Project 2

Computer Intensive Statistical Methods

Erling Fause Steen og Christian Oppegård Moen

08 03 2022

Contents

Introduction

The Tokyo rainfall dataset contain the amount of rainfall for each of the 366 days (including February 29.) for several years. We will consider a portion of this dataset, specifically from $1951 - 1989$, such that for each day $t \neq 60$ we have $n_t = 39$ observations and for $t = 60$, February 39, we have $n_t = 10$ observation. For each day we have the response $y_t = 0, 1, 2, ..., n_t$ being the amount of times the rainfall exceeded 1mm over the given period, given by

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

Here, $\pi(\tau_t)$ is the probability of rainfall exceeding 1mm and τ_t is the logit probability of exceedence. For this project we assume conditional independence among the $y_t|\tau_t \ \forall t = 1, 2, ..., 366$.

We will apply a Bayesian hierarchical model to the dataset, using a random walk of order 1 (RW1) to model the trend. For the model we will implement a Markov chain Monte Carlo (MCMC) sampler for the posterior using Metropolis-Hastings (MH) and Gibbs steps for specific parameters. Then we will investigate the accuracy and computational speed of the implementation compared to the built in method `INLA` in R.

Problem 1

a) Display

In Figure 1 we see the number of times the rainfall has exceeded 1mm. There seem to be fewer days in the start of the year and in the end of the year with an amount of rainfall exceeding 1 mm. This is in January and December. The number of days steadily increases until the begining of the summer which seems to be the period with the most days with an amount of rainfall over 1 mm. Then, the amount of days decreases during july and august before increasing during the autumn. There also seem to be fluctuations on a daily basis from the just mentiod trend in the data. The red dot is the observation for February 29.

```
load("./rain.rda")
day     = rain$day
n.rain  = rain$n.rain
n.years = rain$n.years
##Plotting the data
ggplot(data=rain, mapping=aes(x=day, y=n.rain)) +
  geom_line() +
  xlab("Day") +
  ylab("Number of days with more than 1mm rain") +
  geom_point(aes(x=day[60], y=n.rain[60]), colour="red")
```

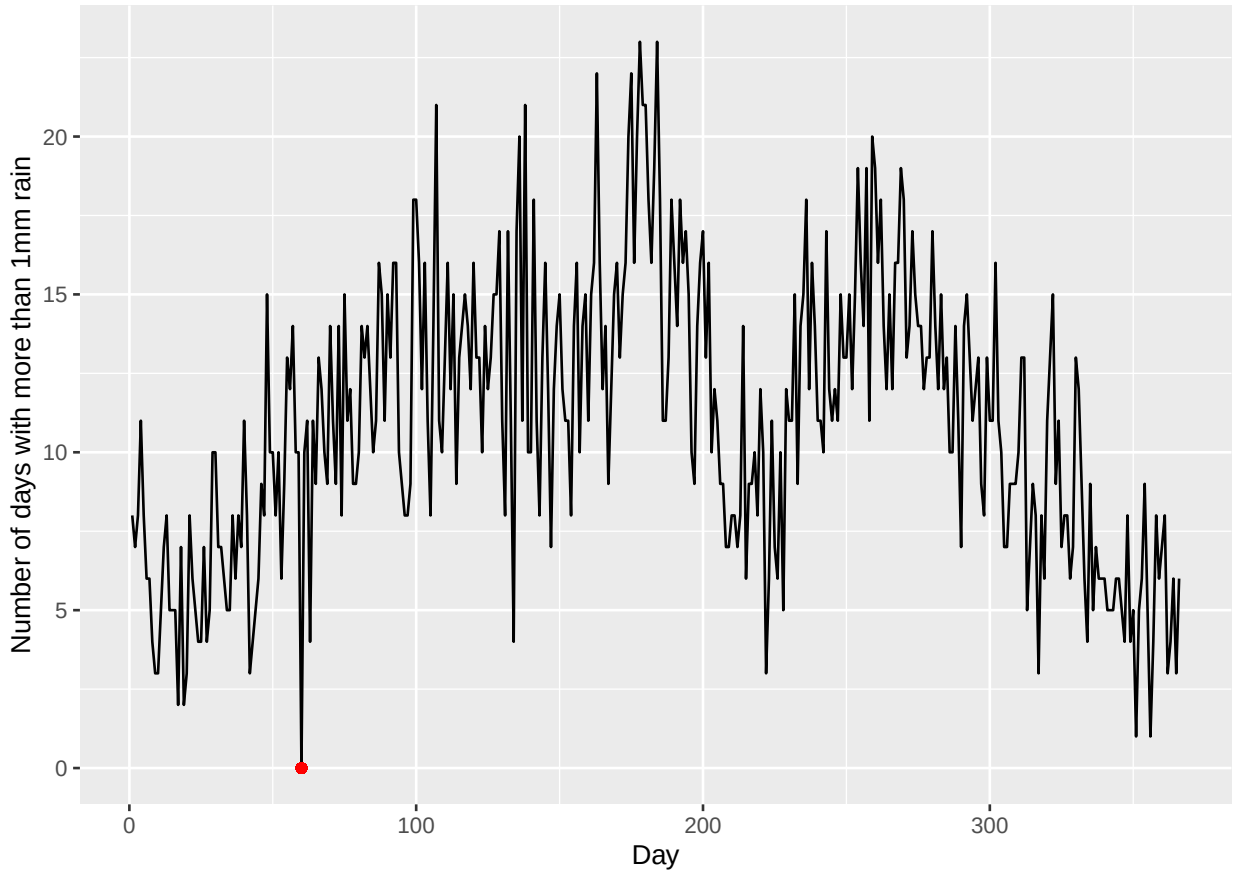


Figure 1: The Tokyo Rainfall dataset

b) Likelihood

The likelihood of Equation (1) is given by bc of conditional indep

$$\begin{aligned} L(\pi(\boldsymbol{\tau})) &= \prod_{i=1}^{T} \begin{pmatrix} n_t \\ y_t \end{pmatrix} \pi(\tau_t)^{y_t}(1-\pi(\tau_t)^{n_t-y_t} \\ &\propto \prod_{i=1}^{T} \pi(\tau_t)^{y_t}(1-\pi(\tau_t))^{n_t-y_t} \\ &= \prod_{t=1}^{T} \left(\frac{\exp(\tau_t)}{1+\exp(\tau_t)}\right)^{y_t} \left(1-\frac{\exp(\tau_t)}{1+\exp(\tau_t)}\right)^{n_t-y_t}, \end{aligned}$$

where $\boldsymbol{\tau} = (\tau_1, ..., \tau_T)$, $y_t = 1, 2, ..., 39$ and $n_t = 39$ for $t \neq 60$, and $y_t = 1, 2, ..., 10$ and $n_t = 10$ for $t \neq 60$.

c) Posterior

As briefly mentioned in the introduction we need the posterior $P(\sigma^2|\tau, y)$ for the Gibbs step in our implementation, given by

$$\begin{aligned} P(\sigma^2|\boldsymbol{\tau}, \boldsymbol{y}) &= \frac{P(\sigma_u^2, \boldsymbol{\tau}, \boldsymbol{y})}{P(\boldsymbol{\tau}, \boldsymbol{y})} \\ &\propto P(\boldsymbol{y}|\sigma_u^2, \boldsymbol{\tau}) P(\sigma_u^2, \boldsymbol{\tau}) \\ &= P(\boldsymbol{y}|\sigma_u^2, \boldsymbol{\tau}) P(\tau|\sigma_u^2) P(\sigma_u^2), \end{aligned}$$

where $\boldsymbol{y} = (y_1, ..., y_T)^T$, $\boldsymbol{\tau} = (\tau_1, ..., \tau_T)$ and $P(\boldsymbol{y}|\sigma_u^2, \boldsymbol{\tau}) = L(\pi(\boldsymbol{\tau}))$. Based on model assumptions mentioned in the introduction, we have $\tau_t \sim \tau_{t-1} + u_t$ for $u_t \overset{iid}{\sim} \mathcal{N}(0, \sigma_u^2)$ so that

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

We place an inverse gamma prior (IG) on σ_u^2 given by

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

Then, the posterior is

$$\begin{aligned} P(\sigma^2|\boldsymbol{\tau}, \boldsymbol{y}) &= \underbrace{\prod_{t=1}^{T} \left(\frac{\exp(\tau_t)}{1+\exp(\tau_t)}\right)^{y_t} \left(1-\frac{\exp(\tau_t)}{1+\exp(\tau_t)}\right)^{n_t-y_t}}_{\text{Constant w.r.t. } \sigma^2} \cdot \\ &\quad \prod_{t=1}^{T} \frac{1}{\sigma_u} \exp\left\{-\frac{1}{2\sigma_u^2}(\tau_t - \tau_{t-1})^2\right\} \cdot \frac{\beta^\alpha}{\Gamma(\alpha)} \left(\frac{1}{\sigma_u^2}\right)^{\alpha+1} exp\left\{-\frac{\beta}{\sigma_u^2}\right\} \\ &\propto \prod_{t=1}^{T} \frac{1}{\sigma_u} \exp\left\{-\frac{1}{2\sigma_u^2}(\tau_t - \tau_{t-1})^2\right\} \cdot \frac{\beta^\alpha}{\Gamma(\alpha)} \left(\frac{1}{\sigma_u^2}\right)^{\alpha+1} exp\left\{-\frac{\beta}{\sigma_u^2}\right\} \\ &= \frac{1}{\sigma_u^{T-1}} exp\left\{-\frac{1}{2\sigma_u^2}\boldsymbol{\tau}\boldsymbol{Q}\boldsymbol{\tau}\right\} \cdot \frac{\beta^\alpha}{\Gamma(\alpha)} \left(\frac{1}{\sigma_u^2}\right)^{\alpha+1} exp\left\{-\frac{\beta}{\sigma_u^2}\right\} \\ &\propto \left(\frac{1}{\sigma_u^2}\right)^{\alpha+\frac{T-1}{2}+1} exp\left\{-\frac{1}{\sigma_u^2}\left(\frac{1}{2}\boldsymbol{\tau}\boldsymbol{Q}\boldsymbol{\tau} + \beta\right)\right\} \end{aligned}$$

for a tri-diagonal matrix $\boldsymbol{Q}$ with diagonal elements equal to two except first and last element which are one, and the off-diagonal elements equal to -1. We recognize the posterior as the core of an inverse gamma IG(α^*, β^*) with shape $\alpha^* = \alpha + \frac{1}{2}(T-1)$ and scale $\beta^* = \beta + \frac{1}{2}\boldsymbol{\tau}\boldsymbol{Q}\boldsymbol{\tau}$.

d) Acceptance probability

Let $\mathcal{I} \subseteq \{1, 2, ..., 366\}$ be a set of time indices, and let $-\mathcal{I} = \{1, 2, ..., 366\} \setminus \mathcal{I}$. Furthermore, let $\boldsymbol{\tau}'$ denote the proposed values for $\boldsymbol{\tau}$. The MH step needs an acceptance probability denoted α for the proposed values $\boldsymbol{\tau}'_{\mathcal{I}}$. By using iterative conditioning we can write the acceptance probability as

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

where our prior proposal distribution is $Q(\boldsymbol{\tau}'_{\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}}, \sigma_u^2, \boldsymbol{y}) = P(\boldsymbol{\tau}'_{\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}}, \sigma_u^2)$. By considering

$$\begin{aligned}
P(\boldsymbol{\tau}'_{\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}}, \sigma_u^2, \boldsymbol{y}) &= \frac{P(\boldsymbol{\tau}'_{\mathcal{I}}, \boldsymbol{\tau}_{-\mathcal{I}}, \sigma_u^2, \boldsymbol{y})}{P(\boldsymbol{\tau}_{-\mathcal{I}}, \sigma_u^2, \boldsymbol{y})} \\
&= \frac{P(\boldsymbol{y}|\boldsymbol{\tau}'_{\mathcal{I}}, \boldsymbol{\tau}_{-\mathcal{I}}, \sigma_u^2) P(\boldsymbol{\tau}'_{\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}}, \sigma_u^2) P(\boldsymbol{\tau}_{-\mathcal{I}}, \sigma_u^2)}{P(\boldsymbol{y}|\boldsymbol{\tau}_{-\mathcal{I}}, \sigma_u^2) P(\boldsymbol{\tau}_{-\mathcal{I}}, \sigma_u^2)} \\
&= \frac{\overbrace{P(\boldsymbol{y}|\boldsymbol{\tau}'_{\mathcal{I}}, \boldsymbol{\tau}_{-\mathcal{I}})}^{\text{Conditionally independent}} \quad P(\boldsymbol{\tau}'_{\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}}, \sigma_u^2)}{P(\boldsymbol{y}|\boldsymbol{\tau}_{-\mathcal{I}}, \sigma_u^2)} \\
&= \frac{P(\boldsymbol{y}_{\mathcal{I}}|\boldsymbol{\tau}'_{\mathcal{I}}) P(\boldsymbol{y}_{-\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}}) P(\boldsymbol{\tau}'_{\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}}, \sigma_u^2)}{P(\boldsymbol{y}|\boldsymbol{\tau}_{-\mathcal{I}}, \sigma_u^2)}
\end{aligned}$$

and equally

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

we can rewrite the acceptance probability as

$$\begin{aligned}
\alpha(\boldsymbol{\tau}_{\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}}, \sigma_u^2, \boldsymbol{y}) &= \min\left(1, \frac{P(\boldsymbol{y}_{\mathcal{I}}|\boldsymbol{\tau}'_{\mathcal{I}}) P(\boldsymbol{y}_{-\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}}) P(\boldsymbol{\tau}'_{\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}}, \sigma_u^2) / P(\boldsymbol{y}|\boldsymbol{\tau}_{-\mathcal{I}}, \sigma_u^2)}{P(\boldsymbol{y}_{\mathcal{I}}|\boldsymbol{\tau}_{\mathcal{I}}) P(\boldsymbol{y}_{-\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}}) P(\boldsymbol{\tau}_{\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}}, \sigma_u^2) / P(\boldsymbol{y}|\boldsymbol{\tau}_{-\mathcal{I}}, \sigma_u^2)} \frac{P(\boldsymbol{\tau}_{\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}}, \sigma_u^2)}{P(\boldsymbol{\tau}'_{\mathcal{I}}|\boldsymbol{\tau}_{-\mathcal{I}}, \sigma_u^2)}\right) \\
&= \min\left(1, \frac{P(\boldsymbol{y}_{\mathcal{I}}|\boldsymbol{\tau}'_{\mathcal{I}})}{P(\boldsymbol{y}_{\mathcal{I}}|\boldsymbol{\tau}_{\mathcal{I}})}\right),
\end{aligned}$$

which is the minimum of 1 and the ratio of likelihoods conditioned on the proposed values and the old values.

e) Individual implementation

In this section we implement an MCMC sampler for the posterior $P(\boldsymbol{\pi}, \sigma_u^2|\boldsymbol{y})$. For the conditional prior, $P(\tau_t|\boldsymbol{\tau}_{-t}, \sigma_u)$, we use MH steps sequentially, and for σ_u we use Gibbs steps. The MH steps acceptance probability for updating individual τ_t parameters can be rewritten to improve computation time. In section 1.d) we found that the acceptance probability is a ratio of likelihoods, which in the individual update case is given by

$$\begin{aligned}
\frac{P(y_t|\tau'_t)}{P(y_t|\tau_t)} &= \frac{\pi(\tau'_t)^{y_t}(1 - \pi(\tau'_t))^{n_t - y_t}}{\pi(\tau_t)^{y_t}(1 - \pi(\tau_t))^{n_t - y_t}} \\
&= \frac{\exp\{y_t \tau'_t - n_t \ln(1 + e^{\tau'})\}}{\exp\{y_t \tau_t - n_t \ln(1 + e^{\tau})\}} \\
&= \exp\left\{y_t(\tau'_t - \tau_t) + n_t \ln\left(\frac{1 + e^{\tau}}{1 + e^{\tau'}}\right)\right\}, \qquad (2)
\end{aligned}$$

for which we chose to use the logarithm seen in the function `logAccRatio`. Thus, for our random walk we accept the proposed value if $\ln(r)$ is smaller than the logarithm of the ratio of likelihoods shown in Equation

(2). The $\tau_{\mathcal{I}}$ parameters for day $\mathcal{I} = t$ and days $-\mathcal{I}$ are multivariate Gaussian,

$$\begin{pmatrix} \tau_t \\ \boldsymbol{\tau}_{-t} \end{pmatrix} \sim MVN\left(\begin{pmatrix} \mu_t \\ \boldsymbol{\mu}_{-t} \end{pmatrix}, \begin{pmatrix} Q_{t,t} & \mathbf{Q}_{t,-t} \\ \mathbf{Q}_{-t,t} & \mathbf{Q}_{-t,-t} \end{pmatrix}^{-1}\right).$$

The conditional mean and precision used to draw the proposed value for τ_t are given by

$$\mu_{t|-t} = \mu_t - Q_{t,t}^{-1}\boldsymbol{Q}_{t,-t}\left(\boldsymbol{\tau}_{-t} - \boldsymbol{\mu}_{-t}\right) = -Q_{t,t}^{-1}\boldsymbol{Q}_{t,-t}\boldsymbol{\tau}_{-t}$$
$$Q_{t|-t} = Q_{t,t}.$$

For the implementation we use the given values $\alpha = 2$ and $\beta = 0.05$ for the response given in Equation (1), and we set the initial value of σ_u^2 to be 0.05. Initial values for τ are drawn from a standard normal distribution. We run the MCMC sampler for a total of $N = 50000$ iterations. In the following chunk are the functions used for the individually updating sampler followed by the chunk for which we run the sampler.

```
link = function(tau){
  # Expit link
  return(exp(tau)/(1+exp(tau)))
}

logbin = function(n, y, tau){
  # Remake and use this
  return(y*log(1+exp(-tau)) - (n-y)*log(1+exp(tau)))
}

logAccRatio = function(n, y, tauProp, tau) {
  return(y*(tauProp - tau) + n* log((1+exp(tau))/(1+exp(tauProp))))
}

mhFirst = function(tau, sigma, yt, t, norm_it, unif){
  # Function to take the first MH step, i.e., for t=1
  mu_ab = tau[2]
  prop_tau = norm_it*sigma + mu_ab
  n = 39
  # ratio = acceptRatio(n, yt, prop_tau, tau[t])
  ratio = logAccRatio(39, yt, prop_tau, tau[t])
  # if (runif(1) < min(c(1,ratio))){
  if (unif < min(c(1,ratio))){
    return(list(tau=prop_tau, accepted=1))
  }
  else{return(list(tau=tau[t], accepted=0))}
}

mhLast = function(tau, sigma, yt, t, norm_it, unif){
  # Function to take the last MH step, i.e., for t=366
  mu_ab = tau[365]
  prop_tau = norm_it*sigma + mu_ab
  # ratio = acceptRatio(n, yt, prop_tau, tau[t])
  ratio = logAccRatio(39, yt, prop_tau, tau[t])
  if (unif < min(c(1,ratio))){
    return(list(tau=prop_tau, accepted=1))
  }
  else{return(list(tau=tau[t], accepted=0))}
}
```

```
mcmcIndivid = function(N, dt, sigma0=0.05){
  # Allocate memory
  Ttot = 366
  tau = matrix(NA, nrow=N, ncol = Ttot)
  sigma = numeric(length = N)
  tau_i = numeric(length = Ttot)
  normMat = matrix(rep(rnorm(Ttot), N), nrow = N, ncol=Ttot)
  # unifMat = matrix(rep(runif(Ttot), N), nrow=N, ncol=Ttot)
  unifMat = matrix(rep(log(runif(Ttot)), N), nrow=N, ncol=Ttot) # log uniform

  # Find init vals
  tau[1,] = rnorm(Ttot) # init tau drawn from normal distr.
  sigma[1] = sigma0

  # Run mcmc for N iterations
  accepted = 0
  for (i in 2:N){
    tau_i = tau[i-1,]
    sigma_i = sqrt(sigma[i-1])

    # Take first MH step for t=1
    mhStep = mhFirst(tau_i, sigma_i, n.rain[1], 1, normMat[i,1], unifMat[i,1])
    tau_i[1] = mhStep$tau
    accepted = accepted + mhStep$accepted

    # Perform MH steps for 1<t<366
    for (t in 2:(Ttot-1)){
      mu_ab    = 1/2 * (tau_i[t-1] + tau_i[t+1])
      prop_tau = normMat[i,t]*sigma_i/2 + mu_ab
      ratio    = logAccRatio(n.years[t], n.rain[t], prop_tau, tau_i[t])
      if (unifMat[i,t] < min(c(1,ratio))){
        tau_i[t] = prop_tau
        accepted = accepted + 1
      }
      # else{tau[i,t] = tau_i[t]}
    }

    # Take last MH step for t=366
    mhStep = mhLast(tau_i, sigma_i, dt$n.rain[366], 1, normMat[i,366], unifMat[i,366])
    tau_i[366] = mhStep$tau
    tau[i,] = tau_i
    accepted = accepted + mhStep$accepted

    # Squared diff. of tau vec.
    # tQt = sum((tau[i,-Ttot] - tau[i,-1])^2) # this sim tau vals.
    tQt = sum(diff(tau[i,])^2) # this sim tau vals.

    # Gibbs step (Draw from IG)
    sigma[i] = 1/rgamma(1, shape=2 + (Ttot-1)/2, scale=0.05 + 0.5*tQt) # Gibbs inline

    setTxtProgressBar(pb,i)
  }
  close(pb)
```

```
  return(list(tau=tau, sigma=sigma, accProb = accepted/(N*Ttot)))
}
```

```
a          = 0
Q          = triDiag(diagonal = 2, upper = -1, lower = -1, nrow = 366)
Q[1,1]     = 1
Q[366,366] = 1

set.seed(321)
N       = 50000
pb      = txtProgressBar(min = 0, max = N, initial = 0)
ptm     = proc.time()
results = mcmcIndivid(N, rain)
```

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

```
time    = proc.time() - ptm
time
```

```
##    user  system elapsed
##   47.28    0.12   47.55
```

In the printout we see that computation time was 47.55. The following chunk contain functions that compute some sought values and make desired figures.

```
CImean = function(p){
  # Computes mean and upper,lower quantiles of vector.
  c(mean=mean(p), quantile(p, probs = c(0.025,0.975)))
}

plotTAH = function(p, hcol = "cyan3", xlab = "", ylab = "", burn=3000){
  # Plots trace, autocorr and hist of probs
  l = quantile(p, probs = 0.025)
  u = quantile(p, probs = 0.975)
  plot(p, type="l", xlab = "Iterations", ylab="Probability")
  abline(h=mean(p), col="red")
  abline(v=burn, lty=2, col="gray")
  acf(p, main="")
  hist(p, nclass=200, prob=T, main="", xlab="Probability", xlim=c(l-0.01,u+0.01))
  abline(v = l, col=hcol)
  abline(v = u, col=hcol)
}
pSeq = apply(results$tau, 2, link)
```

```
par(mfrow=c(3,3))
plotTAH(pSeq[,1])
plotTAH(pSeq[,201])
plotTAH(pSeq[,366])
```

The vertical red line in the trace plots to the left in Figure 2 show the mean of $\pi(\tau_t)$ over all 50000 iterations, for $t = 1, 201, 366$. The horizontal blue line is at the 3000^{nd} iteration. The vertical blue lines in the histogram

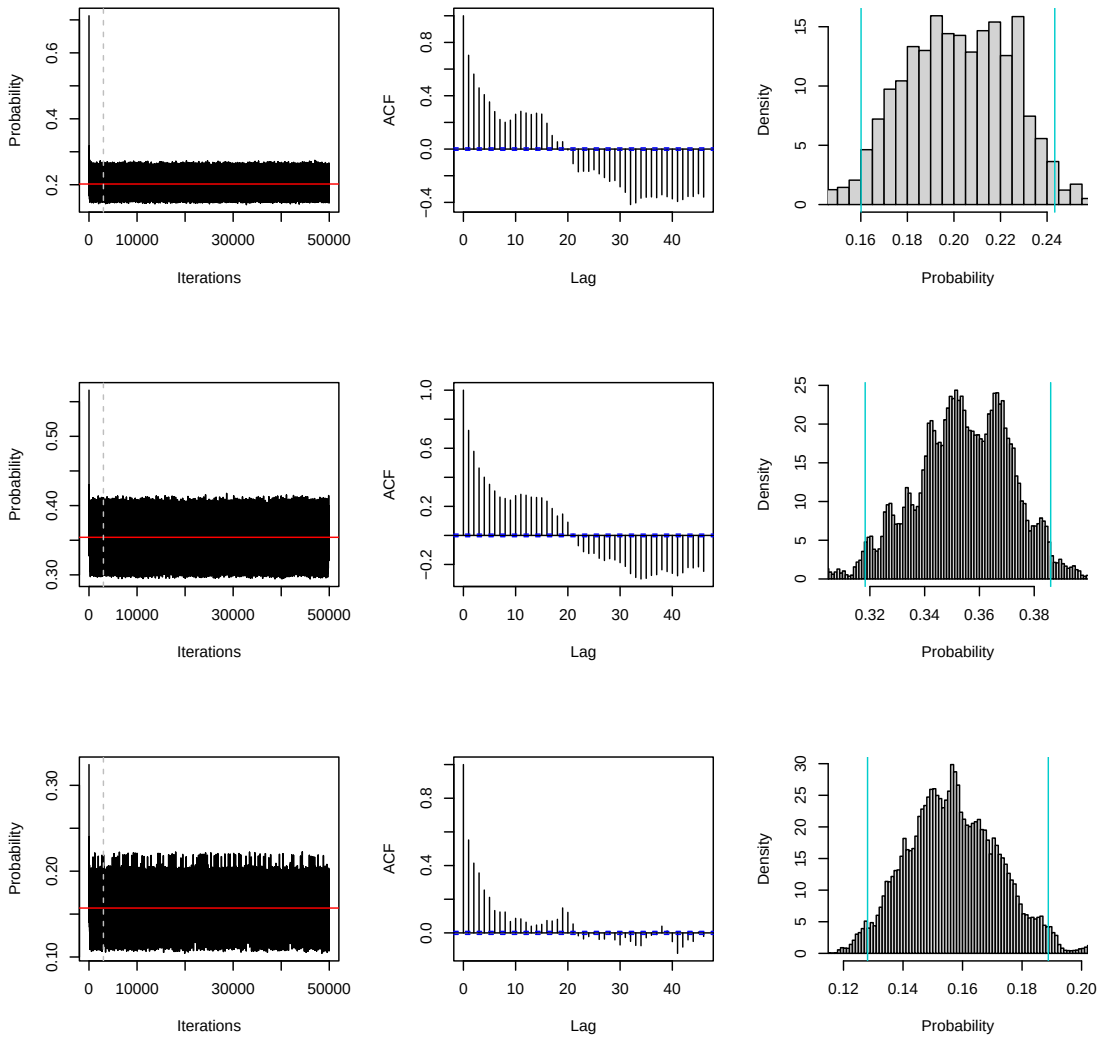


Figure 2: Traceplot, autocorrelation and histogram plots for $\pi(\tau_1), \pi(\tau_{201})$ and $\pi(\tau_{366})$ from top to bottom.

Table 1: Mean and credibility intervals for π_t, $t \in \{1, 201, 366\}$.

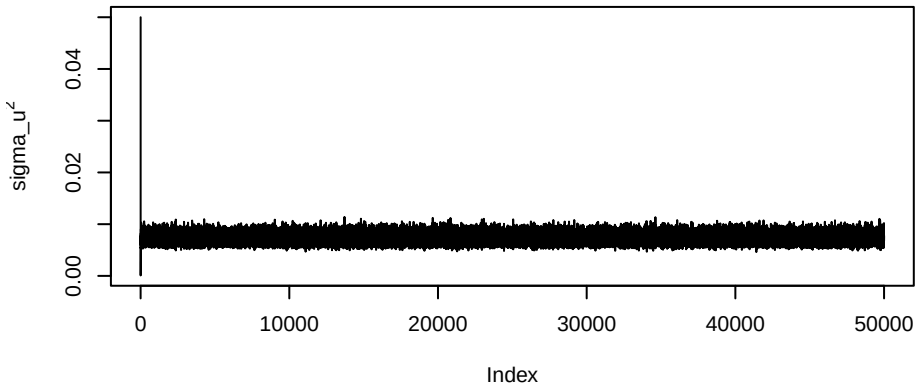
	mean	2.5%	97.5%
1	0.2022	0.1601	0.2433
1 B	0.2022	0.1601	0.2432
201	0.3542	0.3183	0.3860
201 B	0.3542	0.3183	0.3859
366	0.1570	0.1281	0.1888
366 B	0.1570	0.1281	0.1886
sigma	0.0074	0.0057	0.0093
sigma B	0.0074	0.0057	0.0093

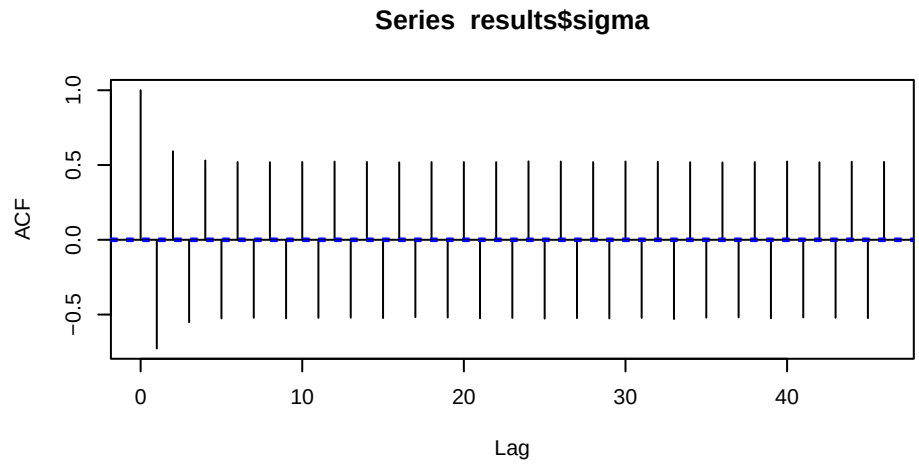
plot show the 95% credible intervals for $\pi(\tau_t)$. The traceplot bares resemblance to that of a random walk for all τ_t after approximately the 3000^{nd} iteration. We notice a decrease in autocorrelation and the histogram show that $\pi(\tau_t)$ seem normally distributed.

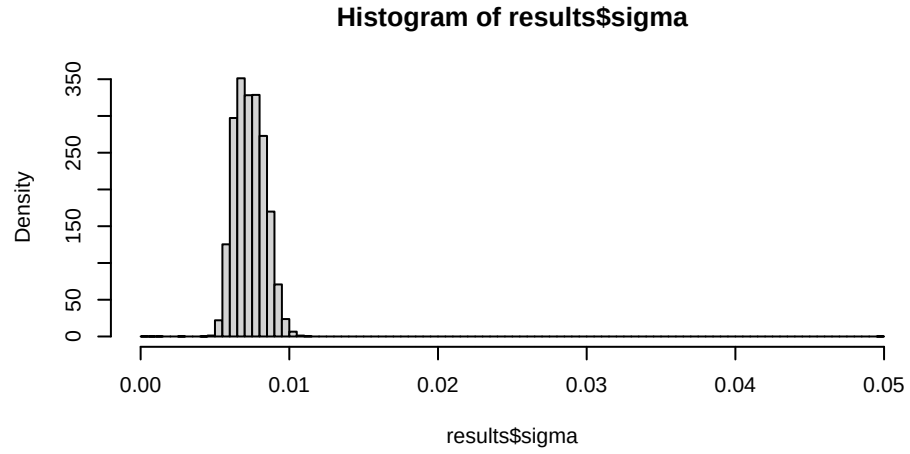
```
pSeqCI = apply(pSeq, 2, CImean)  # Compute mean and credibility interval

# Burn
burn = 3000
resultsB = results
resultsB$tau = results$tau[burn:N, ]
resultsB$sigma = results$sigma[burn:N]
pSeqB = pSeq[burn:N, ]
pSeqCIB = apply(pSeqB, 2, CImean)  # Compute mean and CI
ciAndMean = rbind((pSeqCI[, 1]), (pSeqCIB[, 1]), (pSeqCI[, 201]), (pSeqCIB[, 201]),
    (pSeqCI[, 366]), (pSeqCIB[, 366]), CImean(results$sigma), CImean(resultsB$sigma))
rownames(ciAndMean) = c("1", "1 B", "201", "201 B", "366", "366 B", "sigma", "sigma B")

knitr::kable(round(ciAndMean, 4), caption = "Mean and credibility intervals for $\\pi_t$, $t\\in\\{1,20
```

Burning some of the early iterations, here the first 3000, has some impact, but nothing major due to the fast convergence of MCMC sampler, as we can see in Table 1.

```
par(mfrow=c(3,1))
yl = expression(sigma_u^2)
plot(results$sigma, type="l", ylab=yl)
acf(results$sigma)
hist(results$sigma, nclass=100, prob=T)
```








```
ylim = c(min(rain$n.rain/rain$n.years), max=max(pSeq[1,]))
par(mfrow=c(1,1))
plot(rain$day,pSeq[1,], type = "l",
     ylab=expression(pi), xlab="Day",
```

```
     ylim = ylim, col="gray")
lines(rain$day,pSeq[N/2,], type = "l",
     ylim = ylim, col="cyan3")
lines(rain$day,pSeq[N,],
     ylim = ylim, col = "blue", lty=1)
legend(x="topright", legend = c("Initial values", "Halfway", "Last iteration"), lty = c(1,1,1),
       col=c("gray", "cyan3", "blue"), bg="transparent")
```

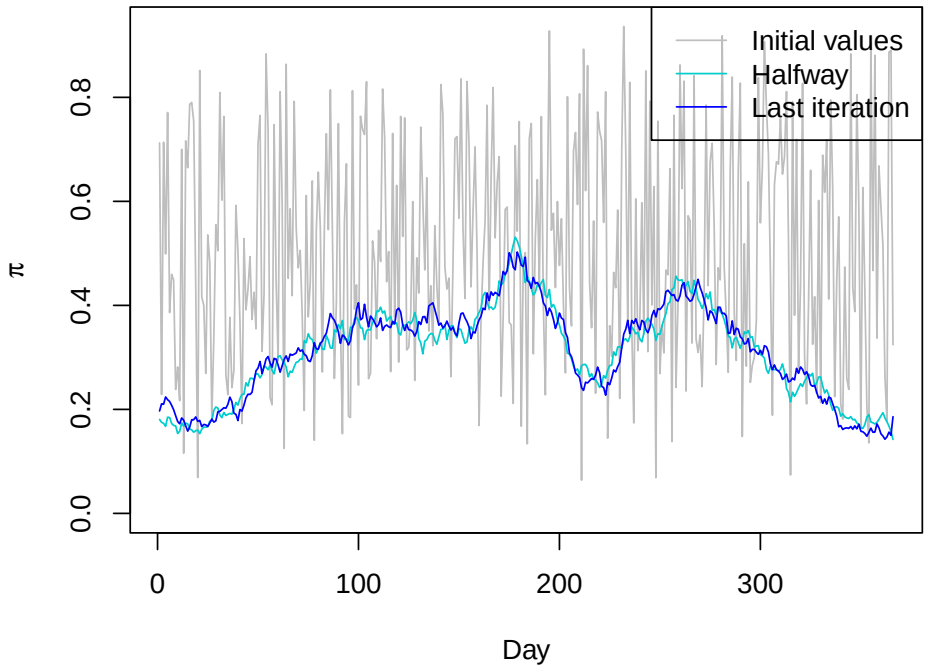


Figure 3: Make cap

The initial values of $\boldsymbol{\pi}$ show no resemblance to the Tokyo rainfall dataset probabilities in Figure 3

```
gg1e.df = data.frame(probPreds = pSeq[N,], l = pSeqCI[2, ], u = pSeqCI[3,],
                     meanPreds = pSeqCI[1,], probsReal = n.rain/n.years,
                     day = day)
ggplot(data = gg1e.df, aes(x = day, y = probPreds)) +
  # geom_ribbon(mapping = aes(ymin = l,ymax = u, fill = "CI"), alpha = 0.2) +
  geom_line(aes(color = "tau", linetype="tau")) +
  geom_line(aes(y = meanPreds, color = "mean", linetype="mean")) +
  geom_point(aes(y = probsReal, color = "n.rain/n.years",
                 size="n.rain/n.years"), alpha = 0.3) +
  scale_linetype_manual(
    name=" ",
    breaks = c("tau", "mean", "n.rain/n.years"),
    values = c(2, 1, NA),
    labels=expression(pi*(tau),'Mean','Data probability')
```

```
  # breaks = c("tau", "mean"),
  # values = c(2, 1),
  # labels=expression(pi*(tau),'Mean')
  # values = c("tau"=1, "mean"=2, "n.rain/n.years"=2)
) +
scale_size_manual(
  name=" ",
  breaks = c("tau", "mean", "n.rain/n.years"),
  values = c(NA, NA, 0.1),
  labels=expression(pi*(tau),'Mean','Data probability')
  ) +
scale_color_manual(
  name=" ",
  # breaks = c("tau", "mean", "n.rain/n.years"),
  values=c("tau"="green", "mean"="black","n.rain/n.years"="blue"),
  # values=c("black", "green","pink"),
  labels=expression(pi*(tau),'Mean','Data probability')
) +
labs(y="Probabilities", x="Day") +
theme_minimal()
```

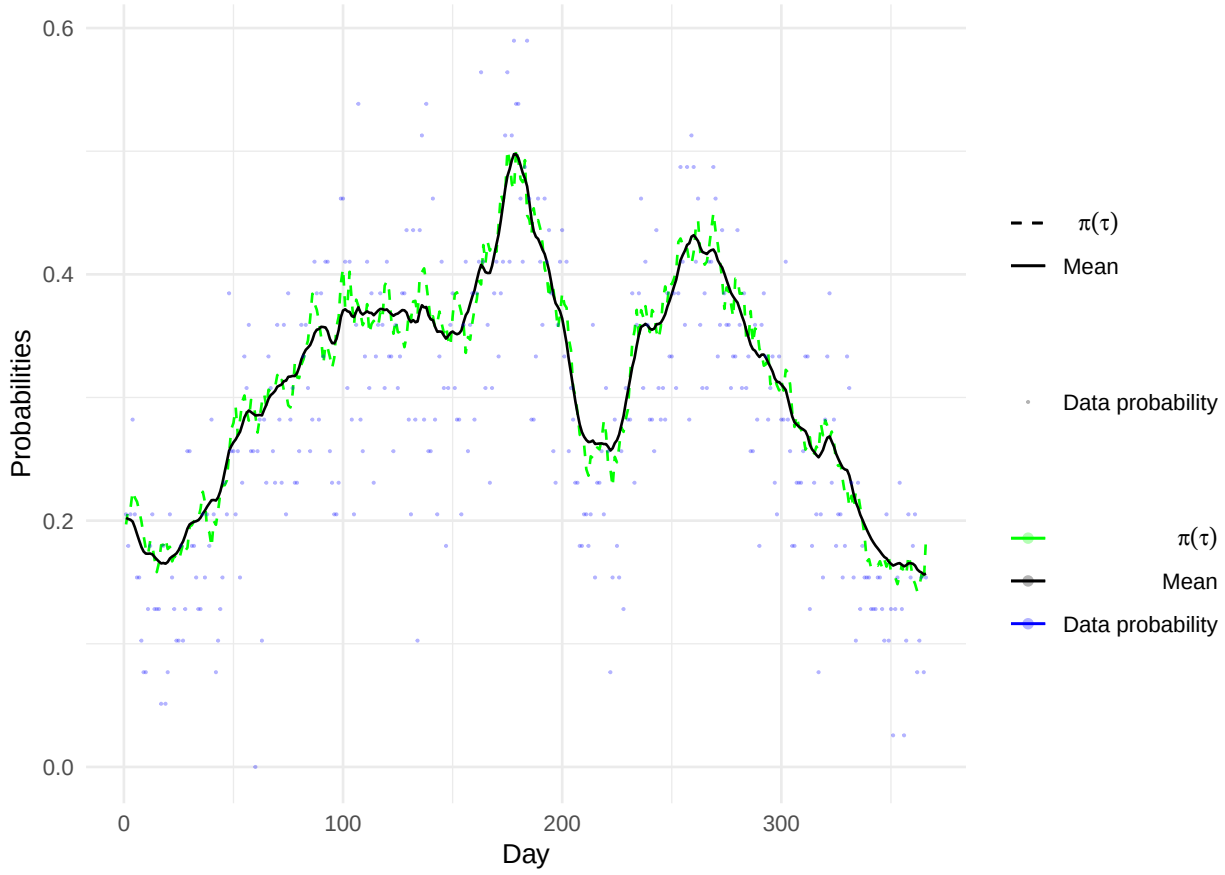


Figure 4: The black line is the mean of all predicted probabilities π excluding the burned predictions. The dashed green and blue lines are last iterations MCMC samples and the data probabilities, respectively.

f) Blocking implementation

Now we will go from sequentially updating τ_t to perform block updates for each iteration. We will start by showing what mathematical implications that comes from blocking before we show the implementation and discuss the results. Rather than updating individual τ_t parameters we will use a conditional prior proposal involving $P(\boldsymbol{\tau}_{(a,b)}|\boldsymbol{\tau}_{-(a,b)})$, where $\boldsymbol{\tau}_{(a,b)} = (\tau_a, ..., \tau_b)$ for interval length $M = b - a$. For intervals of length $M > 1$ we can simplify the acceptance probability as we did in section , only now

$$\frac{P(\boldsymbol{y}_{\mathcal{I}}|\boldsymbol{\tau}'_{\mathcal{I}})}{P(\boldsymbol{y}_{\mathcal{I}}|\boldsymbol{\tau}_{\mathcal{I}})} = \exp\left\{\sum_{\mathcal{I}} y_i(\tau'_i - \tau_i) + n_i \ln\left(\frac{1+e^{\tau_i}}{1+e^{\tau'_i}}\right)\right\}, \tag{3}$$

where $\mathcal{I} = (a, b)$. The conditional mean and precision used to draw the proposed values $\boldsymbol{\tau}_{\mathcal{I}}$ are given by

$$\boldsymbol{\mu}_{\mathcal{I}|-\mathcal{I}} = \boldsymbol{\mu}_{\mathcal{I}} - \mathbf{Q}^{-1}_{\mathcal{I},\mathcal{I}}\mathbf{Q}_{\mathcal{I},-\mathcal{I}}\left(\boldsymbol{\tau}_{-\mathcal{I}} - \boldsymbol{\mu}_{-\mathcal{I}}\right) = -\mathbf{Q}^{-1}_{\mathcal{I},\mathcal{I}}\mathbf{Q}_{\mathcal{I},-\mathcal{I}}\boldsymbol{\tau}_{-\mathcal{I}}$$
$$\mathbf{Q}_{\mathcal{I}|-\mathcal{I}} = \mathbf{Q}_{\mathcal{I},\mathcal{I}}.$$

Assuming $1 < M < T - 2$, we get three different types of precision matrices which can be precomputated. Namely, one when the first day is included, the second where neither the first nor the last day is included, and the third where the last day is included in the set of days ⟩. There are also only three different precision matrices for one iteration for the same day sets mentioned which also can be precomputated for each iteration. precomputed or precomputated?. The shape and size of said deflectors of the $\mathbf{Q}$ matrix will not be discussed further, but can be computed from the `Qprecomp` function seen in the chunk below. The chunk also include the function for computing the acceptance ratio from Equation (3) and the implementation of the MCMC sampler.

```
acceptRatioBlock = function(I, tauProp, tauPrev){
  return(exp(sum(n.rain[I]*(tauProp - tauPrev)) +
              sum(rain$n.years[I]*(log(1+ exp(tauPrev)) -
                                     log(1+exp(tauProp)))
                  )
              )
         )
}

Qprecomp = function(M, Ttot=366){
  ## Make intervals I
  n.sets = ceiling(Ttot/M)
  I = matrix(1:(M*(n.sets-1)), ncol = n.sets-1, byrow=F)
  Ires = (I[M,n.sets-1]+1):Ttot

  # Make Q
  Q = triDiag(diagonal = 2, upper = -1, lower = -1, nrow = Ttot)
  Q[1,1] = 1
  Q[Ttot,Ttot] = 1
  # Find the three Q_AA
  Qaa1 = Q[1:M,1:M]
  Qaa2 = Q[2:(M+1), 2:(M+1)]
  Qaa3 = Q[Ires, Ires]
  Qaa1inv = solve(Qaa1)
  Qaa2inv = solve(Qaa2)
  Qaa3inv = solve(Qaa3)

  Qmult = list(Qmult1 = -Qaa1inv %*% Q[1:M, -(1:M)])
```

```
  for (i in 2:(n.sets-1)){
    Qmult = append(Qmult, list(-Qaa2inv %*% Q[I[,i], -I[,i]]))
  }
  Qmult = append(Qmult, list(-Qaa3inv %*% Q[Ires, -Ires]))
  names(Qmult) = sprintf("Qmult%d", 1:(n.sets))

  return(list(Qaa1inv = Qaa1inv, Qaa2inv = Qaa2inv, Qaa3inv = Qaa3inv,
              Qmult = Qmult,
              n.sets = n.sets, I = I, Ires = Ires))
}

mcmcBlock = function(N, M=10, sigma0=0.1, tau0=rnorm(366)){
  # Allocate memory
  tau      = matrix(NA, nrow = N, ncol = Ttot) # (N x T)
  sigma    = numeric(length = N)
  # Assign initial values
  tau[1,]  = tau0
  sigma[1] = sigma0
  # Assign Q matrices for easier access
  Qm       = Q$Qmult # precomputed -QaaInverse * Qab
  n.sets   = Q$n.sets
  Imat     = Q$I # (M x (n.sets-1))
  Ires     = Q$Ires

  # Simulate N times
  accepted = 0
  for (i in 2:N){
    # Perform first MH block step using upper left sub matrices of Q
    mhBlock = mhBlockStep(tau[i-1,], Imat[,1], sigma[i-1]*Q$Qaa1inv, Qm$Qmult1)
    tau[i, Imat[,1]] = mhBlock$tau
    accepted = accepted + mhBlock$accepted

    # Precompute sigma_u^2 * Q_AA^-1 for all blocks where 1<t<366
    Sigma.mid = sigma[i-1]*Q$Qaa2inv
    # Perform MH block steps for days 1<t<366
    for (t in 2:(n.sets-1)){
      mhBlock = mhBlockStep(tau[i-1,], Imat[,t], Sigma.mid, Qm[[t]])
      tau[i, Imat[,t]] = mhBlock$tau
      accepted = accepted + mhBlock$accepted
    }

    # Perform last MH block step using lower left sub matrices of Q
    mhBlock.last = mhBlockStep(tau[i-1,], Ires, sigma[i-1]*Q$Qaa3inv, Qm[[n.sets]])
    tau[i, Ires] = mhBlock.last$tau
    accepted = accepted + mhBlock.last$accepted

    # Perform Gibbs step for sigma_u^2
    tQt = sum((diff(tau[i,]))^2)
    sigma[i] = 1/rgamma(1, shape=2 + (Ttot-1)/2, scale=0.05 + 0.5*tQt)

    setTxtProgressBar(pb,i)
  }
  close(pb)
```

```
  results = list(tau, sigma, accepted/(N*Ttot))
  names(results) = c(paste0('tau',M), paste0('sigma',M), paste0('accProb',M))
  return(results)
}

mhBlockStep = function(tauPrev, I, Sigma, Qm.I){
  # Inputs
  # tauPrev : Previous sim tau values
  # I       : Indices for this block
  # Sigma   : sigma_u^2 * Q_AA^-1
  # Qm.I    : -Q_AA^-1 x Q_AB

  mu.ab = Qm.I %*% tauPrev[-I]
  tauProp = mvrnorm(n=1, mu=mu.ab, Sigma=Sigma)
  ratio = acceptRatioBlock(I, tauProp, tauPrev[I])
  # Random walk 1
  if (runif(1) < min(c(1,ratio))){
    return(list(tau=tauProp, accepted=length(I)))
  }
  else{return(list(tau=tauPrev[I], accepted=0))}
}

M = 10
N = 5000
Ttot = 366
```

```
N = 5000

Mlist = c(2, 5, 10, 15, 20, 25, 40, 80)
n.runs = length(Mlist)
resultList = list()
time = c()
set.seed(321)
tau0 = rnorm(366)

for (i in 1:length(Mlist)) {
    set.seed(321)
    pb = txtProgressBar(min = 0, max = N, initial = 0)
    Q = Qprecomp(Mlist[i], Ttot)
    start = proc.time()[3]
    resultList = append(resultList, mcmcBlock(N, Mlist[i], tau0 = tau0))
    time = c(time, proc.time()[3] - start)
}
```

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

The chunk above ran the MCMC sampler for $N = 5000$ iterations with interval lengths $\boldsymbol{M} =$

$\{2, 5, 10, 15, 20, 25, 40, 80\}$. In Figure 5 produced by the chunk below we see that the acceptance probability decrease with increasing interval lengths M. For a RW(1) model, a rule of thumb is to have acceptance probability in the range $(0.2, 0.5)$ which coincide with the acceptance probability for $M = \{15, 20, 25\}$. We also notice that the computation time decrease for interval lengths $M \leq 25$ before it starts to increase. Thus, based on these results, $M = 25$ seems like a good candidate for interval length.

```
accProbsBlock = c()
for (m in Mlist) {
    accProbsBlock = c(accProbsBlock, resultList[[paste0("accProb", m)]])
}

plot(Mlist, time, ylim = c(0, max(c(time, 100 * accProbsBlock))), pch = 16, xaxt = "n",
    xlab = "M", ylab = "Time/acceptance probability (%)")
points(Mlist, accProbsBlock * 100, col = "cyan3", pch = 16)
abline(h = c(20, 50), lty = 2)
axis(1, at = Mlist)
legend(x = "topright", legend = c("Computation time", "Acceptance probability"),
    pch = c(16, 16), col = c("black", "cyan3"))
```

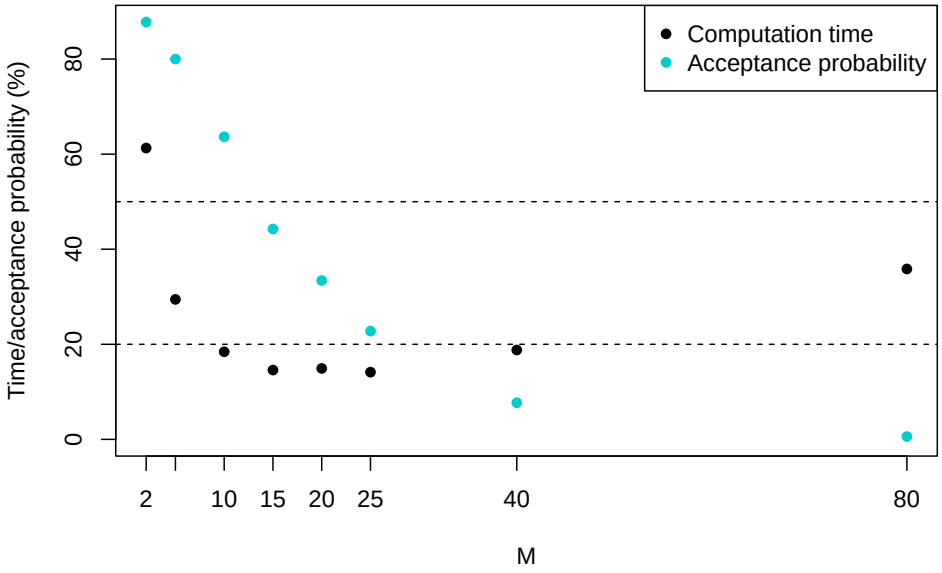


Figure 5: Computation time for different values of M along with the respective acceptance probability. Dashed horizontal lines show desired acceptance probability for RW(1).

To further investigate candidates for interval lengths we consider the traceplots in Figure 6 produced by the chunk below. Since all fits were run for only \$N= 5000 \$ iterations, it is difficult to say whether the chain has fully converged. The three plot rows in the bottom show clear signs of not convergence. Still, the resulting traceplots does not rule out our favorable candidate from before, namely $M = 25$.

```
dToPlot = c(1, 201, 366)
par(mfrow = c(length(Mlist), length(dToPlot)), mar = c(2, 2, 0.1, 1))
for (i in 1:length(Mlist)) {
    xaxt = ifelse(i == length(Mlist), "s", "n")
    # leg = ifelse(i%%3==1, expression(pi), '')
    j = 1
    for (t in dToPlot) {
        plot(resultList[[paste0("tau", Mlist[i])]][, t], type = "l", xlab = "", ylab = "",
            xaxt = xaxt)
        if (j == 1) {
            legend(x = "topright", legend = paste0("M = ", Mlist[i]))
        }
        j = j + 1
    }
}
```

So, some values of M show promicing results, but we still need to compare them to the real data. Since all traceplots in Figure 6 show signs of not convergence for early iterations we chose to burn some values. The amount burned is shown in the chunk below.

```
burnTune = 1000  # burn amount for M tuning
pMeanBlockMat = matrix(NA, nrow = length(Mlist), ncol = Ttot)
for (i in 1:length(Mlist)) {
    pMeanBlockMat[i, ] = apply(link(resultList[[paste0("tau", Mlist[i])]][burnTune:N,
        ]), 2, mean)
}
```

In Figure 7 produced by the chunk below we see that the mean probabilities $\boldsymbol{\pi}$ for our candidates $M = 15$ and 20 does not coincide well with the observed data. This may be because they have not fully converged. Thus, we chose $M = 25$ which show more promising results to perform the full $N = 50000$ iterations for comparison with what we found in section 1e).

```
colfunc <- colorRampPalette(c("red", "yellow", "springgreen", "royalblue"))
c = colfunc(length(pMeanBlockMat[, 1]))
diffTune = numeric()
pReal = n.rain/n.years
# plot(day,pReal, pch=4, cex=0.5)
par(mfrow = c(ceiling(length(Mlist)/2), 2), mar = c(1, 2, 1, 1))
for (i in 1:length(pMeanBlockMat[, 1])) {
    ifelse(i < length(pMeanBlockMat[, 1]) - 1, NA, par(mar = c(2, 2, 1, 1)))
    yaxt = ifelse(i%%2, "s", "n")
    xaxt = ifelse(i < length(pMeanBlockMat[, 1]) - 1, "n", "s")
    xl = paste0("M = ", Mlist[i])
    plot(day, pReal, pch = 16, cex = 0.5, col = "cyan3", ylab = expression(pi * xl),
        yaxt = yaxt, xaxt = xaxt)
    lines(day, pMeanBlockMat[i, ])
    legend(x = "topright", legend = paste0("M=", Mlist[i]))
    diffTune = cbind(diffTune, sum((pMeanBlockMat[i, ] - pReal)^2))
    # diffTune[str(Mlist[i])] = sum((pMeanBlockMat[i,] - pReal)^2)
}
```

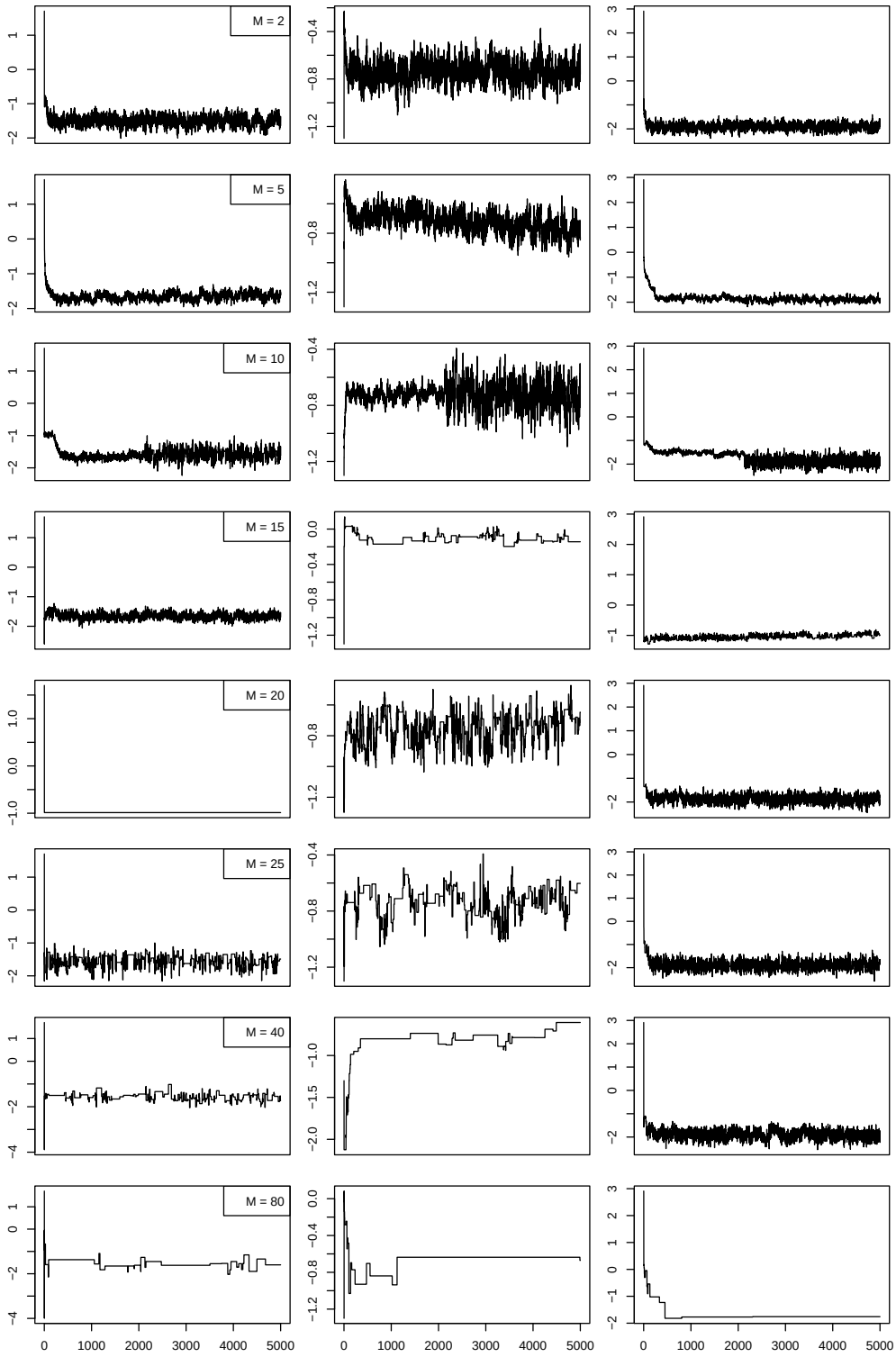


Figure 6: Trace plots of $\boldsymbol{\pi}$ for low to high value of the tuning param M from top to bottom, respectively, for different days t.

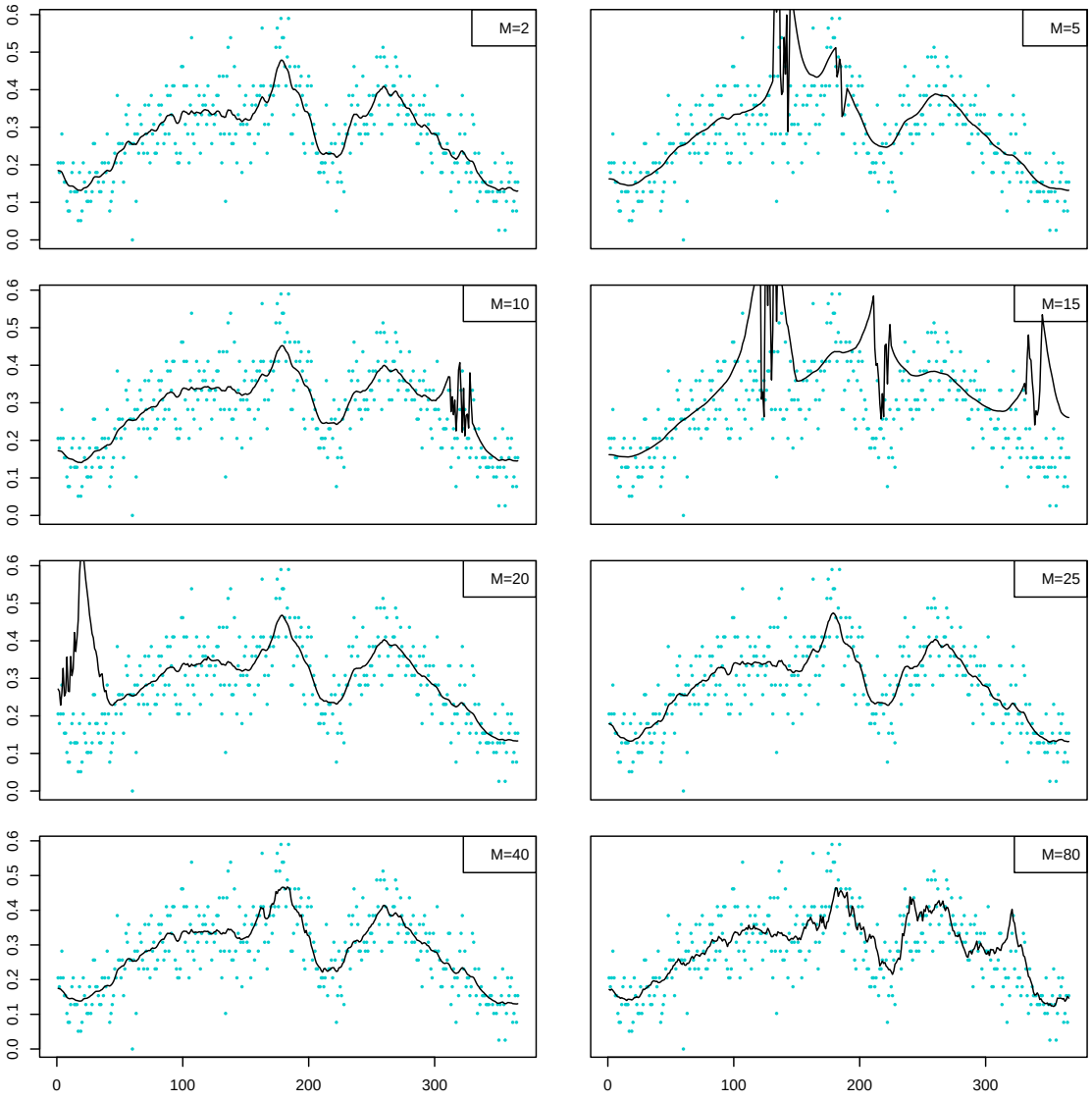


Figure 7: Mean probabilities from 5000 iterations (black line) along with observed data (points) for different interval lengths M.

```
# legend(x='topright', legend = Mlist, col = c, lty = rep(1,7))
# colnames(diffTune) = Mlist diffTune colnames(diffTune) = Mlist
```

```
N = 50000
M = 25
set.seed(321)
pb = txtProgressBar(min = 0, max = N, initial = 0)
Q = Qprecomp(M, Ttot)
start = proc.time()[3]
resultBlock = mcmcBlock(N, M)
```

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

```
time = c(time, proc.time()[3] - start)
```

In the following chunk we produce Figure 8 which show promising results for our choice of interval length. We see that some iterations should be burned which is done below.

```
pBlock = apply(resultBlock$tau, 2, link)
par(mfrow = c(3, 3))
plotTAH(pBlock[, 1])
plotTAH(pBlock[, 201])
plotTAH(pBlock[, 366])
```

```
pBlockCI = apply(pBlock, 2, CImean)  # Compute mean and credibility interval

# Burn
burn = 2000
resultsB = resultBlock
resultsB$tau = resultBlock$tau[burn:N, ]
resultsB$sigma = resultBlock$sigma[burn:N]
pBlockB = pBlock[burn:N, ]
pBlockCIB = apply(pSeqB, 2, CImean)  # Compute mean and credibility interval
ciAndMean = rbind((pBlockCI[, 1]), (pBlockCIB[, 1]), (pBlockCI[, 201]), (pBlockCIB[,
    201]), (pBlockCI[, 366]), (pBlockCIB[, 366]), CImean(resultBlock$sigma), CImean(resultsB$sigma))
rownames(ciAndMean) = c("1", "1 B", "201", "201 B", "366", "366 B", "sigma", "sigma B")
# knitr::kable(round(ciAndMean,4), caption = 'mean and stuff')
knitr::kable(round(ciAndMean, 5), caption = "Mean and credibility intervals for $\\pi_t$, $t\\in\\{1,20
```

In Table 2 we see marginal differences between burned and unburned results, which are also shown in Figure 9.

```
gg1f.df = data.frame(day = day, mp = pBlockCI[1, ], pr = pReal, l = pBlockCIB[2,
    ], u = pBlockCIB[3, ], mpB = pBlockCIB[1, ])
ggplot(gg1f.df, aes(x = day)) + geom_point(aes(y = pr, shape = "mr"), colour = "cyan3",
    size = 1) + geom_line(aes(y = mp, color = "mp")) + geom_line(aes(y = mpB, color = "mpB")) +
    scale_color_manual(name = "", values = c(mp = "black", mpB = "pink"), labels = c("Burned",
        "Unburned")) + scale_shape_manual(name = " ", values = 16, labels = "Observations") +
    labs(y = "Probabilities", x = "Day") + theme_minimal()
```

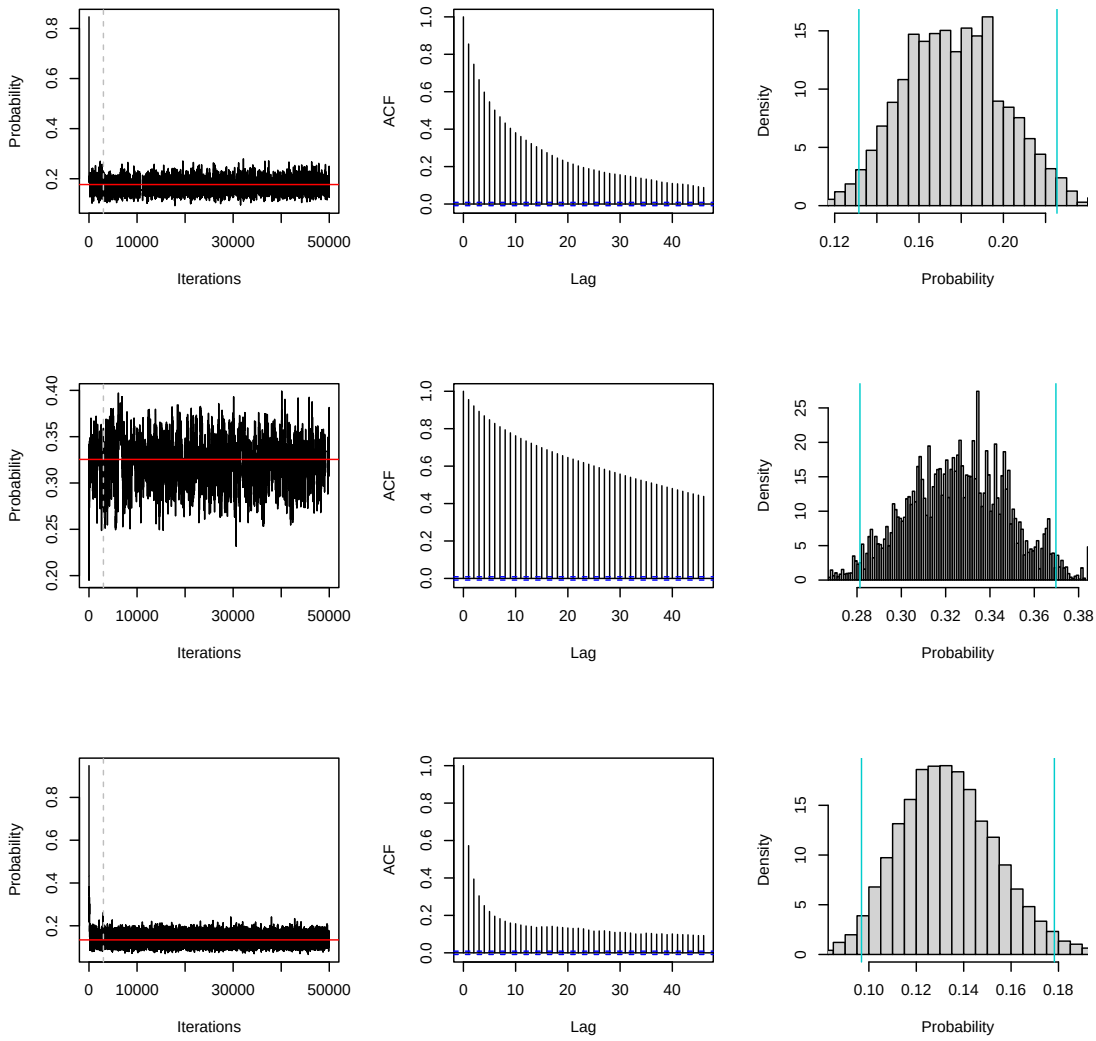


Figure 8: Traceplot, autocorrelation and histogram plots for $\pi(\tau_1), \pi(\tau_{201})$ and $\pi(\tau_{366})$ from top to bottom for block updated τ.

Table 2: Mean and credibility intervals for π_t, $t \in \{1, 201, 366\}$. Burned results are marked with B

	mean	2.5%	97.5%
1	0.17687	0.13148	0.22540
1 B	0.20215	0.16010	0.24318
201	0.32545	0.28133	0.36976
201 B	0.35419	0.31825	0.38592
366	0.13401	0.09688	0.17827
366 B	0.15696	0.12808	0.18864
sigma	0.00528	0.00439	0.00632
sigma B	0.00528	0.00439	0.00632

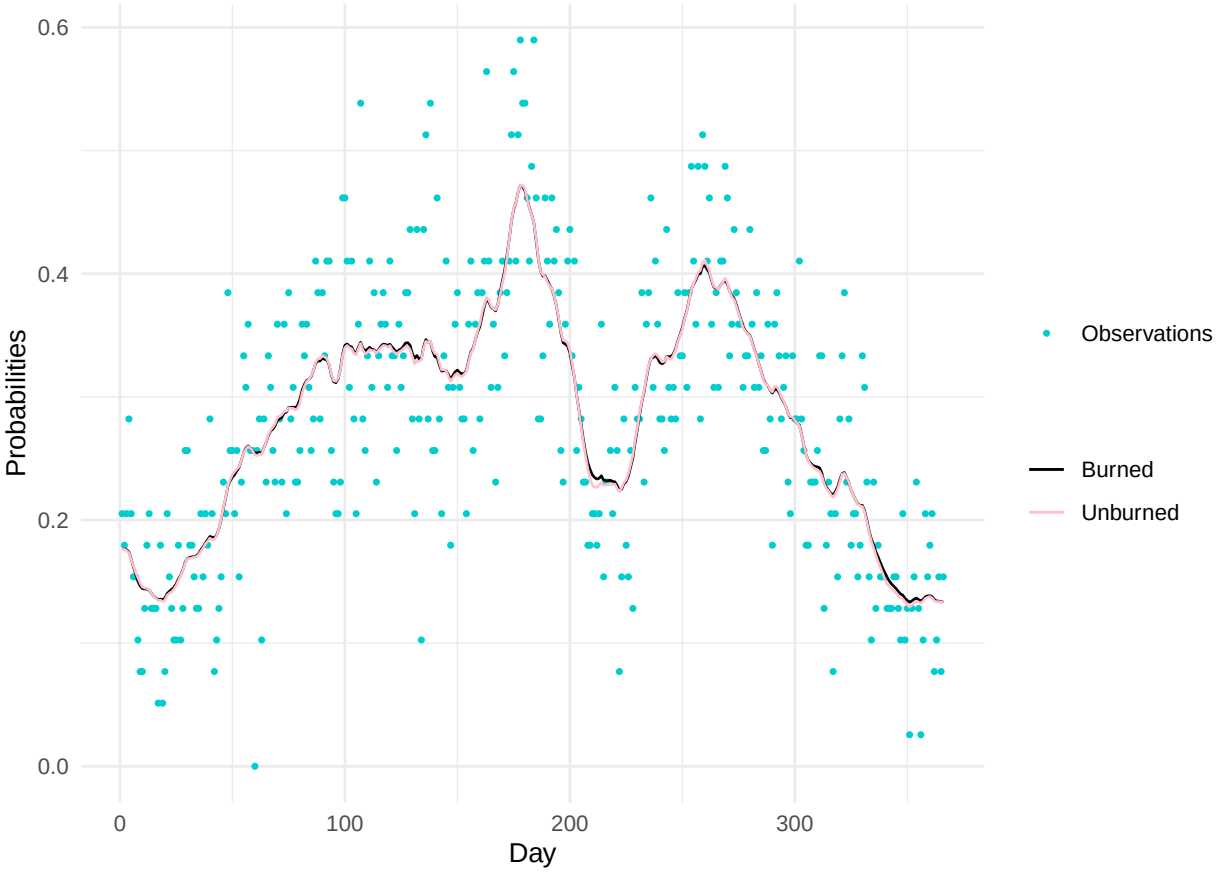


Figure 9: The black line is the mean of all predicted probabilities π excluding the burned predictions. The dashed green and blue lines are last iterations MCMC samples and the data probabilities, respectively.

```
library("INLA")
library("ggplot2")
```

Problem 2

In this problem we will use INLA to fit the same model as in the previous problem.

a) INLA model 1

To be able to compare the model with the previous Markov chain, we need to use the same priors as in Problem 1. In this model the intercept term is removed so we don't include a prior on this term. We do however need to include a prior for σ_u^2. We have

$$\tau_t \sim \tau_{t-1} + u_t$$

for $u_t \sim N(0, \sigma_u^2)$. In INLA priors are placed on the log precision rather than the variance. The precision is $1/\sigma_u^2$, and we place the prior on the hyperparameter

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

We know from problem 1 that σ_u^2 has a inverse Gamma distribution. This means that $\sigma_u^2 \sim \text{Gamma}(\alpha, \beta)$ and $\theta \sim \text{loggamma}(\alpha, \beta)$.

We therefore place this prior on θ and use $\alpha = 2$ and $\beta = 0.05$.

```
# Prior placed on hyperparameter
alpha = 2
beta = 0.05
hyper = list(prec = list(prior = "loggamma", param = c(alpha, beta)))
```

In the chunk below a function that fits a INLA model for given control.inla inputs is made. The function also returns the computation time using " proc.time()[3].

```
INLA_fit <- function(con.inla) {
    time_before = proc.time()[3]
    mod <- inla(n.rain ~ -1 + f(day, model = "rw1", constr = FALSE, hyper = hyper),
        data = rain, Ntrials = n.years, control.compute = list(config = TRUE), family = "binomial",
        verbose = TRUE, control.inla = con.inla)
    # computation time
    time = proc.time()[3] - time_before
    return(list(mod = mod, time = time))
}
```

We also make a function that plots the development of the means of the model with a 95 % credible interval.

```
INLA_plot <- function(mod) {
    lower = mod$summary.fitted.values$`0.025quant`
    # upper bound
    upper = mod$summary.fitted.values$`0.975quant`
    # Plot means with 95 % credible interval
```

```
    plot <- ggplot(data = as.data.frame(mod$summary.fitted.values$mean), mapping = aes(x = 1:366,
        y = mod$summary.fitted.values$mean)) + geom_line(col = "red") + geom_ribbon(aes(ymin = lower,
        ymax = upper), alpha = 0.1) + xlab("Day") + ylab("predicted values")
    return(plot)
}
```

The function below plots the predictions of π from two models together.

```
INLA_plot_2 <- function(mod1, mod2) {
    label_mod1 = paste("Predictions from ", deparse(substitute(mod1)))
    label_mod2 = paste("Predictions from ", deparse(substitute(mod2)))
    ggplot() + geom_line(data = as.data.frame(mod1$summary.fitted.values$mean), mapping = aes(x = 1:366
        y = mod1$summary.fitted.values$mean, color = label_mod1)) + geom_line(data = as.data.frame(mod2
        mapping = aes(x = 1:366, y = mod2$summary.fitted.values$mean, color = label_mod2),
        linetype = 2) + labs(color = "Models") + xlab("Day") + ylab("Predicted values of " ~
        pi)
}
```

We fit a model using simplified Laplace as the strategy for approximations and ccd as the strategy for integration.

```
# control.inla input
control.inla = list(strategy = "simplified.laplace", int.strategy = "ccd")
fit1 <- INLA_fit(control.inla)
mod1 <- fit1$mod
time1 <- fit1$time
```

We look at predictions and uncertainties of INLA and plot the development of the means with a 95 % credible interval for the fitted values.

```
INLA_plot(mod1)
```

We look specifically at $\pi(\tau_1), \pi(\tau_{201})$ and $\pi(\tau_{366})$.

```
# pi(tau_1)
mod1$summary.fitted.values[1, ]
```

```
##                            mean         sd 0.025quant  0.5quant 0.975quant
## fitted.Predictor.001 0.1857484 0.02889558  0.1336346 0.1841876  0.2467132
##                            mode
## fitted.Predictor.001 0.1810559
```

```
mod1$summary.fitted.values[201, ]
```

```
##                            mean         sd 0.025quant  0.5quant 0.975quant
## fitted.Predictor.201 0.3329528 0.02871471  0.2785434 0.3322901  0.3911262
##                            mode
## fitted.Predictor.201 0.3309649
```

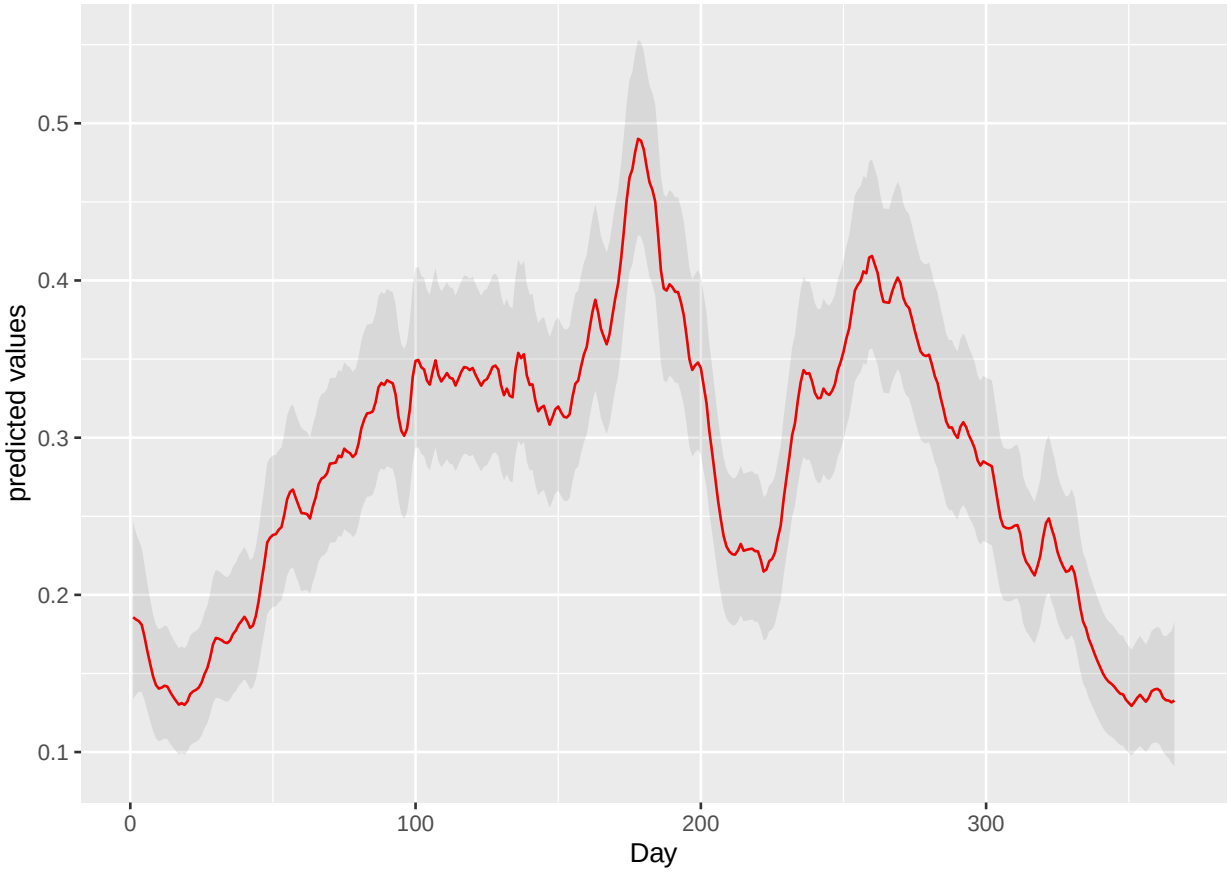


Figure 10: Plot of mean of fitted π-values of model with credible interval

```
mod1$summary.fitted.values[366, ]
```

```
##                            mean         sd 0.025quant  0.5quant 0.975quant
## fitted.Predictor.366 0.1328166 0.02346631 0.09103594 0.1313654  0.1828444
##                            mode
## fitted.Predictor.366 0.1284729
```

By looking at the results from the chunk above and compare it to table 3, we can see that we get similar results to problem 1 which is as expected. We also plot the mean values from problem 1e together with the predcited mean values from INLA.

```
ggplot() + geom_line(data = as.data.frame(mod1$summary.fitted.values$mean), mapping = aes(x = 1:366,
    y = mod1$summary.fitted.values$mean, color = "Fitted values from INLA")) + geom_line(data = gg1e.df
    aes(x = 1:366, y = meanPreds, color = "Fitted values from 1e")) + xlab("Day") +
    ylab("Mean of " ~ pi)
```

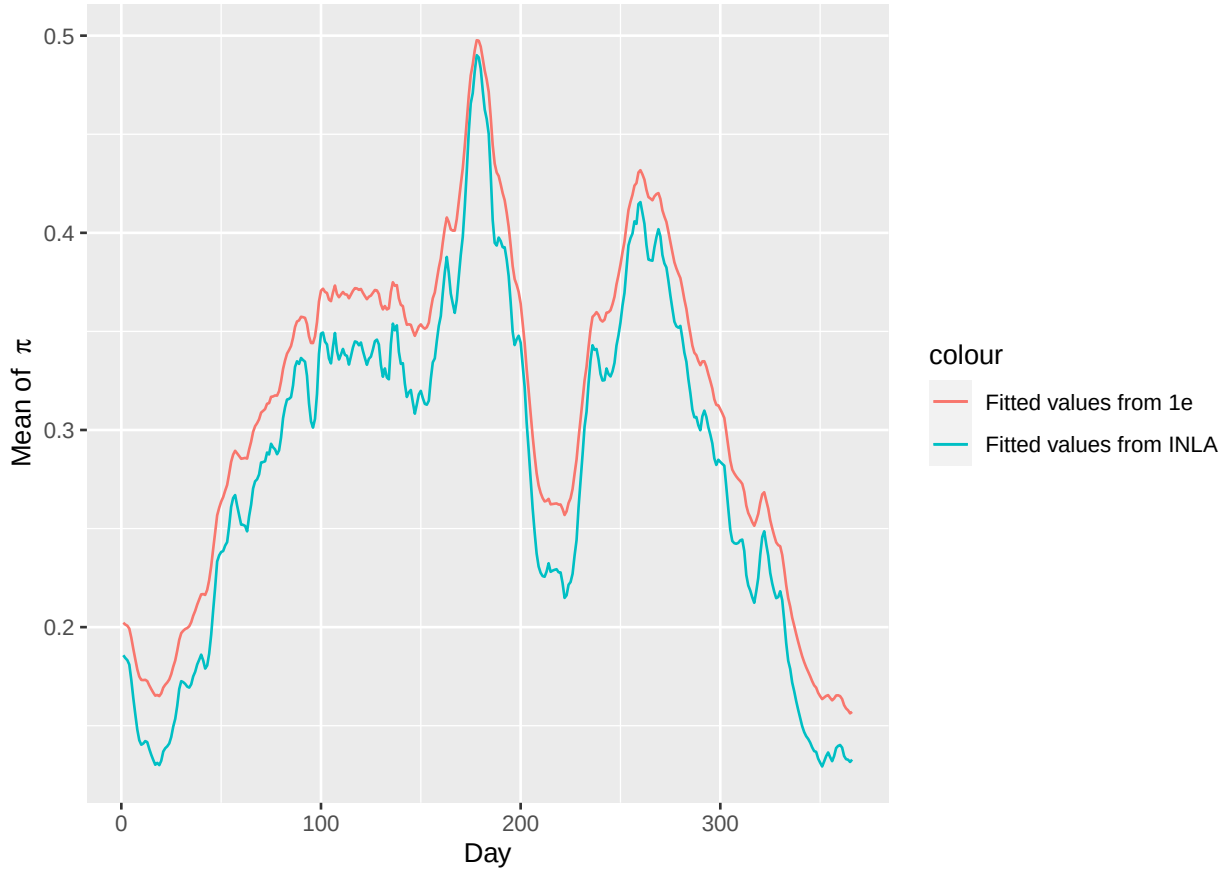


Figure 11: Comparison of INLA and model in 1e

here we see pretty different results. We also plot the predicted means for π for each day together with the mean values from problem 1f.

```
ggplot() + geom_line(data = as.data.frame(mod1$summary.fitted.values$mean), mapping = aes(x = 1:366,
    y = mod1$summary.fitted.values$mean, color = "Fitted values from INLA")) + geom_line(data = gg1f.df
    aes(x = 1:366, y = mpB, color = "Fitted values in 1f")) + xlab("Day") + ylab("Mean of " ~
    pi)
```

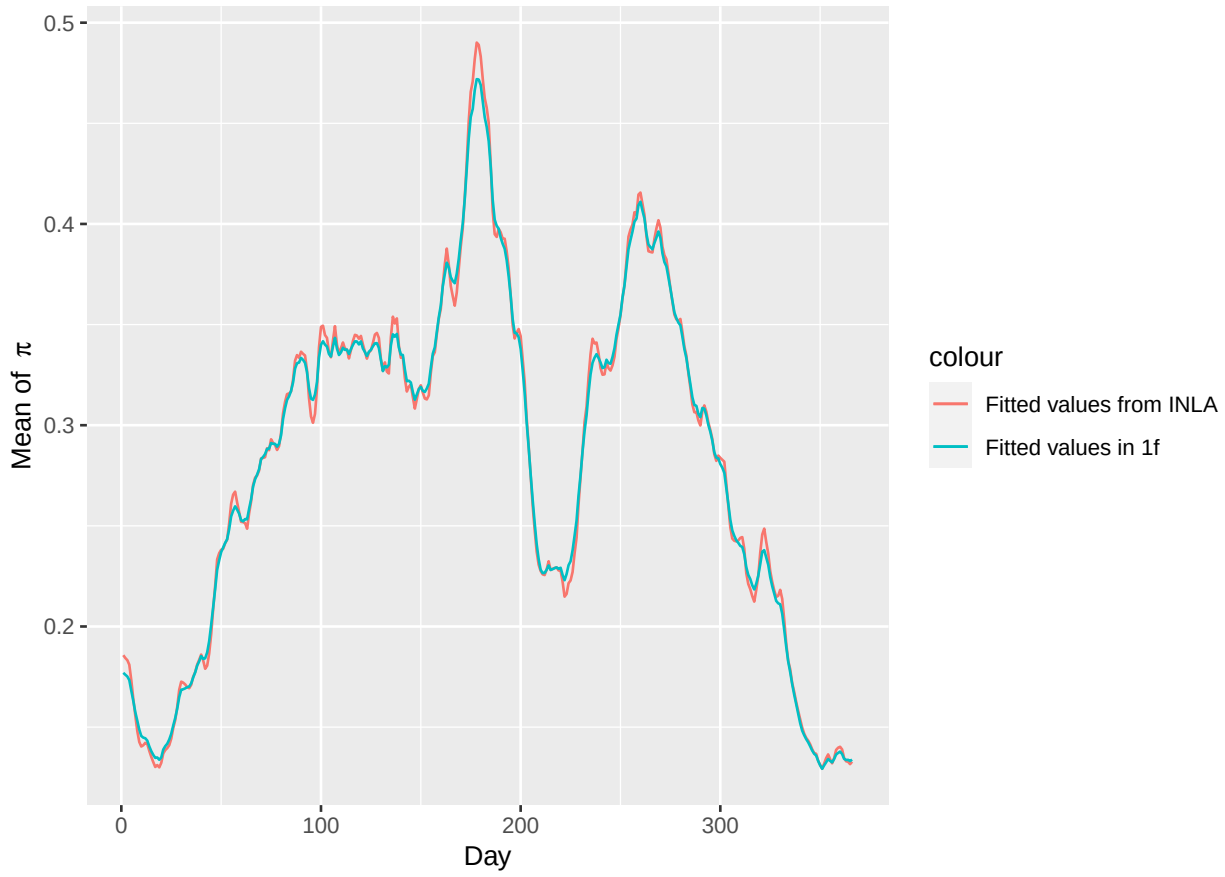


Figure 12: Comparison of INLA and model in 1f

We see that we get very similar results as expected. The computational time according to the proc.time()[3] is 1.41. This is much faster than the algorithms from problem 1.

b) Robustness of the results to different control.inla inputs

We want to look at how robust the results are to the two control.inputs. The strategy we used for integration is ccd. The ccd integration is a less costly alternative compared to the grid strategy when the dimensions of the hyperparameter is large. The grid strategy often gives the most accurate result, but the number of points grow exponentially with the dimension of θ. The ccd approach locates fewer points around the mode and is therefore less computationally expensive. Other options for integration strategies are 'eb', 'user' and 'user.std'. The 'auto' option which is the default integration strategy corresponds with the grid approach if the dimension of θ 2 or lower, and ccd if the dimension of θ is over 2. We have that the dimension of θ is 1, so the auto option will correspond with 'grid'.

The default option for strategy is the simplified Laplace option which is the option we have used here. Another option is the Gaussian approximation which is easy to apply and cheap to compute. However,

there could be errors in location or due to skeweness. Simplified Laplace can correct these errors and is computationally faster than the Laplace approximation.

We fit a couple of different models with different strategies and compare the results. We start by fitting some models with different integration strategies and the 'simplified.laplace' strategy for approximations. The first integration option we use is 'grid'.

```
# control.inla input
control.inla = list(strategy = "simplified.laplace", int.strategy = "grid")
# Fit model
fit2 <- INLA_fit(control.inla)
mod2 <- fit2$mod
time2 <- fit2$time
```

We plot the predictions together with the predictions from the first model.

```
INLA_plot_2(mod1, mod2)
```

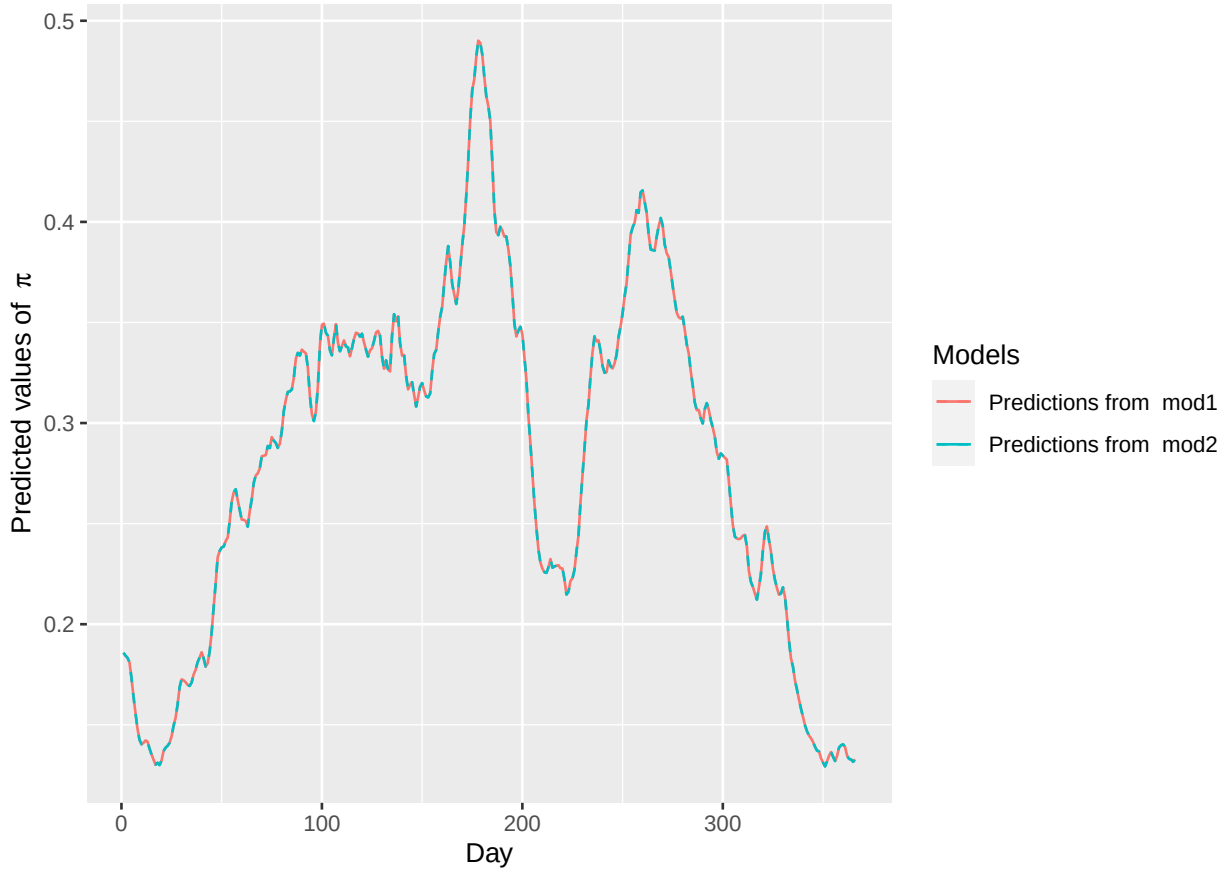


Figure 13: Plot of mean of fitted values of the model with strategy=simplified.laplace and int.strategy=grid compared to plot of model 1

As seen in figure 13, the results from the two models are very similar. The computational time we get by using proc.time()[3] is 1.01, which is barely higher than the first model.

The next integration strategy considered is the 'eb' strategy.

```
control.inla = list(strategy = "simplified.laplace", int.strategy = "eb")
# Time before
fit3 <- INLA_fit(control.inla)
mod3 <- fit3$mod
time3 <- fit3$time
```

We plot it together with the first model

```
INLA_plot_2(mod1, mod3)
```

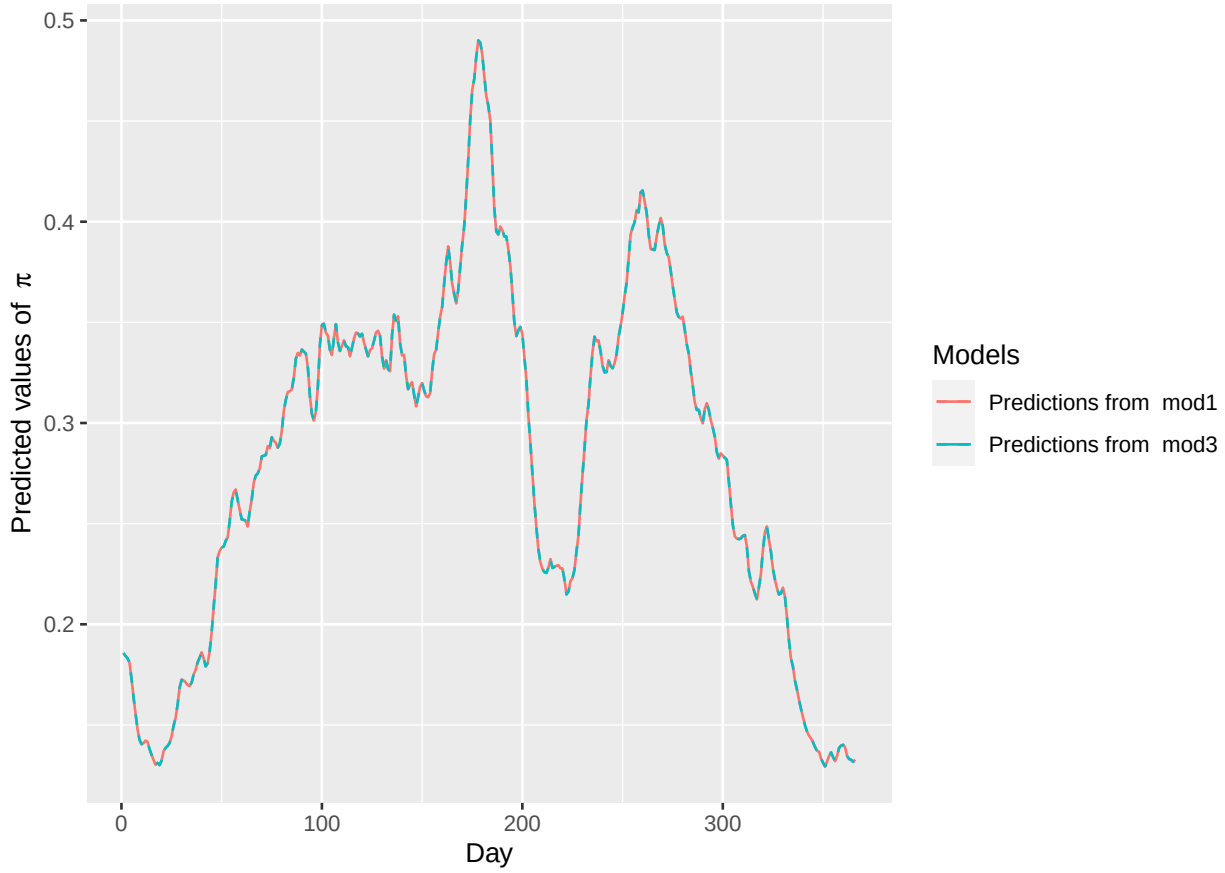


Figure 14: Plot of mean of fitted values of the model with strategy=simplified.laplace and int.strategy=eb compared to plot of model 1

Plot 14 shows that this model yields very similar predictions as the two other models. The computation time is here 0.8, which is about the same as the time when using 'grid'. The results for the different integration strategies are very similar. We now look at different strategies for approximation and use ccd as the method for integration strategy. We start by trying the Gaussian approximation.

```
control.inla = list(strategy = "gaussian", int.strategy = "ccd")
fit4 <- INLA_fit(control.inla)
mod4 <- fit4$mod
time4 <- fit4$time
```

```
INLA_plot_2(mod1, mod4)
```

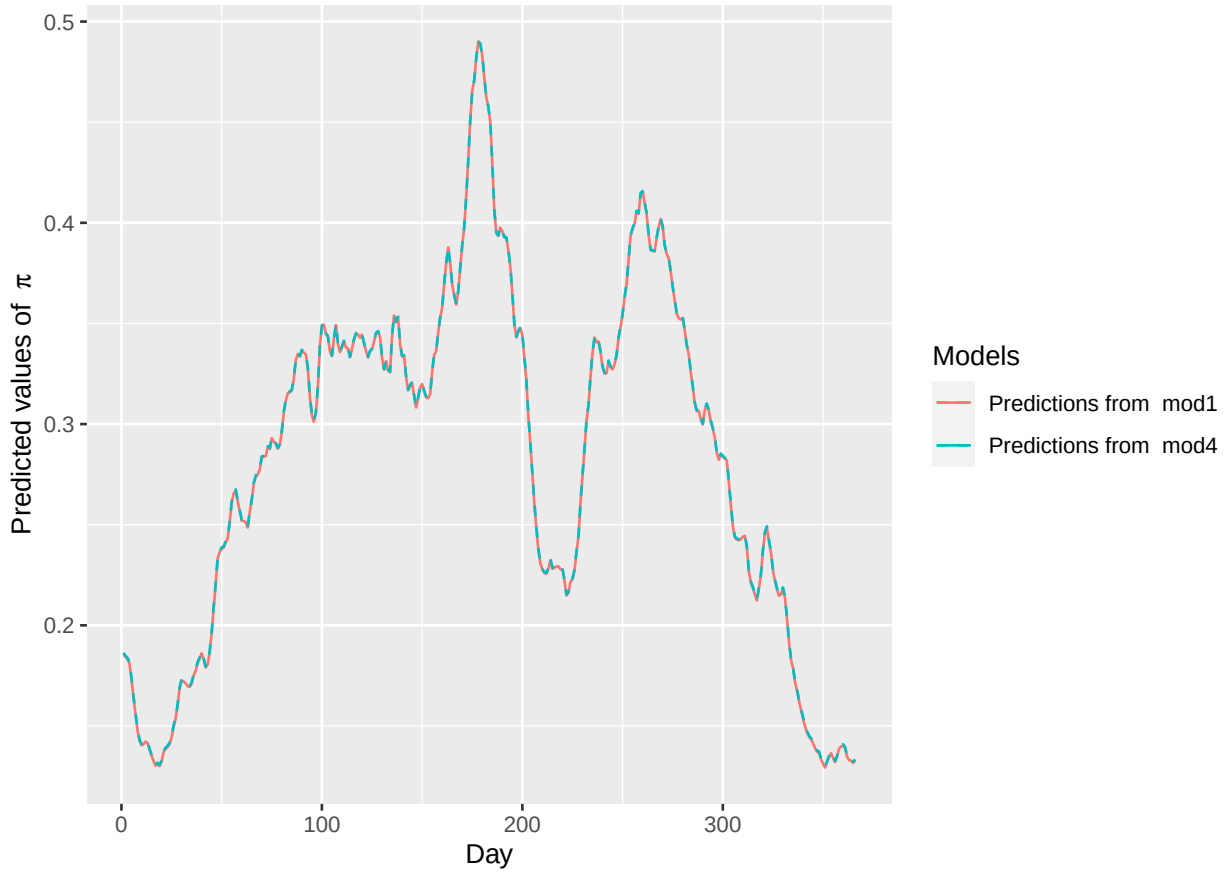


Figure 15: Plot of mean of fitted values of the model with strategy=gaussian and int.strategy=ccd compared to plot of model 1

As seen in 15, we yet again get very similar results. The computation time is 0.92. Now, we use the Laplace approximation.

```
control.inla = list(strategy = "laplace", int.strategy = "ccd")
fit5 <- INLA_fit(control.inla)
mod5 <- fit5$mod
time5 <- fit5$time
```

```
INLA_plot_2(mod1, mod5)
```

Figure 16 shows that for the first days the predictions are very similar. However, from about day 100 and onwards we can see significant differences between the models. The running time is 1.34, which is more compared to the other models. We also try the Laplace approach with the grid approximation for integration.

```
control.inla = list(strategy = "laplace", int.strategy = "grid")
fit6 <- INLA_fit(control.inla)
mod6 <- fit6$mod
time6 <- fit6$time
```

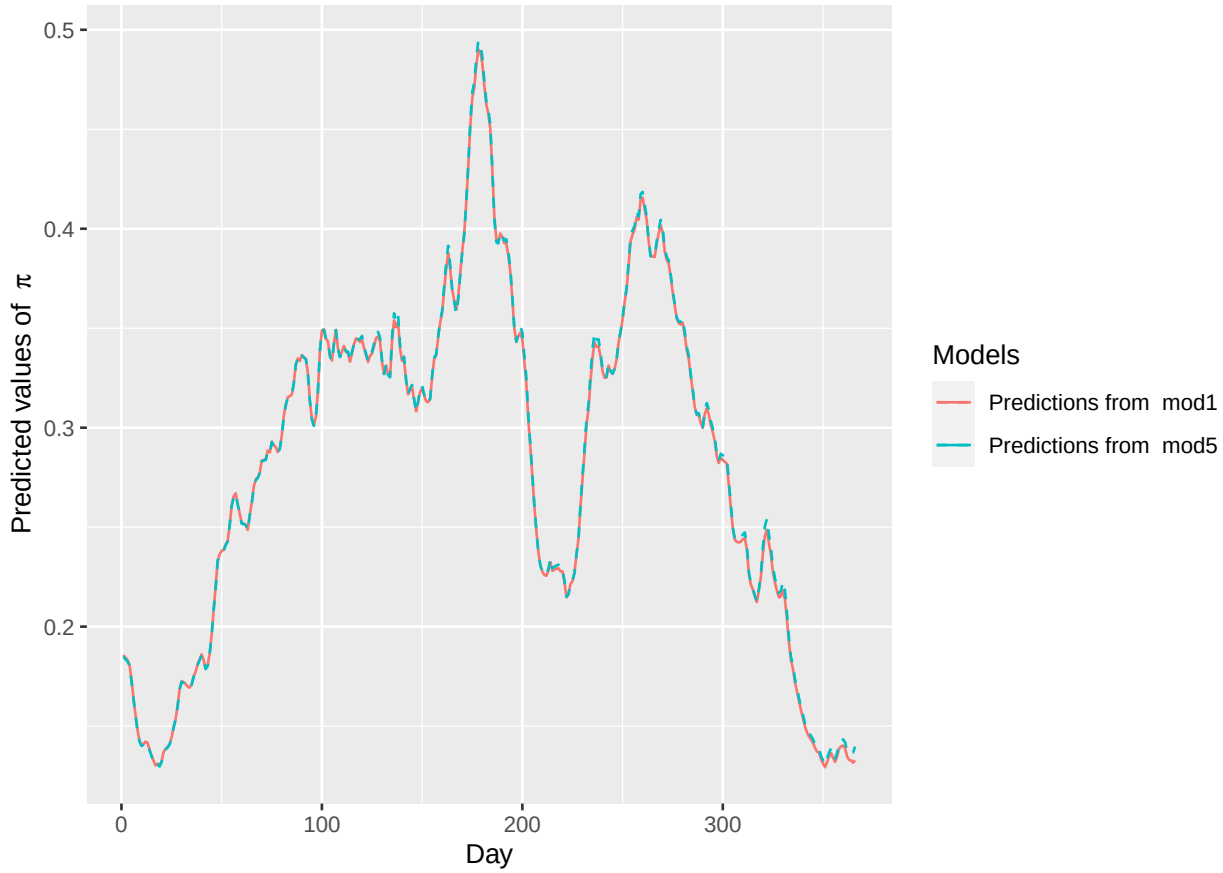


Figure 16: Plot of fitted values with strategy=laplace and int.strategy=ccd compared to model 1

We compare this to the previous model

```
INLA_plot_2(mod5, mod6)
```

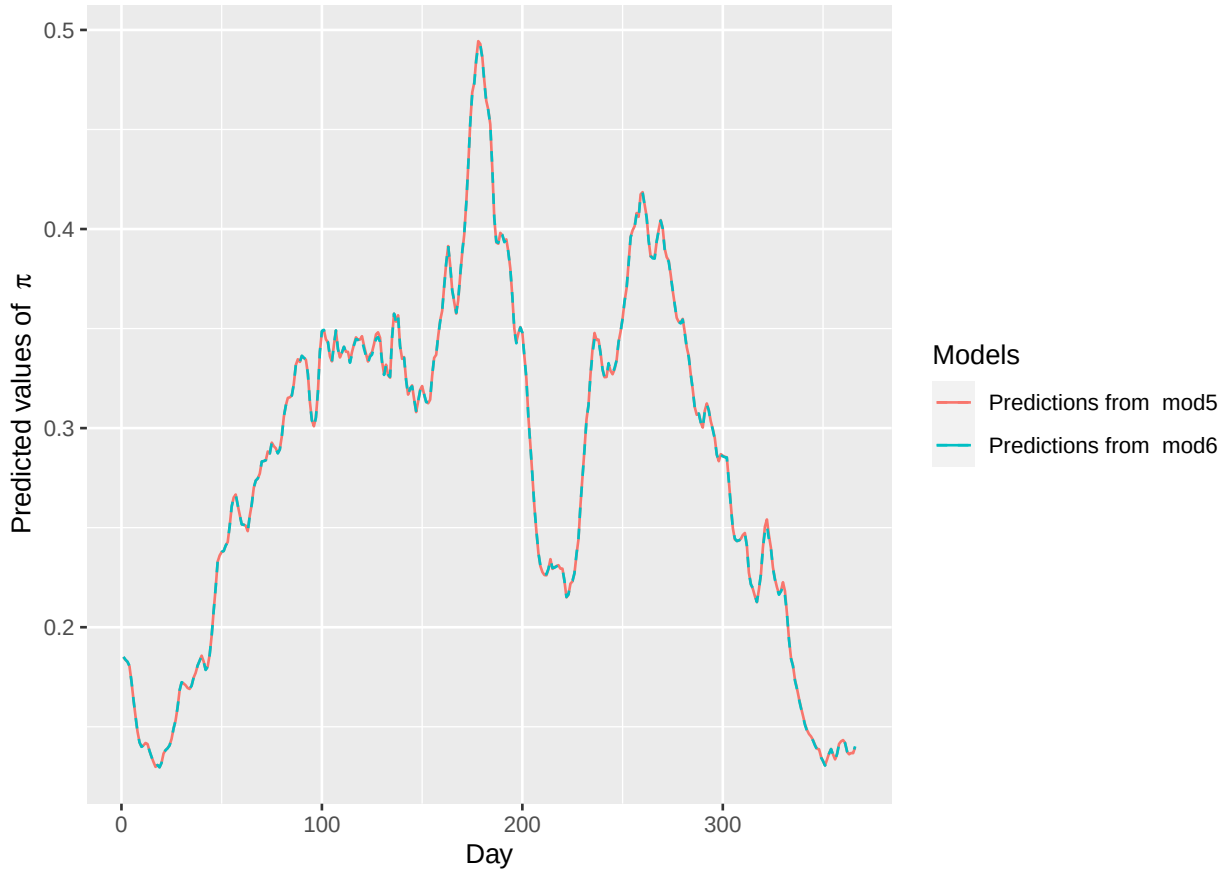


Figure 17: Plot of fitted values with strategy=laplace and int.strategy=grid compared to the values when using strategy=laplace and int.strategy=ccd which is model 5

The predictions are very similar, but we see some differences from about day 100 to day 150. The computation time is 1.94

The Laplace approximation strategy gives different predictions compared to the predictions we get when using other strategies. At last, we look at the 'adaptive' strategy and compare it to our original model.

```
control.inla = list(strategy = "adaptive", int.strategy = "ccd")
fit7 <- INLA_fit(control.inla)
mod7 <- fit7$mod
time7 <- fit7$time
```

We compare the predictions of this model to the predictions of the first model. The running time for fitting this model is 0.98.

The different inputs for control.inla mostly yields very similar results. However, we get different results when using the Laplace approximation as strategy. This imply that the results are not that robust for the different control.inla inputs we use. We also notice a longer running time when using Laplace for approximations. The running time is also larger when using `grid` compared to the other integrartion strategies. All these running times are all very low compared to our previous results from problem 1.

c) INLA model 2

We fit a new model that is fitted with the code below. We refer to this model as model 8.

```
alpha = 2
beta = 0.05
hyper = list(theta = list(prior = "loggamma", param = c(alpha, beta)))
control.inla = list(strategy = "simplified.laplace", int.strategy = "ccd")

mod8 <- inla(n.rain ~ f(day, model = "rw1", hyper = hyper, constr = TRUE), data = rain,
    Ntrials = n.years, control.compute = list(config = TRUE), family = "binomial",
    verbose = TRUE, control.inla = control.inla)
```

The model in problem 2a is has a response given by equation (1). The first mathematical difference of the new model compared to the first model is that the intercept is not removed. This means that the response is given by

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

where $\eta_t = \beta_0 + \tau_t$, and β_0 is the intercept. We use the default prior on the intercept which is a Gaussian distribution with the mean and precision equal to zero. This is an improper prior where the variance is infinite. The prior on θ is the same as in the model in 2a, that is a Loggamma$(\alpha = 2, \beta = 0.05)-$ distribution. Unlike the previous model, this model uses the argument `constr=TRUE`. This means that there is a sum-to-zero constraint, and the sum of $(\tau_1, .., \tau_n)$ is given by

$$\sum_{i=1}^{n} \tau_i = 0.$$

In the model 8, we have that η_t is given by

$$\eta_t = \log\left(\frac{\pi(\eta_t)}{1-\pi(\eta_t)}\right) \implies \tau_t = \log\left(\frac{\pi(\eta_t)}{1-\pi(\eta_t)}\right) - \alpha$$

In the model in 2a, we have

$$\tau_t = \log\left(\frac{\pi(\tau_t)}{1-\pi(\tau_t)}\right).$$

Below, we plot the estimated mean values of τ_t for $t = 1, .., T$ for both models.

```
tau_1 = mod1$summary.random$day$mean
tau_8 = mod8$summary.random$day$mean
label1 = c("Model 1", "Model 8")
ggplot() + geom_line(data = as.data.frame(tau_1), mapping = aes(x = 1:366, y = tau_1,
    color = "Model 1")) + geom_line(data = as.data.frame(tau_8), mapping = aes(x = 1:366,
    y = tau_8, color = "model 8")) + xlab("Day") + ylab("estimated values of" ~ tau) +
    labs(color = "Models")
```

As seen in figure 18, the predicted τ-values are different for the two models. The model with the intercept and sum-to-zero constraint is supposed to have tau-values that sum to zero, and from the figure it seems like that this is the case. The model without the intercept does not have τ-values that sums to zero as seen in the figure. Nevertehelss, we can see that the to graphs have what seems like identical shapes. The predicted $\pi-$values of the models in 2a and 2c are also plotted together.

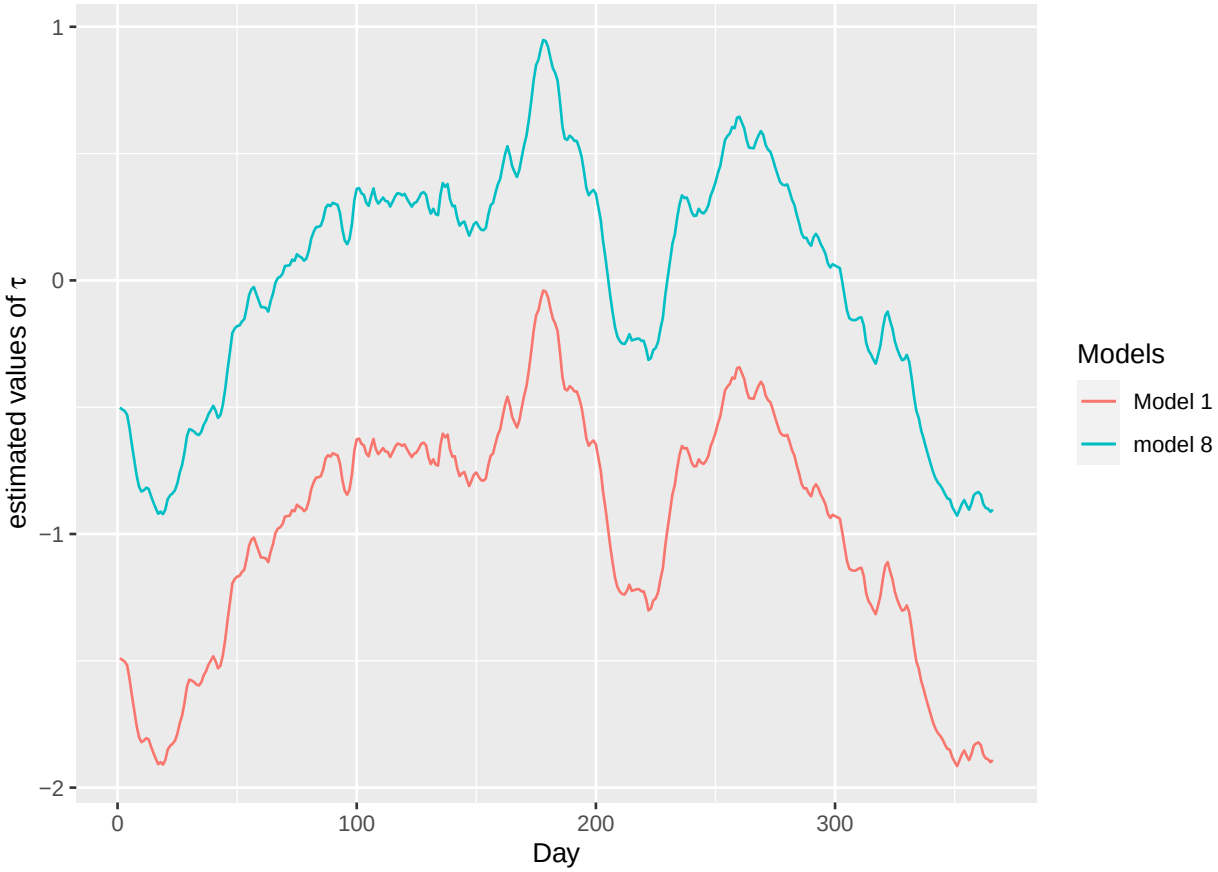


Figure 18: τ-values of model 1 and 8

```
pi_1 <- mod1$summary.fitted.values$mean
pi <- mod8$summary.fitted.values$mean
ggplot() + geom_line(data = as.data.frame(pi_1), mapping = aes(x = 1:366, y = pi_1,
    color = "Model 1"), linetype = 1) + geom_line(data = as.data.frame(pi), mapping = aes(x = 1:366,
    y = pi, color = "Model 8"), linetype = 2) + xlab("Day") + ylab("Predicted values of " ~
    pi) + labs(color = "Models")
```

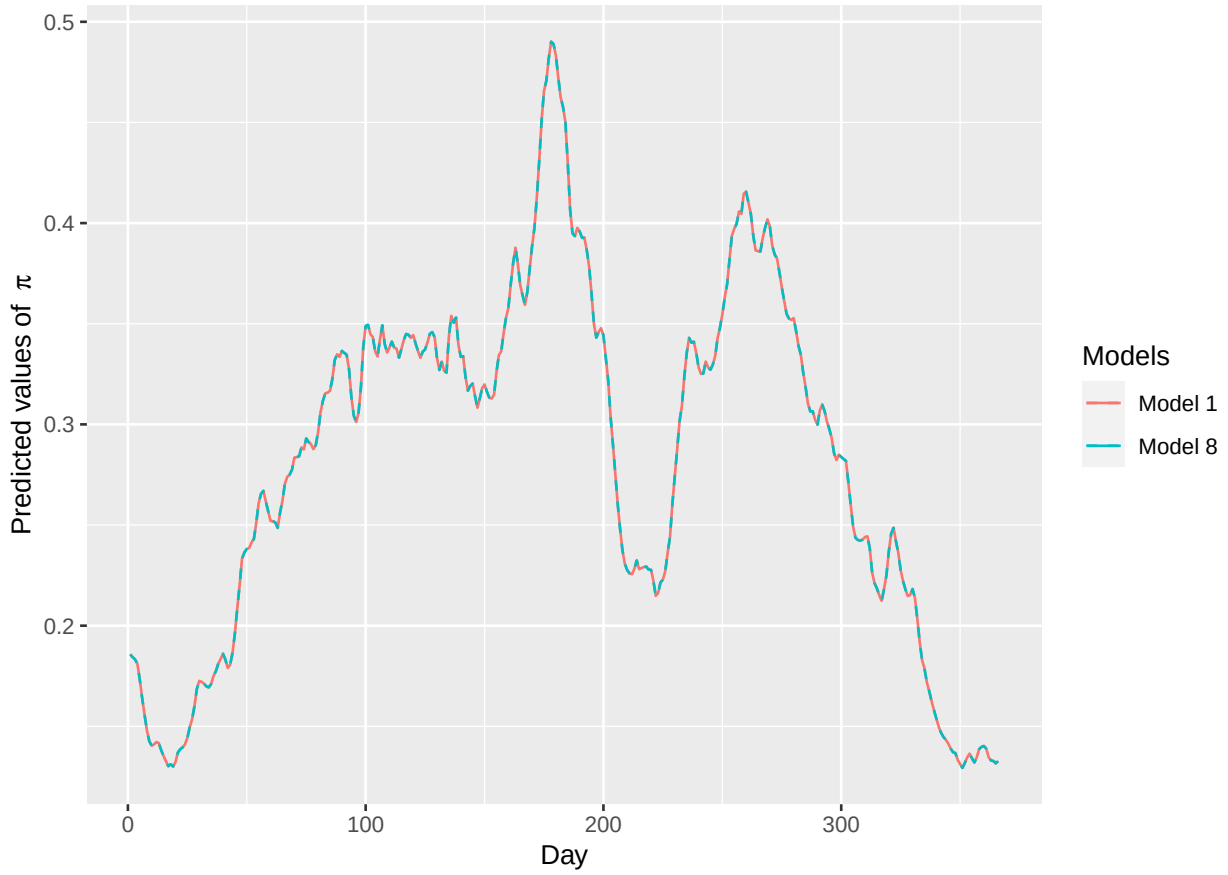


Figure 19: Predicted π-values of the two models

From figure 19 we can't see any significant differences. To further show how similar the predictions are, we can look at π for day 1, 201 and 366.

```
mod8$summary.fitted.values[1, ]
```

```
##                            mean         sd 0.025quant  0.5quant 0.975quant
## fitted.Predictor.001 0.1857528 0.02889692  0.1336366 0.1841919  0.2467205
##                           mode
## fitted.Predictor.001 0.18106
```

```
mod8$summary.fitted.values[201, ]
```

```
##                            mean         sd 0.025quant 0.5quant 0.975quant
## fitted.Predictor.201 0.3329547 0.02871596  0.2785428 0.332292  0.3911305
```

```
##                            mode
## fitted.Predictor.201 0.3309667
```

```
mod8$summary.fitted.values[366, ]
```

```
##                            mean         sd 0.025quant  0.5quant 0.975quant
## fitted.Predictor.366 0.1328183 0.02346726 0.09103573 0.1313672  0.1828478
##                            mode
## fitted.Predictor.366 0.1284748
```

```
intercept <- mod8$summary.fixed$mean
```

These predictions are very similar, and the first differences can be seen in the fith decimal. The estimated intercept term is -0.9876664.

The reason for the similar predictions is that the intercept term in the model with the constraint shifts down the values such that the predicted values of π are almost identical for the two models. The intercept term makes the model as flexible as the model in 2a, and this lead to the same fitted values. If this intercept term wasn't included, the model would be constrained by the sum-to zero constraint and we would get different predictions. The fact that we get so similar predictions can also be shown more formally by looking at the marginal posterior distributions.