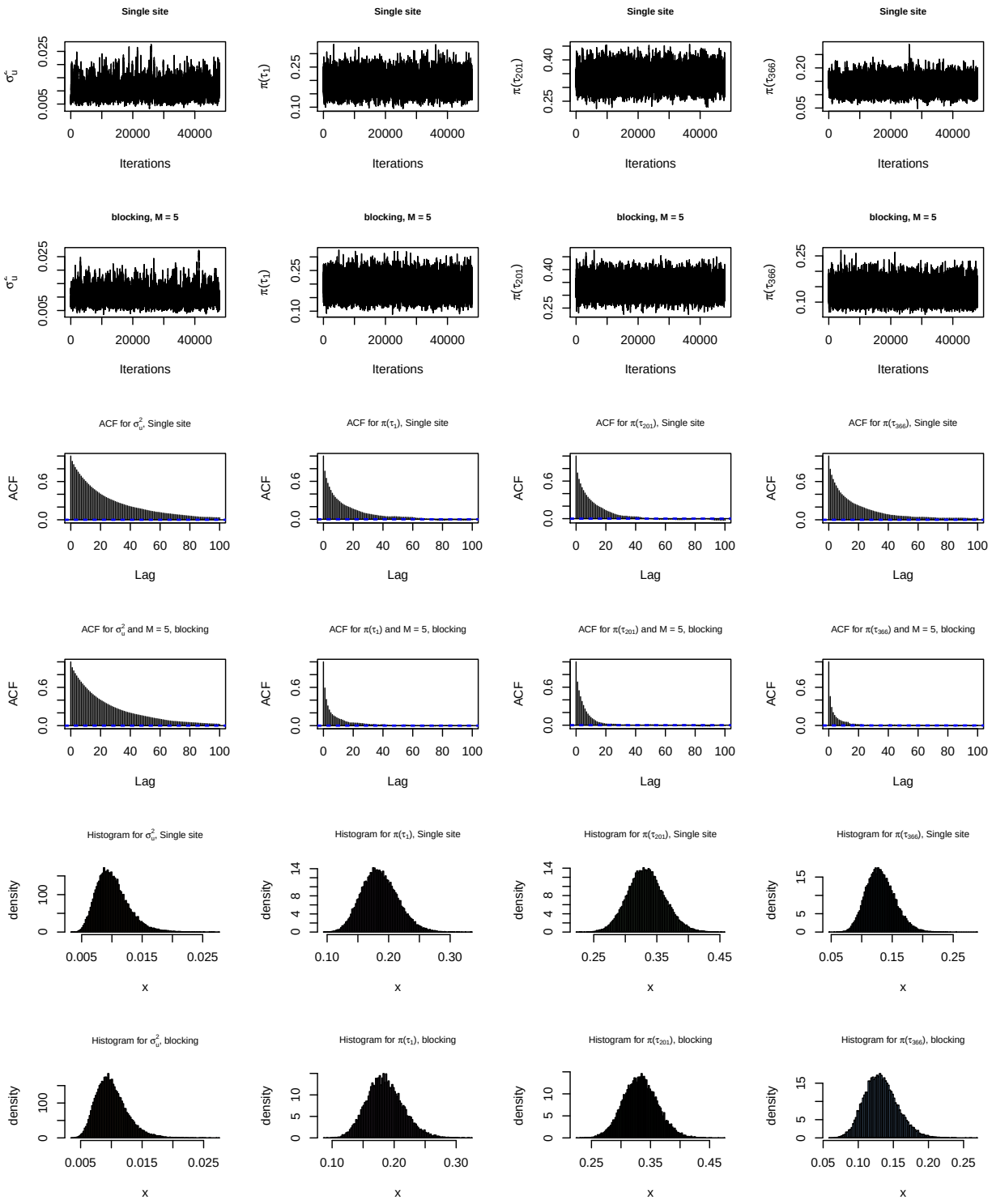


Figure 10: Comparison of the samples drawn using single-site MCMC, and block MCMC with M=5

```
##                           2.5% CI bound  mean 97.5% CI bound
## sigma_u^2, single site            0.006 0.010          0.016
## sigma_u^2, blocking               0.006 0.010          0.016
## pi(tau_1), single site            0.134 0.186          0.247
## pi(tau_1), blocking               0.132 0.185          0.248
## pi(tau_201), single site          0.279 0.333          0.393
## pi(tau_201), blocking             0.278 0.333          0.392
## pi(tau_366), single site          0.089 0.131          0.181
## pi(tau_366), blocking             0.091 0.132          0.183
```

From this table, we see that the single site and blocking algorithm are pretty consistent in their estimates.

2 Problem 2

2.1 a)

We will continue on the Tokyo rainfall data-set, but we will now be using INLA instead of MCMC-methods. When we use INLA we separate our problem into the following three sets:

The observations, which are assumed to be conditionally independent,

$$\boldsymbol{y}|\boldsymbol{\tau} \sim \prod p(y_t|\tau_t),$$

the latent field, which is assumed to be a GRMF,

$$\boldsymbol{\tau}|\sigma_u^2 \sim \mathcal{N}(0, Q(\sigma_u^2)^{-1}),$$

where our precision matrix is given by

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

and from task 1 we know the property of σ_u^2;

$$\sigma_u^2 \sim Gamma^{-1}(\alpha, \beta).$$

and from the documentation in INLA (inla.doc("rw1")) the prior is defined on theta in the following way.

$$\theta = log(\tau^*)$$

and because $\tau^{-1} = \sigma_u^2$ this implies $\theta = log(\frac{1}{\sigma_u^2})$

$$\implies \frac{1}{\sigma_u^2} \sim Gamma(\alpha, \beta)$$

$$\implies \theta = log\left(\frac{1}{\sigma_u^2}\right) \sim \text{log-gamma}(\alpha, \beta)$$

Which we can use for our hyperparameter in our INLA-function with $\alpha = 2, \beta = 0.05$. It's important to get this prior the same as the one we used in problem 1 for the results to be comparable.

```
library("INLA")

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

#Start timer.
t = proc.time()[3]

#mod includes a hyper argument where the prior is loggamma with alpha and beta,.
mod <- inla(n.rain ~ -1 + f(day, model = "rw1", constr=FALSE, hyper = list(theta = list(prior = "loggamm

#Shows how many seconds it takes to run the inla-function above.
INLA_t = round(proc.time()[3]-t, 3)


#Fitted values of mod
INLA_fv = data.frame(mod$summary.fitted.values)

#ID to use for plot
INLA_fv$ID <- seq.int(nrow(INLA_fv))
```

```
#ggplot of the fitted mean (pi(tau_i)) and the 95% credible interval of the INLA function.

plot_1 = ggplot(data = INLA_fv) + geom_line(aes(x = ID,  y = mean), color = 'black') +
  geom_line(aes(x = ID, y = X0.025quant), linetype = "dashed", color = 'orange') +
  geom_line(aes(x = ID, y = X0.975quant), linetype = "dashed", color = 'orange') +
  ggtitle("Rain") +
  ylab('Probability of days with over 1 mm rain') +  xlab("Day") +
  labs(title = NULL, caption = "Black = mean, dotted orange = 95% CI")
plot_1
```

In figure 11 we see that the probability that the rain goes over 1mm rain increases towards to rain-season. This corresponds nicely with the plot from the rain-dataset.

```
INLA_marg_hyper = mod$marginals.hyperpar[[1]]

INLA_marg_tau1 = mod$marginals.random$day$index.1

INLA_marg_tau201 = mod$marginals.random$day$index.201

INLA_marg_tau366 = mod$marginals.random$day$index.366

# Plot the marginals computed by INLA
par(mfrow = c(2, 2))
plot(INLA_marg_hyper, ylab = "Marginal density", main = expression(paste("1/", sigma[u]^2)))

plot(inv.logit(INLA_marg_tau1), ylab = "Marginal density", main = expression(paste(pi,
    "(", tau[1], ")")))

plot(inv.logit(INLA_marg_tau201), ylab = "Marginal density", main = expression(paste(pi,
    "(", tau[201], ")")))
```

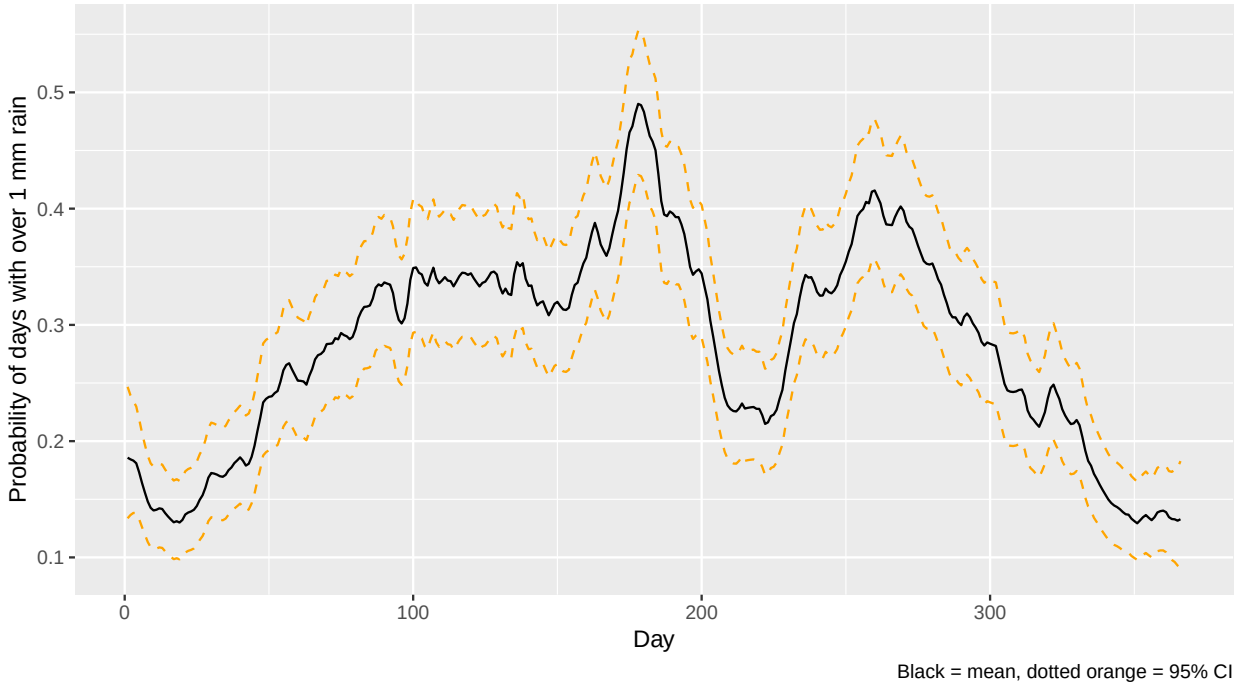


Figure 11: The fitted means and 95% credible intervals from the INLA model.

```
plot(inv.logit(INLA_marg_tau366), ylab = "Marginal density", main = expression(paste(pi,
    "(", tau[366], ")")))
```

In figure 12 we look at the marginals computed by the INLA model, and it seems to correspond well with the histogram we sampled from the MCMC algortihm earlier.

Another observation is that the computations time of the INLA function is a lot quicker than the MCMC-sampler, it takes only 3.61 seconds. And from our experience INLA is easier to implement than a MCMC-sampler.

Now lets compare the predictions ($\pi(\tau_t)$, $t = 1, 201, 366$) and uncertainties (σ_u^2) of our MCMC sampler with the results from INLA:

```
# Creating a comparison-matrix of INLA_fv and the MCMC sampler
comparison_mat = matrix(nrow = 4, ncol = 3)
comparison_mat[1, 1] = round((mod$summary.hyperpar$mean)^-1, 3)  #Adding sigma_u^2 from INLA
comparison_mat[1, 2] = estimated_mat[1, 2]  #Adding sigma_u^2 from MCMC
comparison_mat[1, 3] = estimated_mat[2, 2]

comparison_mat[2, 1] = round(INLA_fv$mean[1], 3)
comparison_mat[2, 2] = estimated_mat[3, 2]
comparison_mat[2, 3] = estimated_mat[4, 2]

comparison_mat[3, 1] = round(INLA_fv$mean[201], 3)
comparison_mat[3, 2] = estimated_mat[5, 2]
comparison_mat[3, 3] = estimated_mat[6, 2]

comparison_mat[4, 1] = round(INLA_fv$mean[366], 3)
comparison_mat[4, 2] = estimated_mat[7, 2]
```

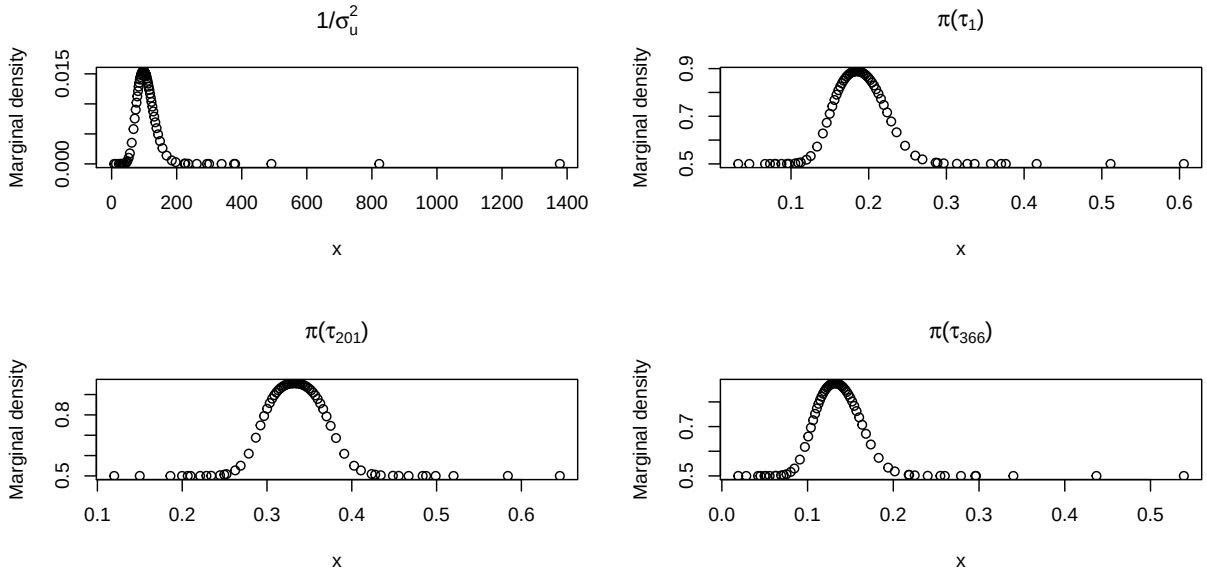


Figure 12: Plot of the marginal densities from the INLA model

```
comparison_mat[4, 3] = estimated_mat[8, 2]


rownames(comparison_mat) = c("sigma_u^2", "pi(tau_1)", "pi(tau_201)", "pi(tau_366)")
colnames(comparison_mat) = c("INLA", "MCMC single site", "MCMC blocking M=5")
comparison_mat
```

```
##              INLA MCMC single site MCMC blocking M=5
## sigma_u^2   0.009            0.010             0.010
## pi(tau_1)   0.186            0.186             0.185
## pi(tau_201) 0.333            0.333             0.333
## pi(tau_366) 0.133            0.131             0.132
```

From the table we see that the mean for the parameters from the different methods are very similar, indicating that all methods work as they are supposed to. However, the computational time for INLA is a clear advantage to the MCMC algorithm.

2.2 b)

In this task we keep the model the same where we use $n.rain \sim -1 + f(day...)$, but look at different options for "strategy" and "int.strategy" in control.inla.

```
##Running INLA-models for int.strategy = 'ccd', while using strategy = 'auto', 'gaussian', 'simplified

#Strategy = auto, int.strategy = ccd
control.inla_a = list(strategy="auto", int.strategy="ccd")

mod_a <- inla(n.rain ~ -1 + f(day, model="rw1", constr=FALSE, hyper = list(theta = list(prior = "loggamm
data=rain, Ntrials=n.years, control.compute=list(config = TRUE),
```

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


#Strategy = gaussian, int.strategy = ccd
control.inla_b = list(strategy="gaussian", int.strategy="ccd")

mod_b <- inla(n.rain ~ -1 + f(day, model="rw1", constr=FALSE, hyper = list(theta = list(prior = "loggamm
data=rain, Ntrials=n.years, control.compute=list(config = TRUE),
family="binomial", verbose=TRUE, control.inla=control.inla_b)


#Strategy = simplified.laplace, int.strategy = ccd
control.inla_c = list(strategy="simplified.laplace", int.strategy="ccd")

mod_c <- inla(n.rain ~ -1 + f(day, model="rw1", constr=FALSE, hyper = list(theta = list(prior = "loggamm
data=rain, Ntrials=n.years, control.compute=list(config = TRUE),
family="binomial", verbose=TRUE, control.inla=control.inla_c)


#Strategy = laplace, int.strategy = ccd
control.inla_d = list(strategy="laplace", int.strategy="ccd")

mod_d <- inla(n.rain ~ -1 + f(day, model="rw1", constr=FALSE, hyper = list(theta = list(prior = "loggamm
data=rain, Ntrials=n.years, control.compute=list(config = TRUE),
family="binomial", verbose=TRUE, control.inla=control.inla_d)


#Strategy = adaptive, int.strategy = ccd
control.inla_e = list(strategy="adaptive", int.strategy="ccd")

mod_e <- inla(n.rain ~ -1 + f(day, model="rw1", constr=FALSE, hyper = list(theta = list(prior = "loggamm
data=rain, Ntrials=n.years, control.compute=list(config = TRUE),
family="binomial", verbose=TRUE, control.inla=control.inla_e)

# Fitted values of the different models above.
fv_a = data.frame(mod_a$summary.fitted.values)              #fitted mean-values of mod_a
fv_b = data.frame(mod_b$summary.fitted.values)              #fitted mean-values of mod_b
fv_c = data.frame(mod_c$summary.fitted.values)              #fitted mean-values of mod_c
fv_d = data.frame(mod_d$summary.fitted.values)              #fitted mean-values of mod_d
fv_e = data.frame(mod_e$summary.fitted.values)              #fitted mean-values of mod_e
ID <- seq.int(nrow(data.frame(mod_a$summary.fitted.values)))#ID, to be used in gg-plot.
```

```
#Comparison plot of 5 different inputs for "strategy"
comp_plot = ggplot() + geom_line(data = fv_a, aes(x = ID, y = mean), color = "yellow") + geom_line(data=
  geom_line(data = fv_c, aes(x = ID, y = mean), color = "black") + geom_line(data = fv_d, aes(x = ID, y
  geom_line(data = fv_e, aes(x = ID, y = mean), color = "blue") +
  ylab('mean values') + xlab('Day')

comp_plot
```

In figure 13 we've plotted the mean-values $(\pi(\tau_i), i \in (1, 366))$ for each day of the year using int.strategy = 'ccd', but letting the argument 'strategy' differ between 'auto', 'gaussian', 'simplified.laplace', 'laplace' and 'adaptive'.

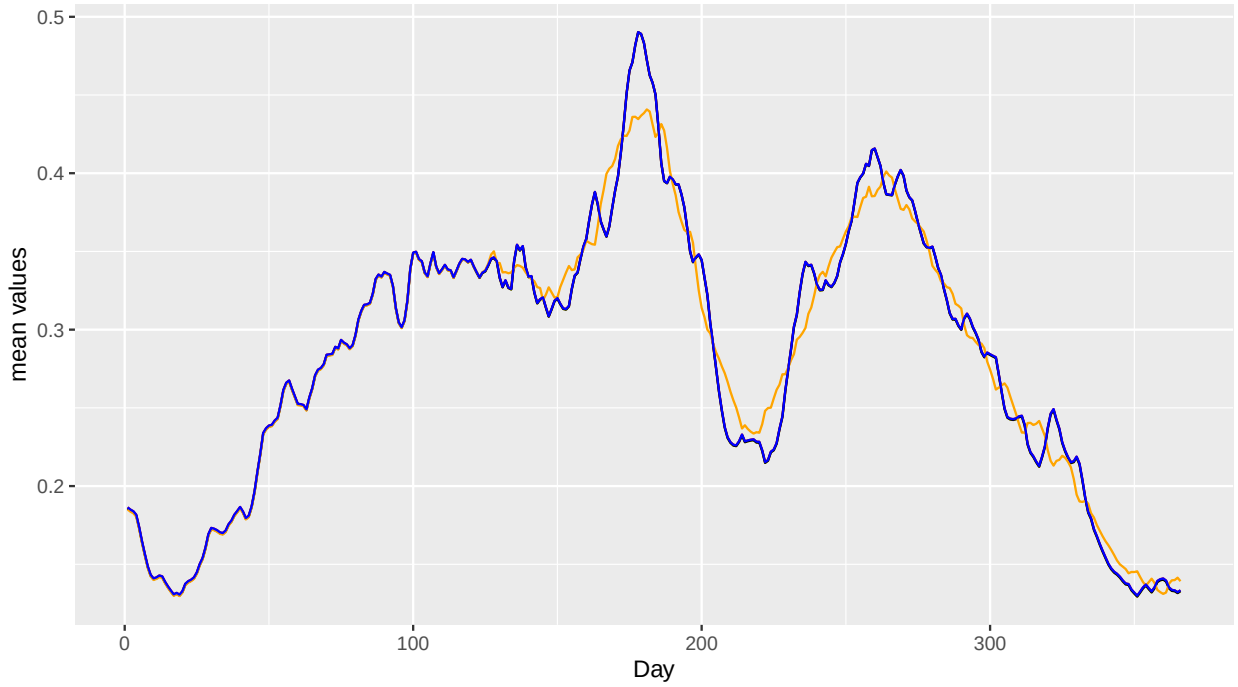


Figure 13: Plot that compares different parameters in control.inla

All the lines lie exactly on top of each other (the blue line hides the other ones), except the orange line (strategy = 'laplace'). The 'Laplace'-line and the rest are almost the same up till approximately day 130 (9th of May), then it's starting to differ from the other models. The same trend was observed when changing int.strategy to 'auto', 'grid' or 'eb' (did not inlude all these plots, because they are so similar).

The plot shows that when strategy is 'auto', 'gaussian', 'simplified.laplace' or 'adaptive' the mean is basically the same. But 'laplace' is a bit different. This difference can come from the fact that 'laplace' might be a bit more accurate in it's interpolation compared to 'simplified.laplace', but as we'll see further down, the computation time for strategy = 'laplace' is a bit longer than when strategy = 'simplified.laplace'. But it's only $\approx 0.2 - 0.4$ seconds longer when using 'laplace', so not a huge difference.

```
int.strategies = c('auto', 'ccd', 'grid', 'eb') #list of the different int.strategies.
times_list = numeric(4*5)                       #because we have 4 different int.strategies and 5 strat
pi_tau_values = numeric(4*5*3)                  #because we have three predictions, pi(tau_1), pi(tau_2
counter = 0                                     #start a counter to iterate through int.strategies list

#for loop to find the computation time of all the different 'strategy' and 'int.strategy' arguments
for (i in int.strategies){
  counter = counter + 1     #counter to i
  strat = i                 #variable to be the different int.strategies.

  #strategy = 'auto', and int.strategy is changing.
  control.inla_a = list(strategy="auto", int.strategy= strat)

  t_a = proc.time()[3]            #start timer.

  mod_a <- inla(n.rain ~ -1 + f(day, model="rw1", constr=FALSE, hyper = list(theta = list(prior = "logg
  data=rain, Ntrials=n.years, control.compute=list(config = TRUE),
  family="binomial", verbose=TRUE, control.inla=control.inla_a)
```

```
time_a = proc.time()[3] - t_a   #end timer. Do the same for the different 'strategy'.

times_list[counter] = time_a    #adding time_a to the times_list.

#strategy = 'gaussian', and int.strategy is changing.
control.inla_b = list(strategy="gaussian", int.strategy=strat)

counter = counter + 1           #increasing counter, for adding the 'counter'-th element to times_lis

t_b = proc.time()[3]

mod_b <- inla(n.rain ~ -1 + f(day, model="rw1", constr=FALSE, hyper = list(theta = list(prior = "logga
data=rain, Ntrials=n.years, control.compute=list(config = TRUE),
family="binomial", verbose=TRUE, control.inla=control.inla_b)

time_b = proc.time()[3] - t_b

times_list[counter] = time_b

#strategy = 'simplified.laplace', and int.strategy is changing.
control.inla_c = list(strategy="simplified.laplace", int.strategy=strat)

counter = counter + 1

t_c = proc.time()[3]

mod_c <- inla(n.rain ~ -1 + f(day, model="rw1", constr=FALSE, hyper = list(theta = list(prior = "logga
data=rain, Ntrials=n.years, control.compute=list(config = TRUE),
family="binomial", verbose=TRUE, control.inla=control.inla_c)

time_c = proc.time()[3] - t_c

times_list[counter] = time_c

#strategy = 'laplace', and int.strategy is changing.
control.inla_d = list(strategy="laplace", int.strategy=strat)

counter = counter + 1

t_d = proc.time()[3]

mod_d <- inla(n.rain ~ -1 + f(day, model="rw1", constr=FALSE, hyper = list(theta = list(prior = "logga
data=rain, Ntrials=n.years, control.compute=list(config = TRUE),
family="binomial", verbose=TRUE, control.inla=control.inla_d)

time_d = proc.time()[3] - t_d

times_list[counter] = time_d


#strategy = 'adaptive', and int.strategy is changing.
control.inla_e = list(strategy="adaptive", int.strategy=strat)
```

```
  counter = counter + 1

  t_e = proc.time()[3]

  mod_e <- inla(n.rain ~ -1 + f(day, model="rw1", constr=FALSE, hyper = list(theta = list(prior = "logg
  data=rain, Ntrials=n.years, control.compute=list(config = TRUE),
  family="binomial", verbose=TRUE, control.inla=control.inla_e)

  time_e = proc.time()[3] - t_e
  time_e
  times_list[counter] = time_e

}
```

```
#comparison matrix for the computation-time using all the different strategies available.
comparison_time = matrix(nrow = 5, ncol = 4)

#adding all the different computation-times to the matrix defined above.
comparison_time[1, 1] = times_list[1]
comparison_time[2, 1] = times_list[2]
comparison_time[3, 1] = times_list[3]
comparison_time[4, 1] = times_list[4]
comparison_time[5, 1] = times_list[5]
comparison_time[1, 2] = times_list[6]
comparison_time[2, 2] = times_list[7]
comparison_time[3, 2] = times_list[8]
comparison_time[4, 2] = times_list[9]
comparison_time[5, 2] = times_list[10]
comparison_time[1, 3] = times_list[11]
comparison_time[2, 3] = times_list[12]
comparison_time[3, 3] = times_list[13]
comparison_time[4, 3] = times_list[14]
comparison_time[5, 3] = times_list[15]
comparison_time[1, 4] = times_list[16]
comparison_time[2, 4] = times_list[17]
comparison_time[3, 4] = times_list[18]
comparison_time[4, 4] = times_list[19]
comparison_time[5, 4] = times_list[20]

#Naming rows as the different 'strategy' arguments.
rownames(comparison_time) = c('auto','gaussian','simplified laplace', 'laplace', 'adaptive')
#Naming coloumns as the different 'int.strategy' arguments.
colnames(comparison_time) = c('auto', 'ccd', 'grid', 'eb')

#The matrix with all the computation times.
comparison_time
```

```
##                    auto  ccd grid   eb
## auto               2.46 2.71 3.19 2.39
## gaussian           2.71 2.64 2.68 2.44
## simplified laplace 2.84 2.61 2.77 2.62
## laplace            5.17 3.98 5.70 3.16
## adaptive           2.36 3.22 2.63 2.64
```

In the matrix above we see the computation times where the different 'strategy' inputs is the rows, and int.strategy is the different columns. The computation time is generally low. The computations time when using strategy = 'simplified.laplace' and int.strategy = 'ccd' have a good low computation time. Especially compared to the MCMC-sampler.

NOTE: Did not use int.strategy = 'user' or 'user.std' or 'user.expert' because they require the integration design in the argument 'int.design'.

2.3 c)

Now we consider another model, mod2, where we include an intercept. In model 1 (from task 2a) we excluded the intercept. And in mod2 the constr = TRUE, but in mod constr = False. This means that we add a sum-to-zero constraint on the latent effect for mod2. Here we assume that the prior for σ_u^2 is the same for both models.

```
#Same control arguments as in model 1.
control.inla = list(strategy="simplified.laplace", int.strategy="ccd")

#model 2 where there is an intercept and sum-to-zero contraint is TRUE.
mod2 <- inla(n.rain ~ f(day, model="rw1", constr = TRUE, hyper = list(theta = list(prior="loggamma", pa

#Fitted values of mod2
INLA_fv2 <- data.frame(mod2$summary.fitted.values)
```

```
#Creating a comparison-matrix, to compare pi values between INLA_fv and INLA_fv2.
comparison_mat = matrix(nrow = 3, ncol = 2)

#adding the pi(tau)-values for 1, 201 and 366.
comparison_mat[1,1] = INLA_fv$mean[1]
comparison_mat[1,2] = INLA_fv2$mean[1]
comparison_mat[2,1] = INLA_fv$mean[201]
comparison_mat[2,2] = INLA_fv2$mean[201]
comparison_mat[3,1] = INLA_fv$mean[366]
comparison_mat[3,2] = INLA_fv2$mean[366]

#naming the rows and coloumns of the matrix.
rownames(comparison_mat) = c('pi(tau_1)','pi(tau_201)','pi(tau_366)')
colnames(comparison_mat) = c('INLA: mod', 'INLA: mod2')

#Run the comparion_mat
comparison_mat
```

```
##             INLA: mod INLA: mod2
## pi(tau_1)   0.1857548  0.1857560
## pi(tau_201) 0.3329584  0.3329575
## pi(tau_366) 0.1328158  0.1328181
```

From the comparison-matrix above we see that he fitted values of $\pi(\tau_t), t = 1, 201, 366$ from mod and mod2 are the same up to the 5th decimal. This is because mathematically the models can be thought of equivalent, because excluding the intercept and no sum-to-zero contraint (mod) yields the same predictions as including the the intercept and adding a sum-to-zero contraint (mod2).

But we will observe in a later plot, is that there will be shift in the tau-values for model 2 being the size of the intercept.

When we're looking at the model in a) and mod2 we can write them in the following way:

$$mod_1 : y_t|\tau_t, n_t, \sigma_t \sim Bin(n_t, \pi(\tau_t))$$

and

$$mod_2 : y_t|\tau_t, n_t, \sigma_t, \beta_0 \sim Bin(n_t, \pi(\beta_0 + \tau_t))$$

Where mod_1 is without an intercept β_0 term, but mod_2 has it included. This can be confirmed by using the function "model$summary.fixed":

```
#Check the estimated intercept of model 2
intercept_mod2 = mod2$summary.fixed
intercept_mod2[1]
```

```
##                    mean
## (Intercept) -0.9876671
```

```
#Confirm that their is no intercept in model 1
intercept_mod = mod$summary.fixed
intercept_mod
```

```
## data frame with 0 columns and 0 rows
```

As expected intercept_mod2 return a value for the intercept, but for mod we get a data frame with 0 rows and 0 coloumns because the intercept is not included in the model.

So we see that the linear predictor in model 2 is different, and includes an intercept (β_0). Let's call the linear predictors for η_i, and can then we can write them as:

$$i = 1 \text{ (model 1)} : \text{The linear predictor is: } \eta_1 = \tau_t$$

$$i = 2 \text{ (model 2)} : \text{The linear predictor is: } \eta_2 = \beta_0 + \tau_t$$

This is the main mathematical difference of the models, and can be observed when we plot the tau-values of the the models (see plot below).

```
#Retrieve the tau-values from mod and mod2.
random_mod2 = data.frame(mod2$summary.random)
random_mod = data.frame(mod$summary.random)

#ID to be used in x, in ggplot.
random_mod2$ID <- seq.int(nrow(random_mod2))
random_mod$ID <- seq.int(nrow(random_mod))

#plot of tau-values for mod and mod2
plot_4 = ggplot() + geom_line(data =random_mod2, aes(x= ID,  y = day.mean)) + geom_line(data=random_mod
  ylab('Tau-value') +  xlab("Day") +
  labs(subtitle = "Tokyo rainfall dataset",
       caption = "orange = mod (without intercept), black = mod2 (with intercept)")
plot_4
```

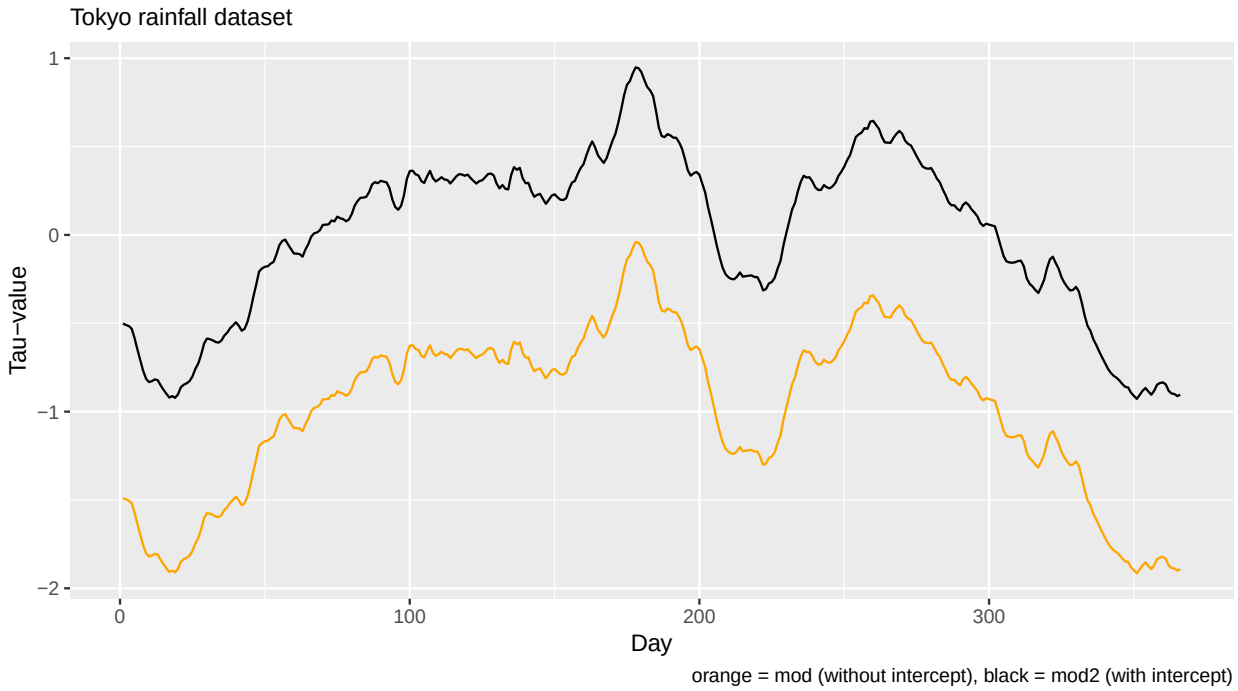


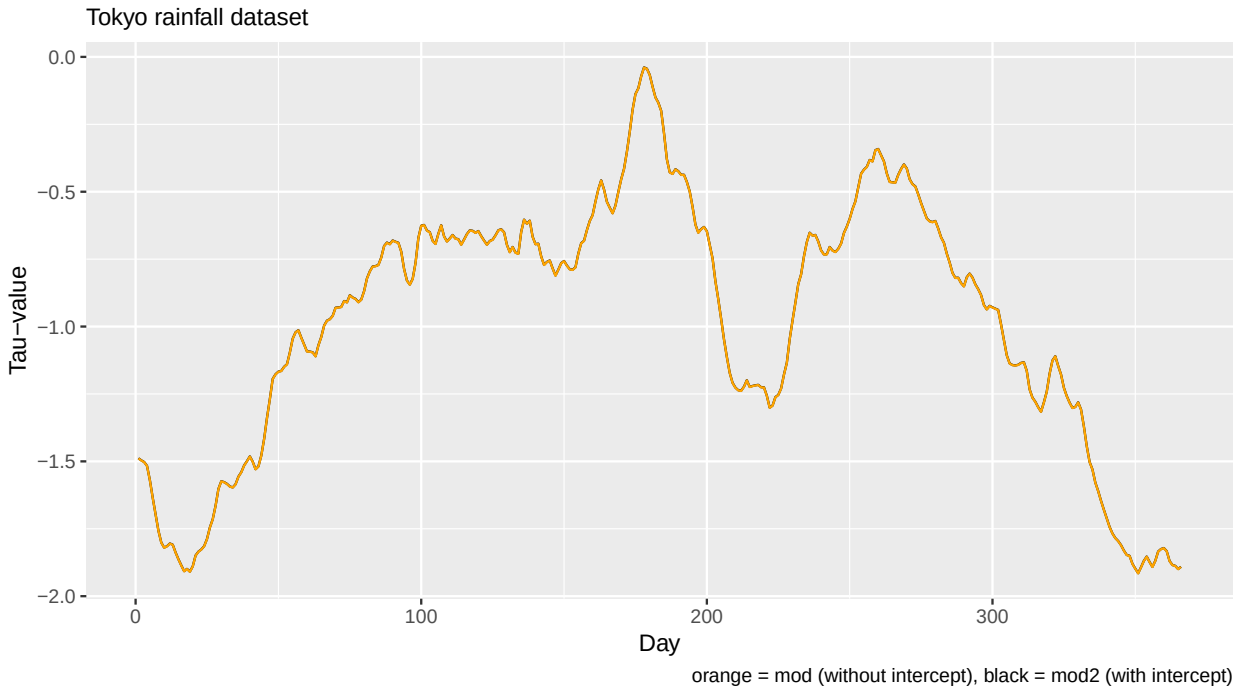
Figure 14: This plots show a comparison of the INLA model with and without an intercept

Figure 14 confirms a shift in the tau-values with approximately $-0,987$, the intercept value (β_0).

So $mod = f_1$(no intercept, constr = FALSE) and $mod_2 = f_2$(with intercept, constr = TRUE) are not mathematically the same, but they are equivalent. This can be shown if we use the same plot as above, but for mod_2 we take the tau-values and add $\beta_0 \approx -0.987$ for the y-values. Then we get the following:

```
# Plot where y = day.mean is changed to y = -0.987 + day.mean, to see that the
# to plots align.
plot_5 = ggplot() + geom_line(data = random_mod2, aes(x = ID, y = -0.987 + day.mean)) +
    geom_line(data = random_mod, aes(x = ID, y = day.mean), color = "orange") + ylab("Tau-value") +
    xlab("Day") + labs(subtitle = "Tokyo rainfall dataset", caption = "orange = mod (without intercept)

plot_5
```



As we can see from figure **??**, the black and orange line lie perfectly on top of each other. Just as predicted.

This confirms that the two models of interest can be thought of as equivalent, and is the reason why the mean values are so similar.