TMA4300 Project 2

Magnus Grytten & Petter J. Gudbrandsen

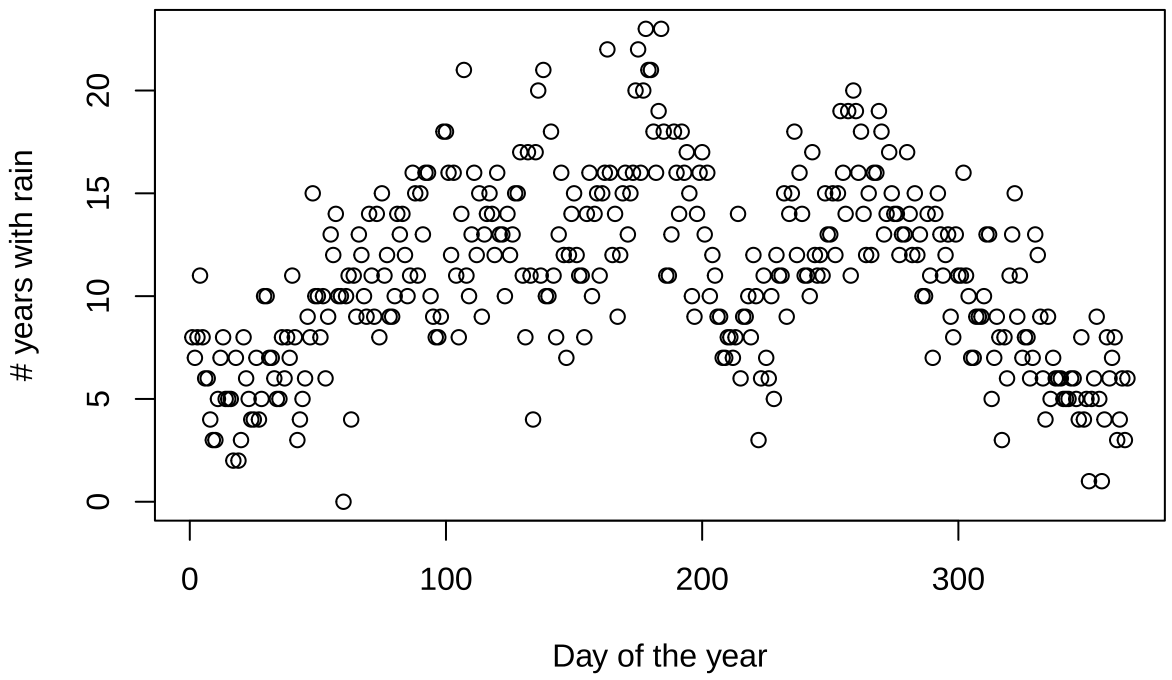
```r
# Import classes
library("INLA")
library(invgamma)
library(matlib)
library(MASS)
library(latex2exp)

# Load data
load("rain.rda") # set working directory correctly
```

Problem 1

a)

Explore the Tokyo rainfall dataset, plot the response as a function of t, and describe any patterns that you see.

```r
plot(rain$day, rain$n.rain, xlab = 'Day of the year', ylab = '# years with rain')
```



The plot shows the number of days with rain in each day of the year over a period of 39 years.

During the winter period there is little rain. It increases during the spring. After that it seems to have two 'down' bumps during the summer. During the autumn, from about day 275 and until new year, it decreases.

The points seem to follow a continuous function with a similar error around it throughout the year.

b)

The likelihood of y_t given $\pi(\tau_t)$ is given to be a binomial distribution.

$$p(y_t|\pi(\tau_t)) = \binom{n_t}{y_t}\pi(\tau_t)^{y_t}(1-\pi(\tau_t))^{n_t-y_t}$$

c)

To find the conditional $p(\sigma^2|\boldsymbol{y},\boldsymbol{\tau})$, we first use Bayes' theorem and the chain rule for conditional probability to obtain,

$$\begin{aligned}
p(\sigma^2|\boldsymbol{y},\boldsymbol{\tau}) \propto & p(\boldsymbol{y},\boldsymbol{\tau}|\sigma^2)\cdot p(\sigma^2) \\
=& p(\boldsymbol{y}|\boldsymbol{\tau},\sigma^2)\cdot p(\boldsymbol{\tau}|\sigma^2)\cdot p(\sigma^2) \\
=& p(\boldsymbol{y}|\boldsymbol{\tau})\cdot p(\boldsymbol{\tau}|\sigma^2)\cdot p(\sigma^2) \\
=& \prod_{t=1}^{T}\binom{n_t}{y_t}\pi(\tau_t)^{y_t}(1-\pi(\tau_t))^{n_t-y_t} \\
& \cdot\prod_{t=2}^{T}\frac{1}{\sigma_u}e^{-\frac{1}{2\sigma_u^2}(\tau_t-\tau_{t-1})^2} \\
& \cdot\frac{\beta^\alpha}{\Gamma(\alpha)}(\frac{1}{\sigma_u^2})^{\alpha+1}e^{-\frac{\beta}{\sigma_u^2}}
\end{aligned}$$

We remove factors without σ_u.

$$\begin{aligned}
\propto & \prod_{t=2}^{T}\frac{1}{\sigma_u}e^{-\frac{1}{2\sigma_u^2}(\tau_t-\tau_{t-1})^2}\cdot(\frac{1}{\sigma_u^2})^{\alpha+1}\cdot e^{-\frac{\beta}{\sigma_u^2}} \\
=& \frac{1}{\sigma_u^{T-1}}e^{-\frac{1}{2\sigma_u^2}\boldsymbol{\tau}^\intercal\boldsymbol{Q}\boldsymbol{\tau}}\cdot(\frac{1}{\sigma_u^2})^{\alpha+1}\cdot e^{-\frac{\beta}{\sigma_u^2}} \\
=& (\frac{1}{\sigma_u^2})^{\alpha+1+\frac{T-1}{2}}e^{-\frac{1}{2\sigma_u^2}(\boldsymbol{\tau}^\intercal\boldsymbol{Q}\boldsymbol{\tau}+\beta)}
\end{aligned}$$

In the last expression we recognize the core of an inverse gamma distribution:

$$IG\sim(\alpha+\frac{T-1}{2},\frac{\boldsymbol{\tau}^\intercal\boldsymbol{Q}\boldsymbol{\tau}+\beta}{2})$$

for shape α and rate β.

d)

Since assume conditional independence among the $y_t|\tau_t$ we have,

$$p(\boldsymbol{y}|\boldsymbol{\tau})=\prod_{t=1}^{T}p(y_t|\tau_t)=p(\boldsymbol{y_I}|\boldsymbol{\tau_I})\cdot p(\boldsymbol{y_{-I}}|\boldsymbol{\tau_{-I}})$$

To find the acceptance probability we start by looking at the posterior $p(\boldsymbol{\tau},\sigma^2|\boldsymbol{y})$, were we again use the Bayes' theorem and the chain rule, as well as the expression above to find,

$$\begin{aligned}
p(\boldsymbol{\tau},\sigma^2|\boldsymbol{y}) \propto & p(\boldsymbol{y}|\boldsymbol{\tau},\sigma^2)\cdot p(\boldsymbol{\tau},\sigma^2) \\
=& p(\boldsymbol{y}|\boldsymbol{\tau})\cdot p(\boldsymbol{\tau}|\sigma^2)\cdot p(\sigma^2) \\
=& p(\boldsymbol{y}_I|\boldsymbol{\tau}_I)\cdot p(\boldsymbol{y}_{-I}|\boldsymbol{\tau}_{-I})\cdot p(\boldsymbol{\tau}_{-I}|\sigma^2)\cdot p(\boldsymbol{\tau}_I|\boldsymbol{\tau}_{-I},\sigma^2)\cdot p(\sigma^2)
\end{aligned}$$

The acceptance probability is given by,

$$\alpha=min(1,\frac{p(\boldsymbol{\tau'}_I,\sigma^2|\boldsymbol{y})\cdot p(\boldsymbol{\tau}_I|\boldsymbol{\tau'}_{-I},\sigma^2)}{p(\boldsymbol{\tau}_I,\sigma^2|\boldsymbol{y})\cdot p(\boldsymbol{\tau'}_I|\boldsymbol{\tau}_{-I},\sigma^2)})$$

Inserting the expression for $p(\boldsymbol{\tau},\sigma^2|\boldsymbol{y})$ in the the expression for the acceptance probability we get,

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

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

Since we only propose new values for $\boldsymbol{\tau'}_I$ we have that $\boldsymbol{\tau'}_{-I} = \boldsymbol{\tau}_{-I}$, inserting this in the expression above we get,

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

Here almost all the terms cancel out leaving giving us with

$$\alpha = min(1, \frac{p(\boldsymbol{y}_I|\boldsymbol{\tau'}_I)}{p(\boldsymbol{y}_I|\boldsymbol{\tau}_I)})$$

## e) First we make some functions that we are going to need for the sampler,

```
# Tau to pi
pi_func <- function(tau){
  1/(1+exp(-tau))
}

# Pi to tau
pi_inv <- function(pi){
  log(pi/(1-pi))
}
```

We have that τ_t is normaly distributed with mean $\boldsymbol{Q}_{t,t}^{-1}\boldsymbol{Q}_{t,-t}\boldsymbol{\tau}_{-t}$ and variance $\sigma^2\boldsymbol{Q}_{t,t}^{-1}$, however since Q is mostly zeroes we can simplify this. Simplifying we get,

$$\begin{aligned}
\tau_1 &\sim N(\tau_2, \sigma^2) \\
\tau_{1<t<T} &\sim N\left(\frac{1}{2}(\tau_{t-1}+\tau_{t-1}), \frac{1}{2}\sigma^2)\right) \\
\tau_T &\sim N(\tau_{T-1}, \sigma^2)
\end{aligned}$$

```
mcmc <- function(N){
  t0 <-proc.time()[3]
  accepted_count <- 0
  count <- 0

  alpha <- 2
  beta <- 0.05

  T <- length(rain$day)

  tau <- matrix(nrow = N+1, ncol = T)
  tau[1,] <- rep(pi_inv(0.3), T) #Setting tau to be the aproximate mean of yt/nt


  #Make Q matrix
  Q <- matrix(data = 0,nrow = T, ncol = T)
  diag(Q) <- rep(2,T)
  diag(Q[-nrow(Q),-1]) <- rep(-1,T-1)
  diag(Q[-1,-ncol(Q)]) <- rep(-1,T-1)
  Q[1,1] <- 1
```

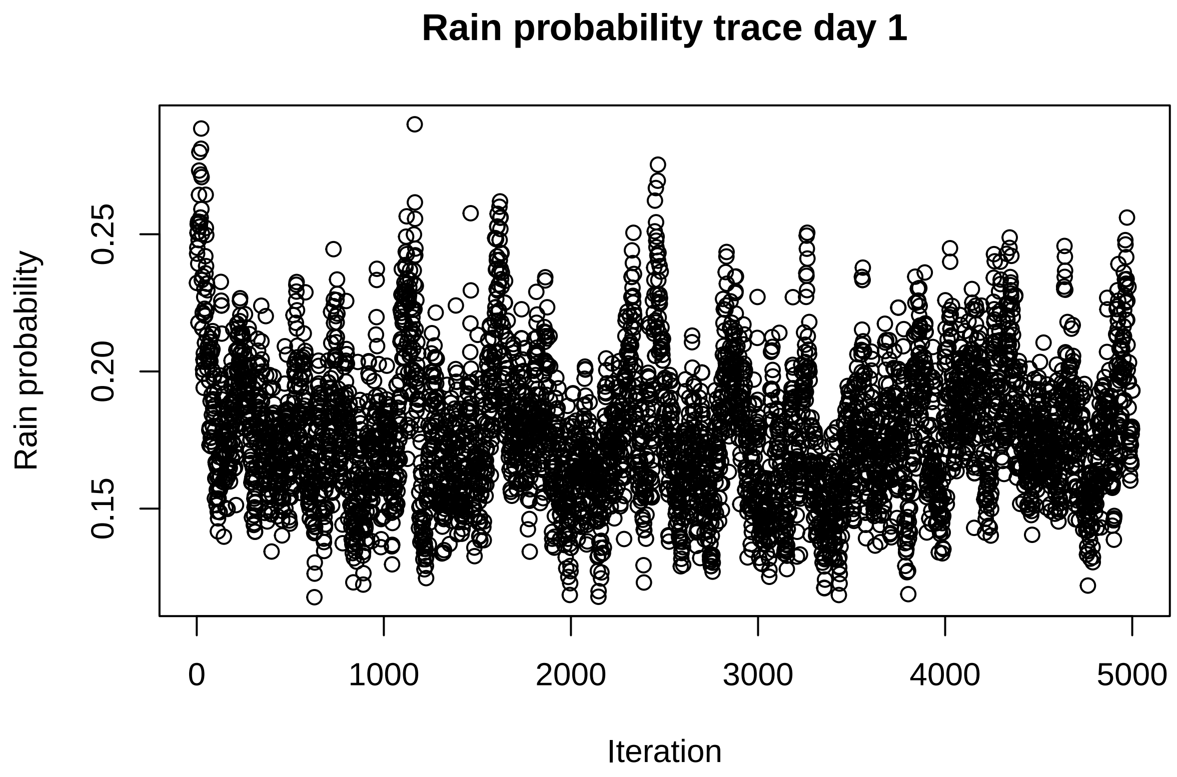
```r
  Q[T,T] <- 1


  sigma_squared <- 1:N


  for (i in c(1:N)){
    #Sampling sigma from the conditional found in task c
    sigma_squared[i] <- rinvgamma(n = 1, shape = alpha + (T-1)/2,
                                  scale = beta + 1/2 * t(tau[i,]) %*% Q %*% tau[i,] )
    sigma <- sqrt(sigma_squared[i])


    for (d in c(1:T)){

      #Sampling tau from precomputed distribution
      if (d == 1){
        new_tau_d <- rnorm(n=1, mean = tau[i,d+1], sd = sigma)
      }
      else if (d == T){
        new_tau_d <- rnorm(n=1, mean = tau[i,d-1], sd = sigma)
      }
      else{
        new_tau_d <- rnorm(n=1, mean = 1/2*(tau[i,d-1]+tau[i,d+1]), sd = sigma/sqrt(2))
      }

      old_tau_d <- tau[i, d]

      #Calculate acceptance probability
      prob <- dbinom(rain$n.rain[d], size = rain$n.years[d], pi_func(new_tau_d))/
              dbinom(rain$n.rain[d], size = rain$n.years[d], pi_func(old_tau_d))

      u <- runif(1)
      if (u < min(1, prob)){
        tau[i, d] <- new_tau_d
        accepted_count <- accepted_count + 1
      }
    }

    tau[(i+1),] <- tau[i,]
  }

  print('Processing time:')
  print(proc.time()[3] - t0)

  print('Acceptance probability:')
  print(accepted_count / (366*N))

  list(
    pi = apply(tau, 2, pi_func),
    sigma_squared = sigma_squared
  )
}
```

```
# Run sampler
N = 5000
mcmc_sim <- mcmc(N)
```

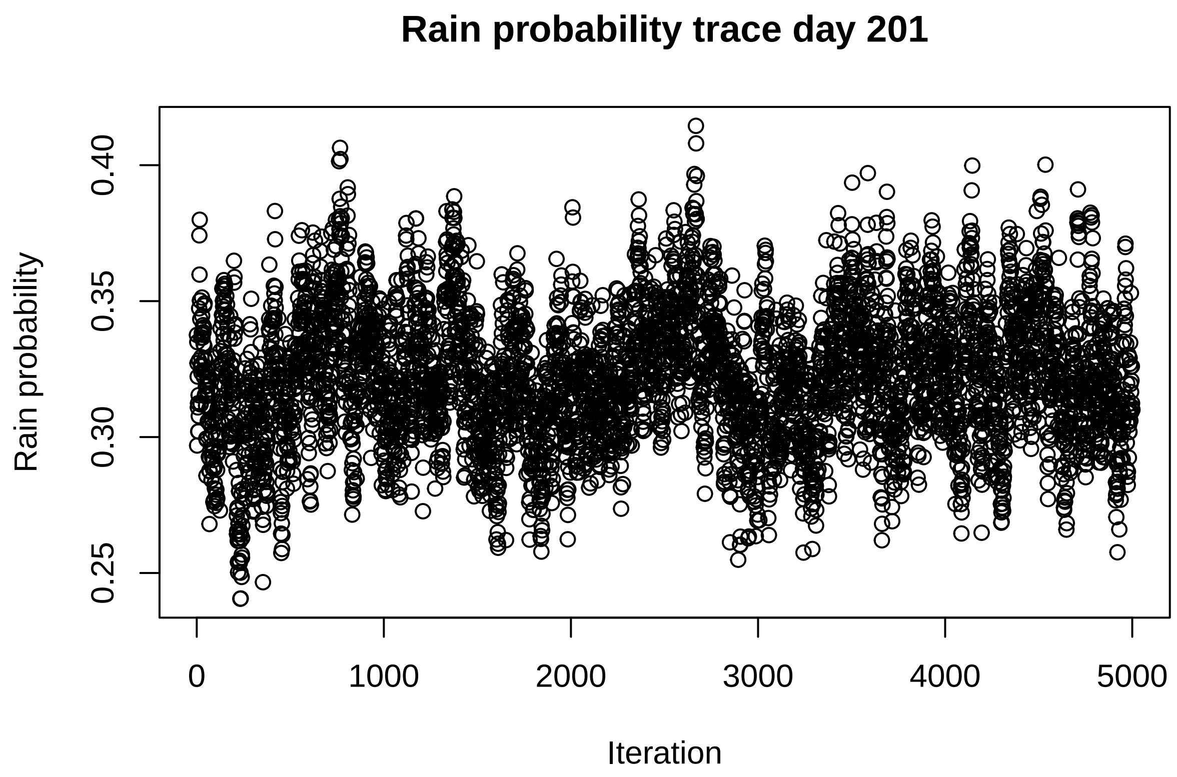
```
## [1] "Processing time:"
## elapsed
##    46.6
## [1] "Acceptance probability:"
## [1] 0.9378596
```

```
sim_pi <- mcmc_sim$pi
sim_sigsq <- mcmc_sim$sigma_squared
```

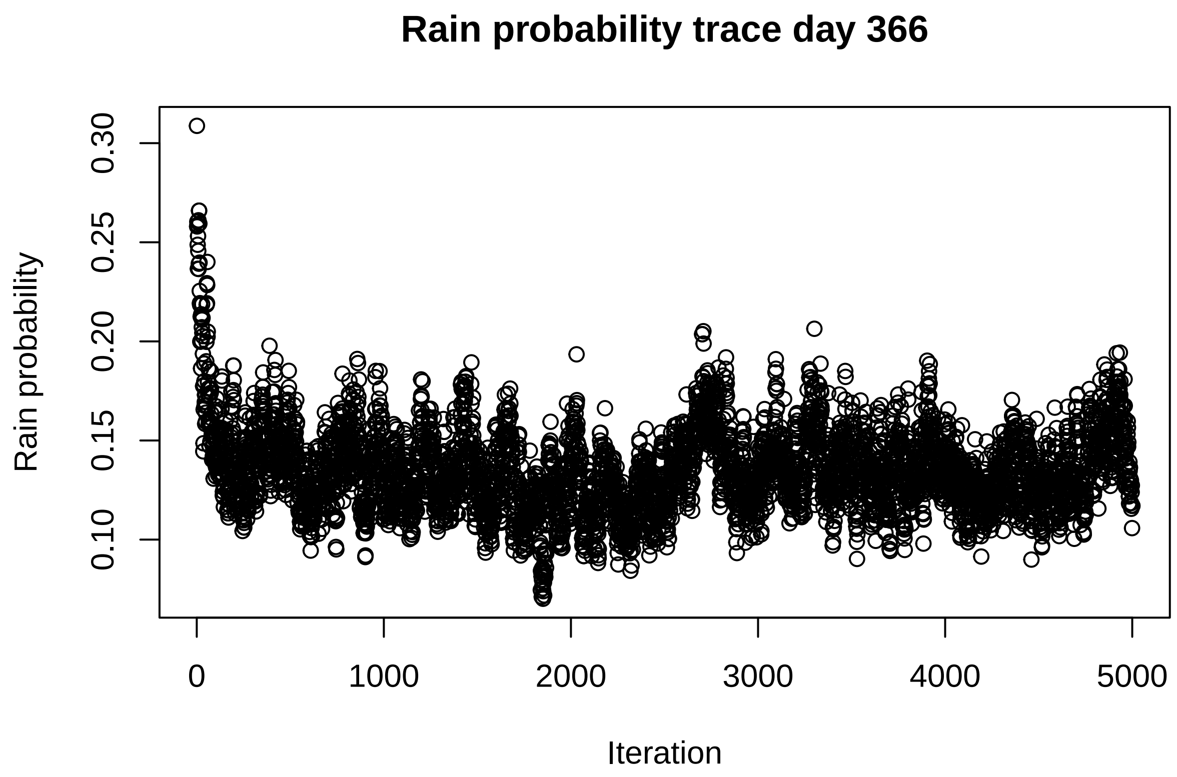
```
# Traceplots
plot(sim_pi[, 1], main = c('Rain probability trace day 1'), xlab = 'Iteration', ylab = 'Rain probabilit
```



```
plot(sim_pi[, 201], main = c('Rain probability trace day 201'), xlab = 'Iteration', ylab = 'Rain probab
```



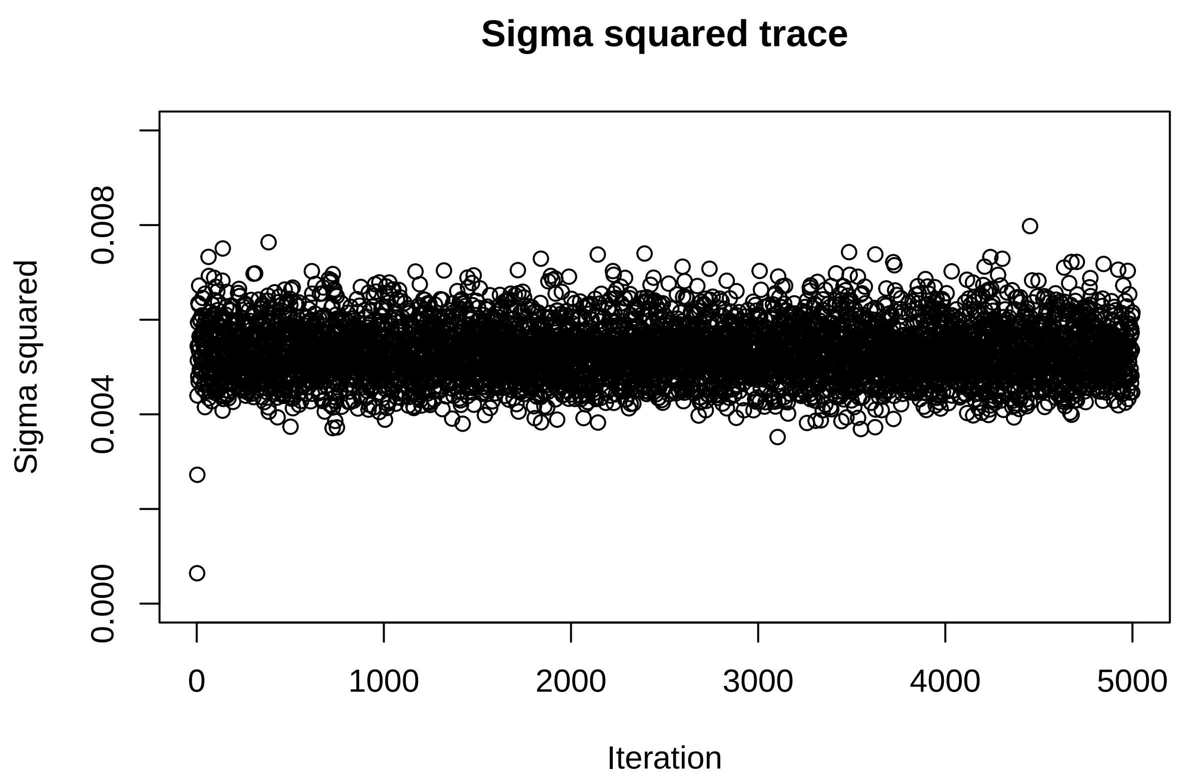
```
plot(sim_pi[, 366], main = c('Rain probability trace day 366'), xlab = 'Iteration', ylab = 'Rain probabi
```



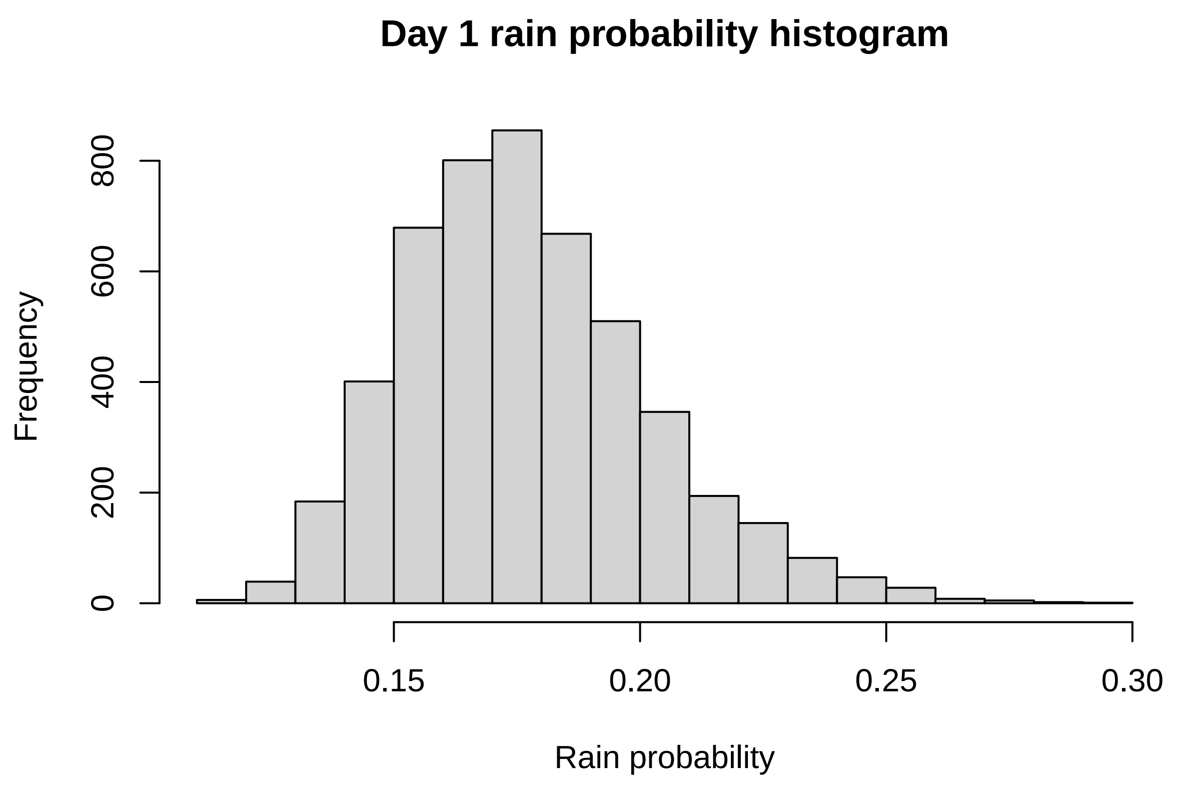
```
print(length(sim_sigsq))
```

```
## [1] 5000
```

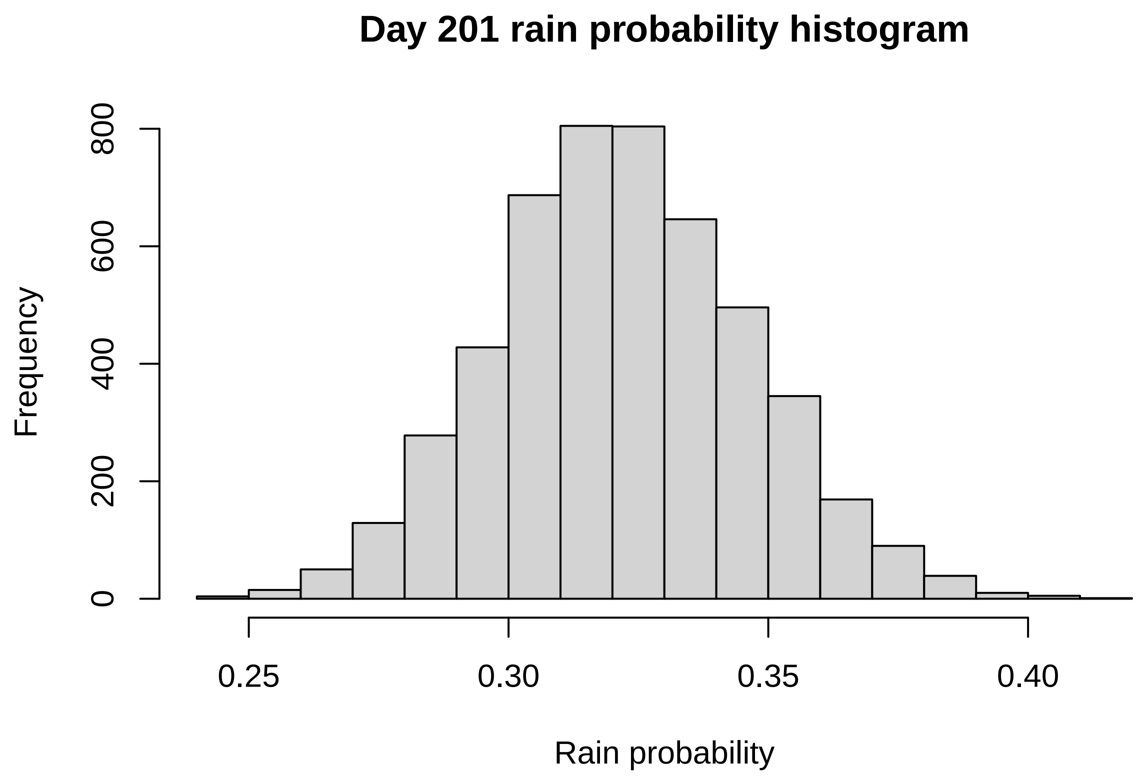
```
plot(1:N, sim_sigsq, main = 'Sigma squared trace', xlab = 'Iteration', ylab = 'Sigma squared', ylim = c
```



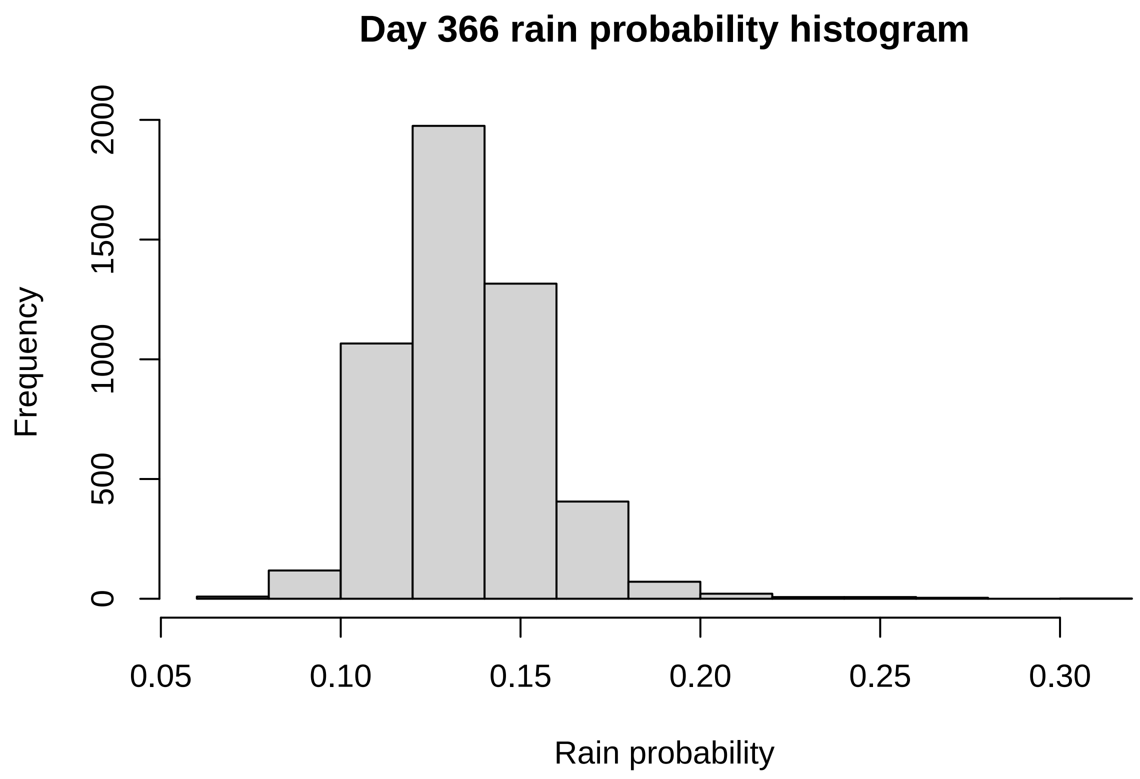
```
# Histogram
hist(sim_pi[, 1], main = c('Day 1 rain probability histogram'), freq = TRUE, xlab = 'Rain probability',
```



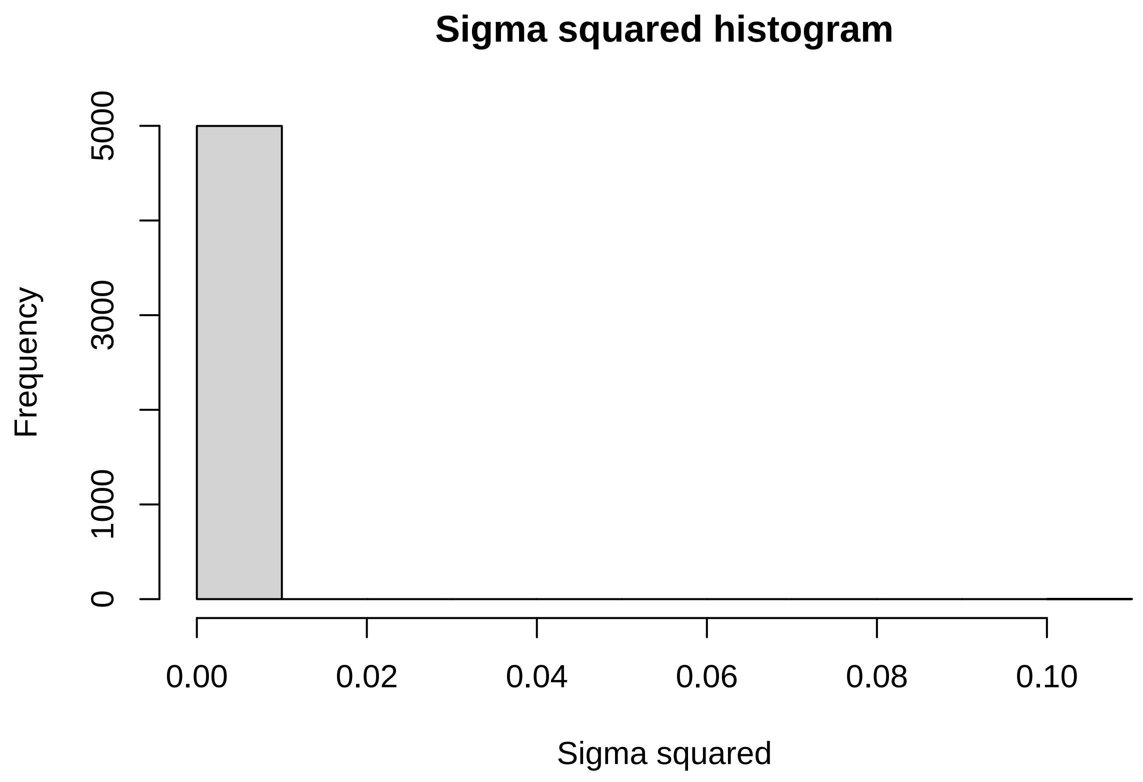
```
hist(sim_pi[, 201], main = c('Day 201 rain probability histogram'), freq = TRUE, xlab = 'Rain probabili
```



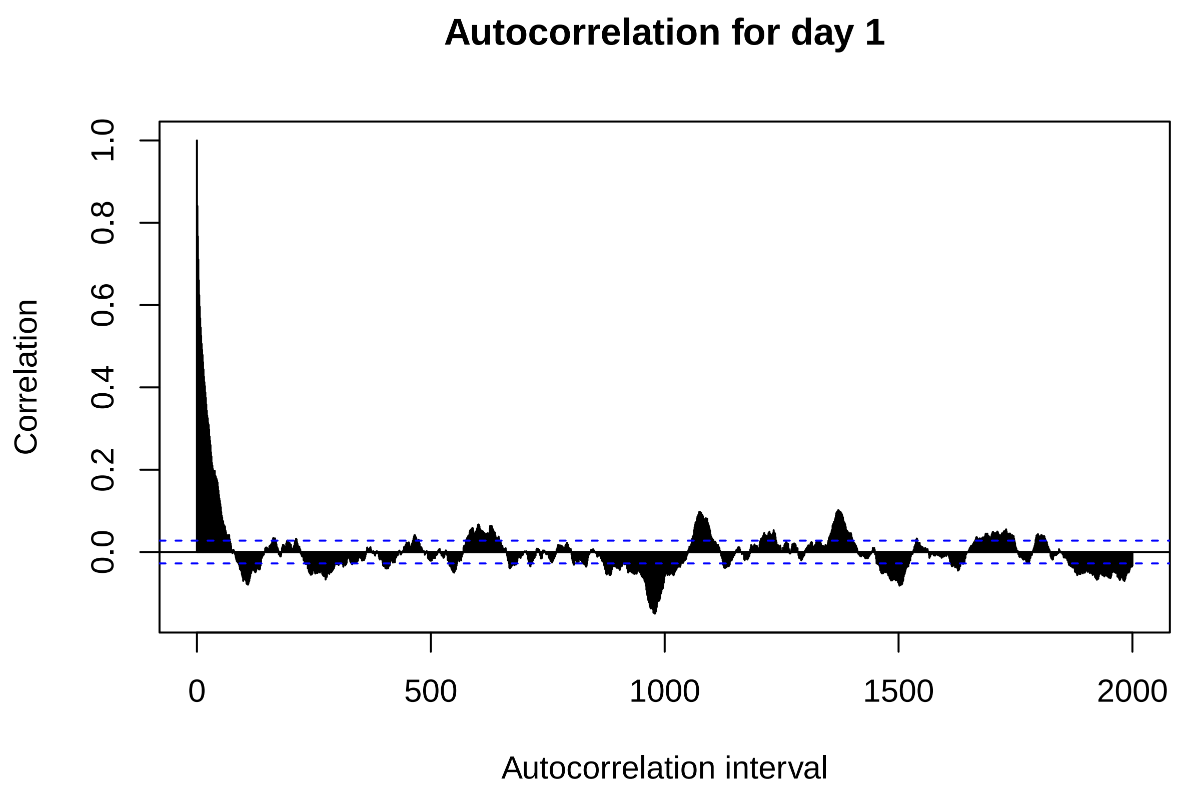
```
hist(sim_pi[, 366], main = c('Day 366 rain probability histogram'), freq = TRUE, xlab = 'Rain probabili
```



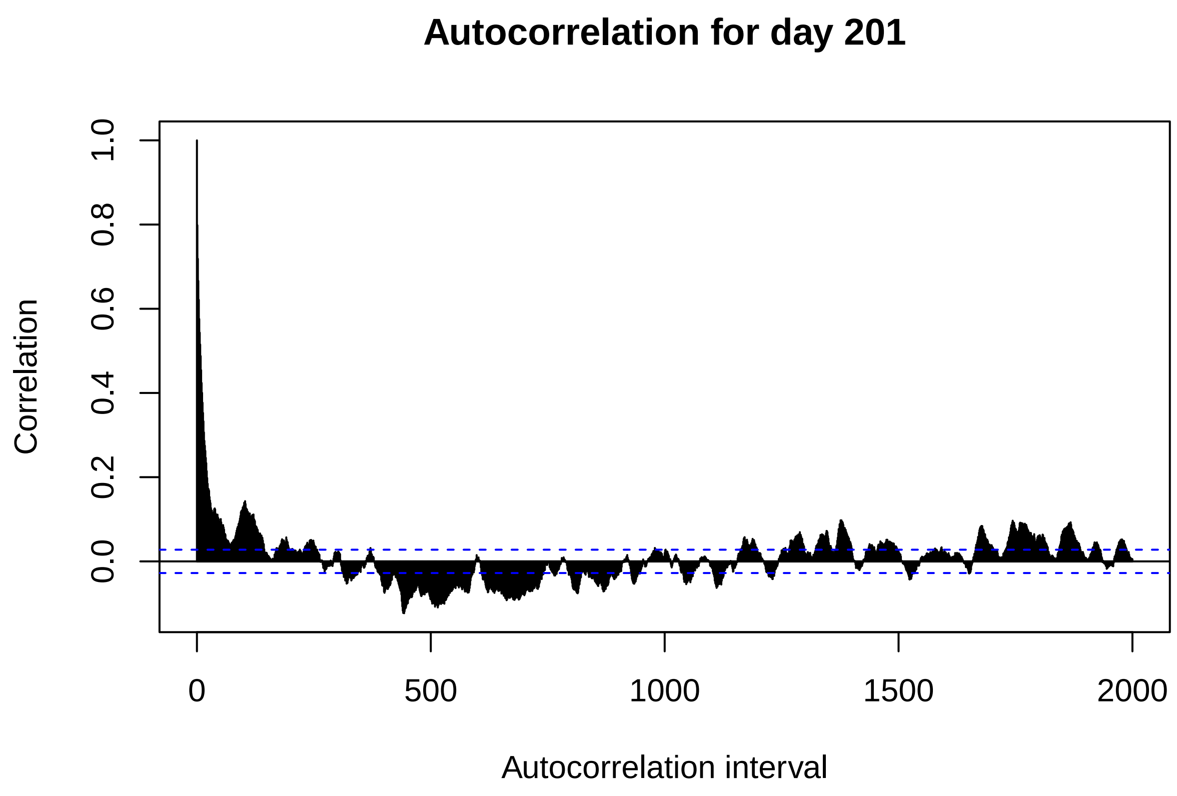
```
hist(sim_sigsq, main = 'Sigma squared histogram', freq = TRUE, xlab = 'Sigma squared', ylab = 'Frequenc
```



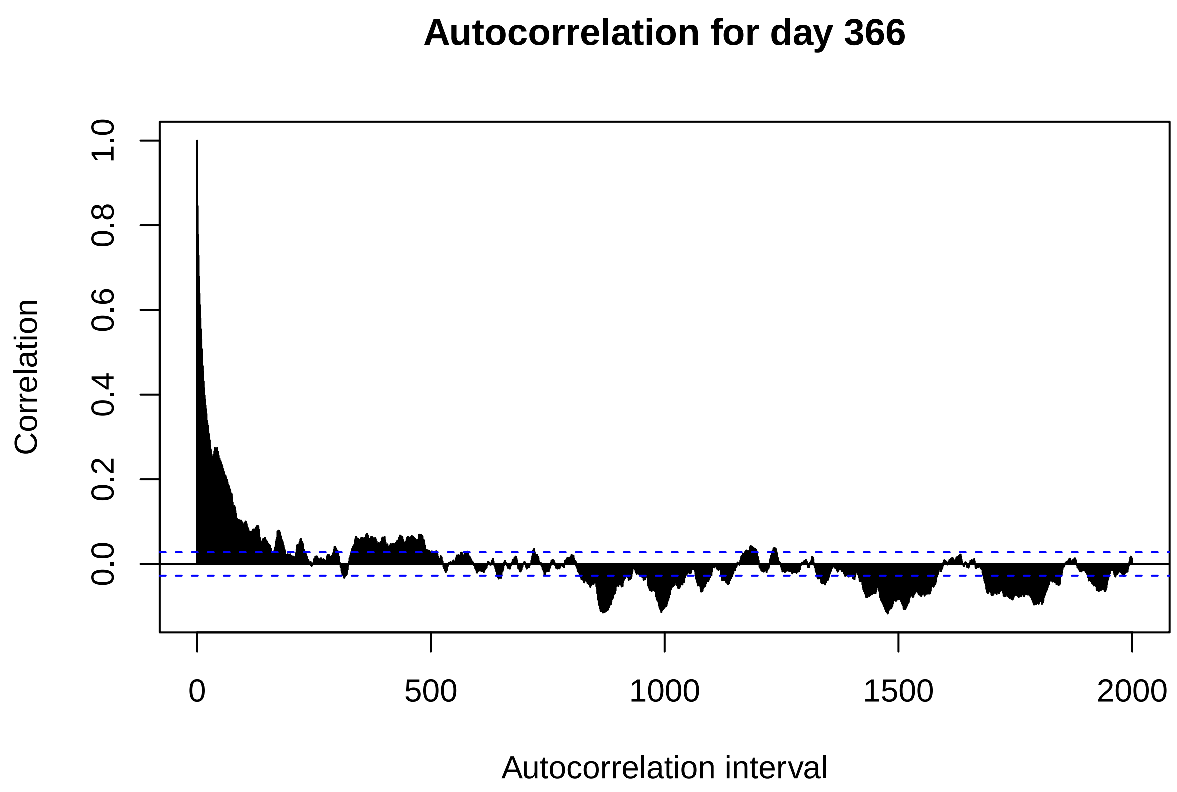
```
# Autocorrelation
acf_plot <- acf(sim_pi[, 1], lag.max = 2000, plot = FALSE)
plot(acf_plot, main = c('Autocorrelation for day 1'),  xlab = 'Autocorrelation interval', ylab = 'Correl
```



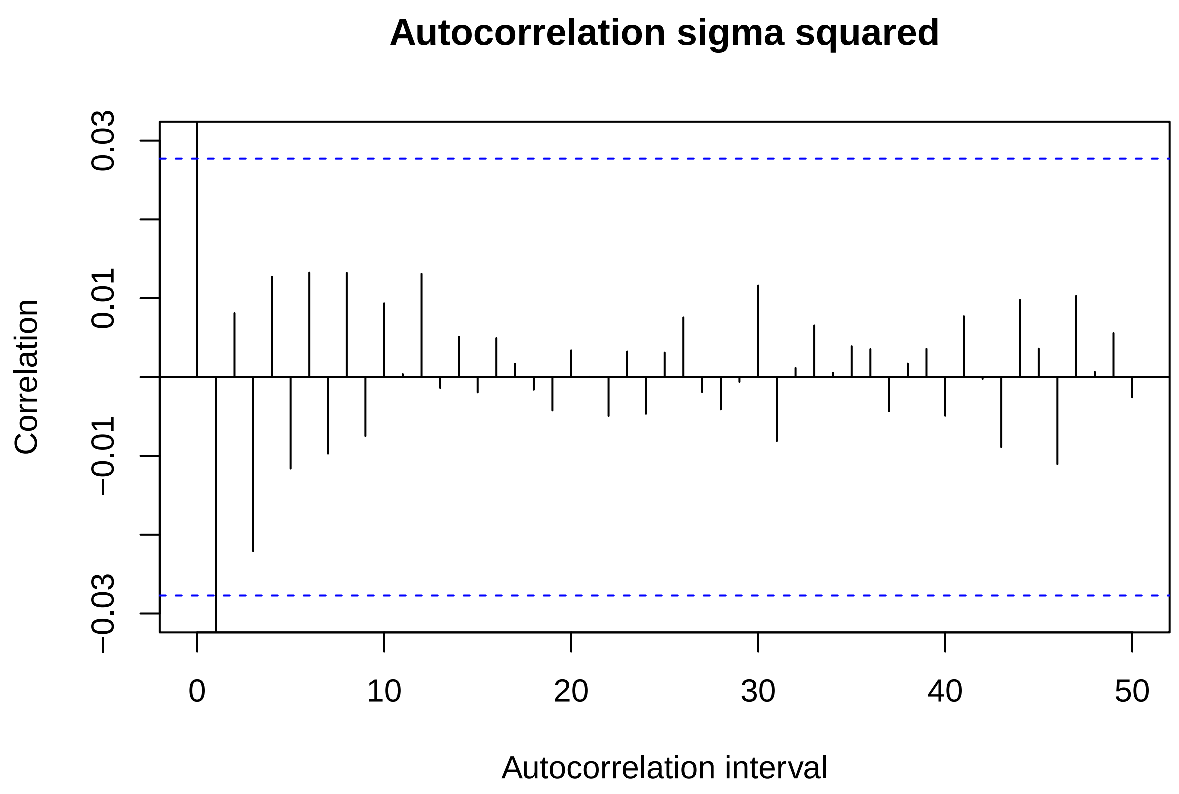
```
acf_plot <- acf(sim_pi[, 201], lag.max = 2000, plot = FALSE)
plot(acf_plot, main = c('Autocorrelation for day 201'),  xlab = 'Autocorrelation interval', ylab = 'Cor
```



```
acf_plot <- acf(sim_pi[, 366], lag.max = 2000, plot = FALSE)
plot(acf_plot, main = c('Autocorrelation for day 366'),  xlab = 'Autocorrelation interval', ylab = 'Cor
```



```
acf_plot <- acf(sim_sigsq, lag.max = 50, plot = FALSE)
plot(acf_plot, main = 'Autocorrelation sigma squared', xlab = 'Autocorrelation interval', ylab = 'Correl
```



We can set a burn-in period by looking at the plots above. The trace plots for day 1 and 366 drop fast in the first 150 iterations. Day 201 has an initial value close to its mean result and we therefore do not observe any unusual change in the beginning.

The autocorrelation plots support this burn-in period. It shows that π is correlated to its closest (approximately) 150 values.

The sigma squared converges much faster, and we see there is only a correlation to the 10 closest sigmas.

We remove the first 200 simulations to remove the effect of the burn in period.

```
# Burn in removal
sim_pi <- sim_pi[201:N, ]
sim_sigsq <- sim_sigsq[201:N]
N_new <- (N - 200)
```

```
# Means
print('Means:')
```

```
## [1] "Means:"
```

```
for (day in c(1, 201, 366)){
  print(paste('Day ', day, ': ', mean(sim_pi[, day])))
}
```

```
## [1] "Day  1 :  0.176381472569664"
## [1] "Day  201 :  0.32214191431276"
## [1] "Day  366 :  0.133603106154546"
```

```
print(paste('Sigma: ', mean(sim_sigsq)))
```

```
## [1] "Sigma:  0.00531347830802682"
```

```
# 95% credible intervals
print('95 % credible intervals')
```

```
## [1] "95 % credible intervals"
```

```
for (day in c(1, 201, 366)){
  interval <- unname(quantile(sim_pi[, day], probs = c(0.025, 0.975)))
  print(paste('Day ', day, ': ', interval[1], ', ', interval[2]))
}
```

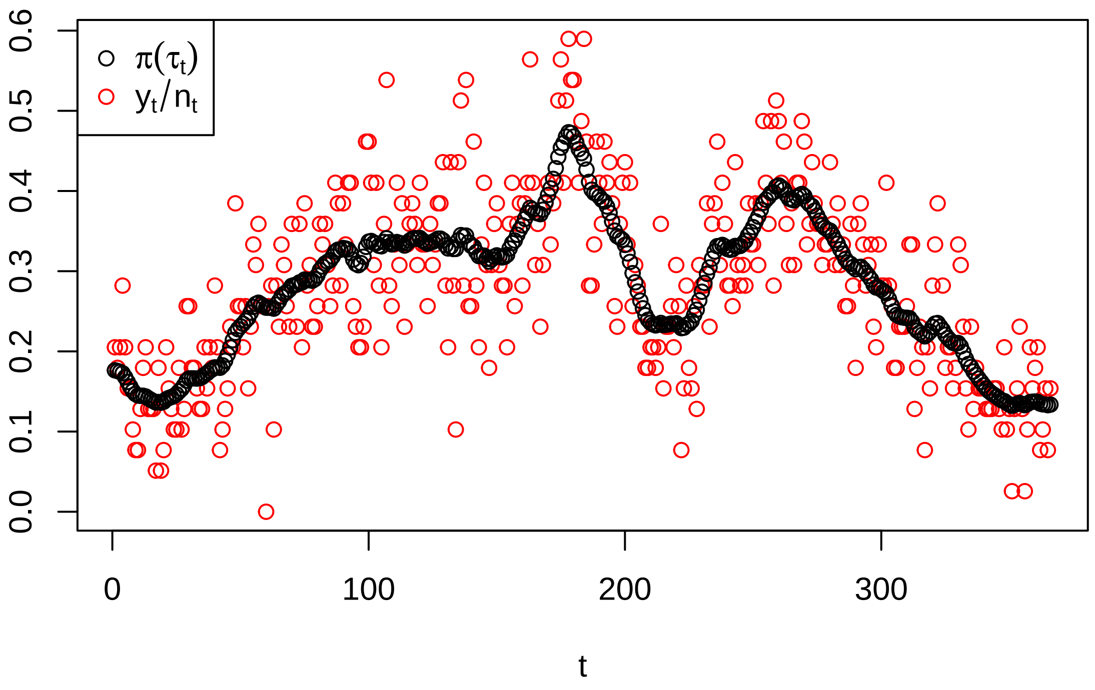
```
## [1] "Day  1 :  0.135371731317002 ,  0.23173106256678"
## [1] "Day  201 :  0.275488976754061 ,  0.371945243313497"
## [1] "Day  366 :  0.0993767529005406 ,  0.174885231482118"
```

```
interval <- unname(quantile(sim_sigsq, probs = c(0.025, 0.975)))
print(paste('Sigma: ', interval[1], ', ', interval[2]))
```

```
## [1] "Sigma:  0.00426335810136088 ,  0.00655955803337686"
```

```
# We estimate tau  to be the mean of its values after the burn-in period
tau_est <- unname(colMeans(sim_pi))

# Plot to compare tau estimate (pi) and day rain probability (y_t/n_t)
plot(rain$n.rain/rain$n.years, main = TeX(r'(Comparison of $\pi(\tau_t)$ and $y_t/n_t$)'),
     ylab='', xlab = 't', col = 'Red')
points(tau_est)

legend('topleft', legend=c(TeX(r'($\pi(\tau_t)$)'), TeX(r'($y_t/n_t$)')), col=c("black", "red"), pch=c(
```

Comparison of $\pi(\tau_t)$ and y_t/n_t



In the plot we see that y_t/n_t follows τ_t, but has a much higher variance. τ_t therefore seems to be a good prediction for rain probability in the future.

The traceplots converge quickly. The traceplots show a rapid change at the beginning (within the first 150 simulations, 10 for σ^2). Then it stabilizes at keeps a regular (random) pattern around the mean. This is a good indication for Markov chain convergence.

f)

```
# Sample from MVN using a given cholesky decomposition
dnormal <- function(my,chol_sig,n) {
  d = length(my)
  x <- matrix(rnorm(d), d, n)
  A = chol_sig
  y <- my + A%*%x
  return(y)
}
```

As in the last task we do some precomputation such that it will be faster to calculate the mean and the covariance matrix for the multivariate normal distributions, we also calculate the cholesky decomostition to make the sampling faster.

```
mcmc_block<- function(N,M){
  t0 <-proc.time()[3]
```

```
alpha <- 2
beta <- 0.05

T <- length(rain$day)

tau <- matrix(nrow = N+1, ncol = T)
tau[1,] <- rep(pi_inv(0.3), T)

#Make Q matrix
Q <- matrix(data = 0,nrow = T, ncol = T)
diag(Q) <- rep(2,T)
diag(Q[-nrow(Q),-1]) <- rep(-1,T-1)
diag(Q[-1,-ncol(Q)]) <- rep(-1,T-1)
Q[1,1] <- 1
Q[T,T] <- 1

#Find size of last block and number of blocks
M_last <- T%%M
Num_blocks <- T%/%M+1


#Precompute -QAA^-1*QAB and the Cholesky decomposition of QAA^-1
#for the first, middle, and last blocks
Q_first <- Q[1:M,1:M]
QAB_first <- c(rep(0,M-1),-1)
Q_inv_first <- inv(Q_first)
chol_first <- t(chol(Q_inv_first))
QQ_first <- -Q_inv_first%*%QAB_first

Q_mid <- Q[(M+1):(2*M),(M+1):(2*M)]
QAB_mid <- matrix(data = 0, nrow = M, ncol = 2)
QAB_mid[1,1] <- -1
QAB_mid[M,2] <- -1
Q_inv_mid <- inv(Q_mid)
chol_mid <- t(chol(Q_inv_mid))
QQ_mid <- -Q_inv_mid%*%QAB_mid

Q_last <- Q[(T-M_last+1):T,(T-M_last+1):T]
QAB_last <- c(-1,rep(0,M_last-1))
Q_inv_last <- inv(Q_last)
chol_last <- t(chol(Q_inv_last))
QQ_last <- -Q_inv_last%*%QAB_last

sigma_squared <- c(1:N+1)*0

for (i in c(1:N)){
  #Sampling sigma from the conditional found in task c
  sigma_squared[i] <- rinvgamma(n = 1, shape = alpha + (T-1)/2,
                                scale = beta + 1/2 * t(tau[i,]) %*% Q %*% tau[i,] )
  sigma <- sqrt(sigma_squared[i])

  #Sampling the taus using the precomputed data from earlier and defining
  #the interval [a,b] for this block
```

```
    for (d in c(1:Num_blocks)){
      if (d == 1){
        a <- 1
        b <- M
        new_tau_d <- dnormal(my = QQ_first*tau[i,b+1], chol_sig = sigma*chol_first, n = 1)
      }
      else if (d == Num_blocks){
        a <- T - M_last + 1
        b <- T
        new_tau_d <- dnormal(my = QQ_last*tau[i,a-1], chol_sig = sigma*chol_last, n = 1)
      }
      else{
        a <- (d-1)*M + 1
        b <- d*M
        new_tau_d <- dnormal(my = QQ_mid%*%c(tau[i,a-1],tau[i,b+1]), chol_sig = sigma*chol_mid, n = 1)
      }

      old_tau <- tau[i,]
      new_tau <- tau[i,]
      new_tau[a:b] <- new_tau_d

      #Calculating the product of the probabilities in log scale
      logprob <- 0
      for (j in c(a:b)){
        logprob = logprob + log(dbinom(rain$n.rain[j], size = rain$n.years[j],pi_func(new_tau[j]))) -
                            log(dbinom(rain$n.rain[j], size = rain$n.years[j], pi_func(old_tau[j])))
      }

      u <- runif(1)

      if (u < min(1, exp(logprob))){
        tau[i,] <- new_tau
      }
    }
    tau[(i+1),] <- tau[i,]
  }

  sigma_squared[N+1] <- sigma_squared[N]

  dt <- proc.time()[3] - t0
  #print('Processing time:')
  #print(dt)

  result <- list(pi = apply(tau, 2, pi_func), sigma_squared = sigma_squared, dt = dt)

  return(result)
}
```

We explore different values for the block size,

```
explore_M <- function(){
  M <- c(4,15,30,60,90,120,150)
  I <- 1:length(M)
  fixed_t <- I*0

  for (i in I){
    res <- mcmc_block(1,M[i])
    fixed_t[i] <- res$dt
  }
  plot(M,fixed_t, xlab = 'block size', ylab = 'time', main = 'Time for setup + precomputations')

  itr_t <- I*0

  itr <- 1000
  for (i in I){
    res <- mcmc_block(itr,M[i])
    itr_t[i] <- (res$dt-fixed_t[i])/itr
  }
  plot(M,itr_t, xlab = 'block size', ylab = 'time', main = 'Extra time per iteration')

  tot_t <- function(t0,ti,itr){
    t <- t0+ti*itr
    return(t)
  }

  itrs <- c(1:1000)*10
  plot(itrs,tot_t(fixed_t[1],itr_t[1],itrs), type = "l",
       col = 1, xlab = 'iterations', ylab = 'time', main = 'Total computation time')
  for (i in 2:length(M)){
    lines(itrs,tot_t(fixed_t[i],itr_t[i],itrs), col = i)
  }
  legend('topleft',
         legend= M,
         col= I,
         lty = 1)

}
```

```
#explore_M()
```

From having run the exporation a few times we observe quite a bit of variance when it comes to the computation times, ecpecialy for block sizes of 90, 120 and 150, which can cause some issues like negative time per iteration, or the time per iteration jumping up for one size then going back down for the next. Better method would be to find calulation times for different amounts of iterations for each block size, and do linear regresion, this would however take a very long time.

However we can still observe some trends, the the precomputation time increases exponentaly with increesing block size, but the time per iteration decreses with block size. We find that a block size of 60 hits a nice balance of low precomputation times and low time per iteration.

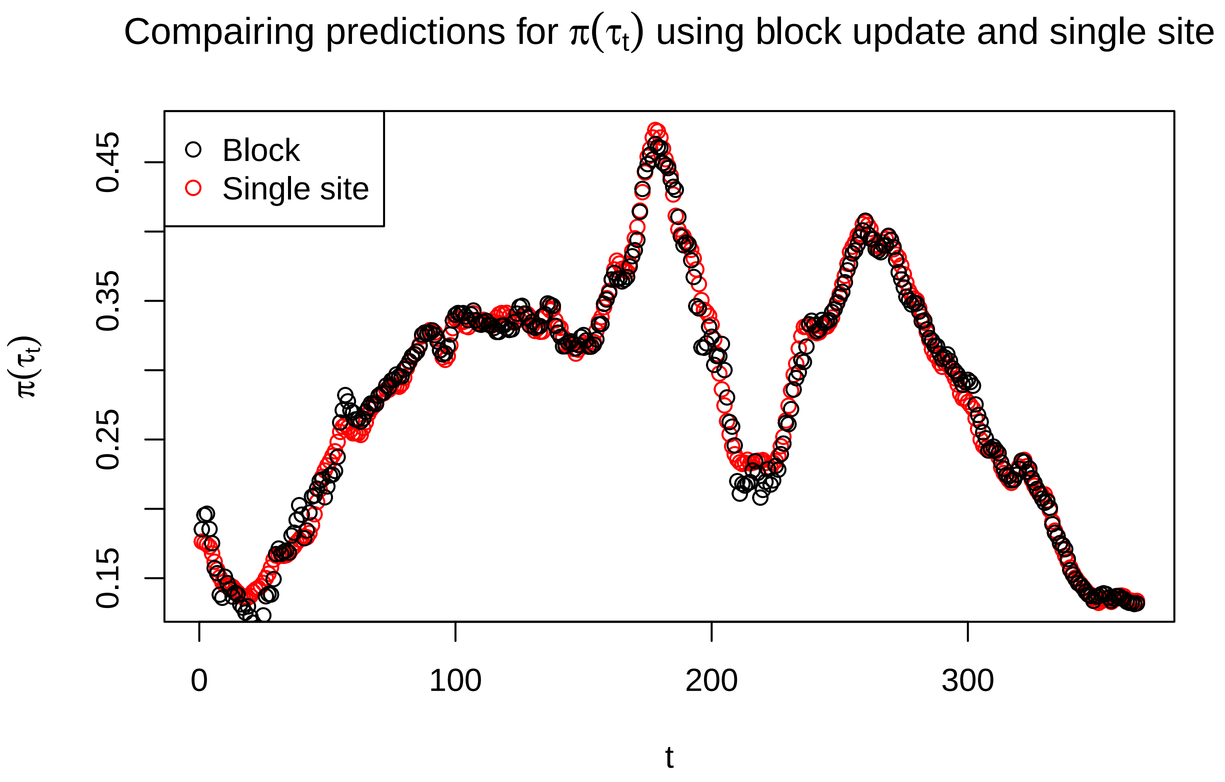
Finaly we check that the predictions for $\pi(\tau)$ are reasonable,

```
blockres <- mcmc_block(N,60)
```

```
block_mean_pi <- apply(blockres$pi[200:N,], 2, mean)
print(blockres$dt)
```

```
## elapsed
##   38.83
```

```
plot(tau_est, ylab= TeX(r'($\pi(\tau_t)$)') , xlab = 't', col = 'red',
     main = TeX(r'(Compairing predictions for $\pi(\tau_t)$ using block update and single site)'))
points(block_mean_pi)
legend('topleft', legend=c('Block','Single site'),
       col=c("black", "red"), pch=c(1,1))
```



Problem 2

a)

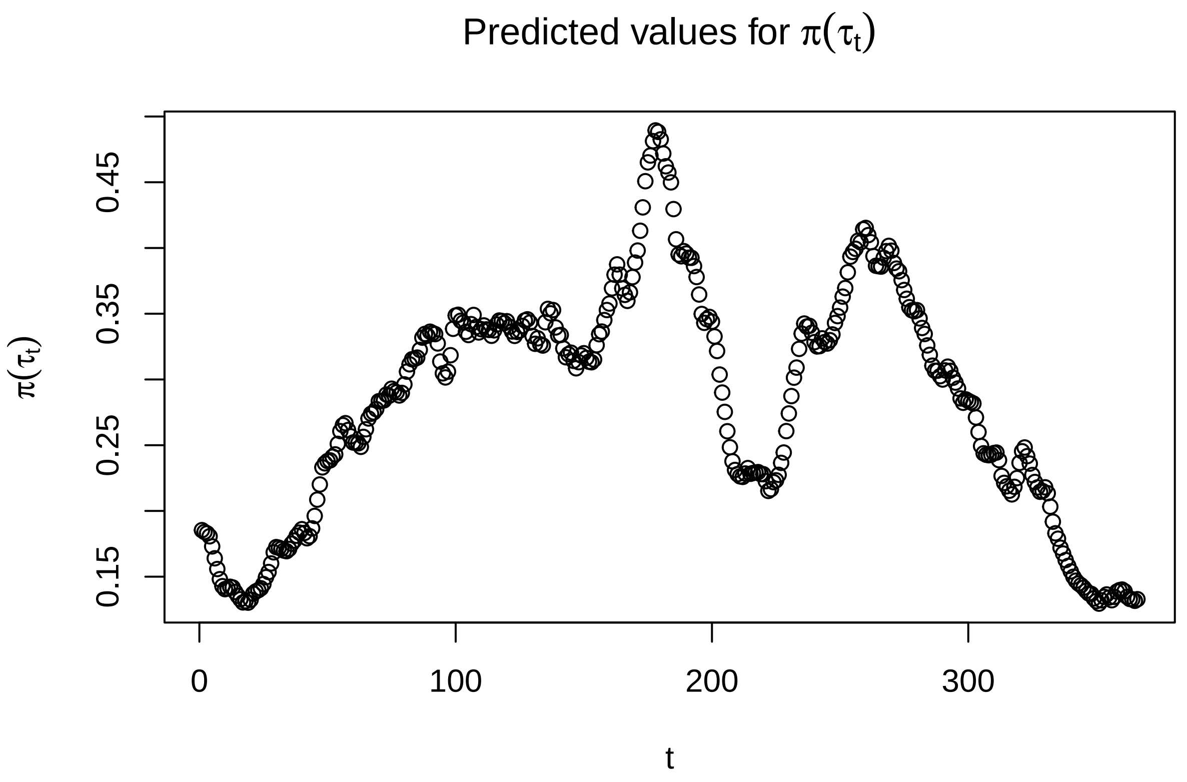
Since there isn't a bulit in inverse gamma prior (at least that we could find), we implement it ourself using "Expression", remembering to implement it on a log scale.

```
use_INLA <- function(){
  t0 <-proc.time()[3]

  loginvgamma = "expression:
                  a = 2;
                  b = 0.05;
                  precision = exp(log_precision);
                  logdens = log(b^a) - lgamma(a)
                  - (a+1)*(log_precision) - b/precision;
                  return(logdens);"

  hyper = list(prec = list(prior = loginvgamma))


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




  t <-proc.time()[3]
  print('Processing time:')
  print(t-t0)

  return(mod)
}
mod <- use_INLA()
```

```
## [1] "Processing time:"
## elapsed
##    2.37
```

We see that the computation time is far less than using Markov chains, and compairing the predictions and 95% CIs:

```
plot(mod$summary.fitted.values$mean, ylab=TeX(r'($\pi(\tau_t)$)'), xlab = 't',
     main = TeX(r'(Predicted values for $\pi(\tau_t)$)'))
```



```r
inla_print_ci <- function(mod){
  I <- c(1,201,366)
  print('INLA 95% CIs for pi(tau_t):')
  for (i in I){
    print(paste('t =',i, ': [', mod$summary.fitted.values$`0.025quant`[i], ',',
                mod$summary.fitted.values$`0.975quant`[i], ']'))
  }
}
inla_print_ci(mod)
```

```
## [1] "INLA 95% CIs for pi(tau_t):"
## [1] "t = 1 : [ 0.133565385261235 , 0.246443359807532 ]"
## [1] "t = 201 : [ 0.278600796505389 , 0.390950003885284 ]"
## [1] "t = 366 : [ 0.0911955494344203 , 0.182670065099277 ]"
```

```r
print('MCMC 95% CIs for pi(tau_t):')
```

```
## [1] "MCMC 95% CIs for pi(tau_t):"
```

```r
for (day in c(1, 201, 366)){
  interval <- unname(quantile(sim_pi[, day], probs = c(0.025, 0.975)))
  print(paste('t = ', day, ': [', interval[1], ', ', interval[2], ']'))
}
```

```
## [1] "t =  1 : [ 0.135371731317002 ,  0.23173106256678 ]"
## [1] "t =  201 : [ 0.275488976754061 ,  0.371945243313497 ]"
## [1] "t =  366 : [ 0.0993767529005406 ,  0.174885231482118 ]"
```