exercise2

Florian Beiser, Martin Lie

19.03.2021

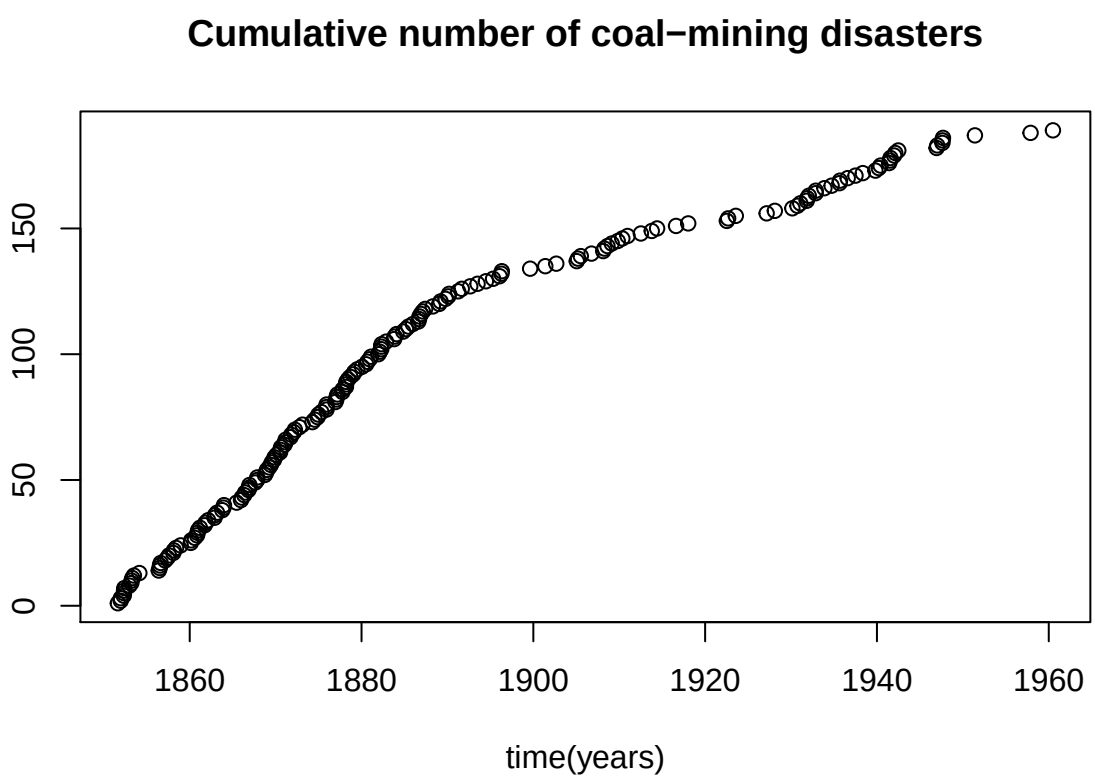
Problem A: The coal-mining disaster

In this exercise we analyse the coal-mining disasters in the UK from 1851 to 1962.

1

The dataset contains the dates of coal-mining disasters with more than 10 victims. We start with an illustration of the data.

```
# Load data
coal = as.double(as.matrix(read.table("coalmine.txt")))

# Plotting
plot(coal[-c(1,length(coal))], seq(length(coal)-2),
     main="Cumulative number of coal-mining disasters",
     xlab="time(years)", ylab="")
```



The previous figure suggests that between 1850 and 1890 disasters happend very regularily such that we have an almost linear increase in the cumulative number. After 1890 the frequency reduced, but it still seems rather regularily (somewhat linear behaviour with less steep slope).

For statistical modelling a hierachical Bayesian model with an inhomogenous Poisson process is chosen. The time interval is devided into 2 parts $[t_0, t_1]$ and $[t_1, t_2]$ where t_0 and t_2 are the start and end point of the observation interval respectively. On each interval the Poisson model follows the coefficients λ_0 and λ_1 respectively. For an observation $x = (y_1, y_2)$ were y_i are the number of disasters in interval i, the likelihood is

$$f(x|\theta) = \exp(-\lambda_0(t_1 - t_0) - \lambda_1(t_2 - t_1))\lambda_0^{y_0}\lambda_1^{y_1}.$$

The Poisson coefficients are equipped with a $Gamma(2, \beta)$ prior:

$$f(\lambda_i|\beta) = \frac{1}{\beta^2}\lambda_i \exp(\frac{\lambda_i}{\beta})$$

Last the hyperparameter β get the improper prior

$$f(\beta) \propto \frac{1}{\beta}\exp(-\frac{1}{\beta})$$

For notational convenience all parameters are collected together $\theta = (t_1, \lambda_0, \lambda_1, \beta)$.

2

The posterior is calculated Bayesian formula:

$$f(\theta|x) \propto f(x|\theta)f(\lambda_0|\beta)f(\lambda_1|\beta)f(t_1)f(\beta)$$
$$= \frac{1}{\beta^5}\lambda_0^{y_0+1}\lambda_1^{y_1+1}\exp(\lambda_0 t_0 - \lambda_0 t_1 - \frac{\lambda_0}{\beta} + \lambda_1 t_1 - \frac{\lambda_1}{\beta} - \lambda_1 t_2 - \frac{1}{\beta})1_{[t_0,t_2]}(t_1)$$

3

The full conditionals are derived from the posterior by omitting all multiplicative factors that do not depend on the parameter of interest (remember that $f(\theta_i|x,\theta_{-i}) \propto f(\theta|x)$):

$$f(t_1|x,\theta_{-t_1}) \propto \lambda_0^{y_0}\lambda_1^{y_1}\exp(-(\lambda_0-\lambda_1)t_1)1_{[t_0,t_2]}(t_1) \sim \texttt{"unnamed"}$$
$$f(\lambda_i|x,\theta_{-\lambda_i}) \propto \lambda_i^{y_i+1}\exp(-\lambda_i(t_{i+1}-t_i+\frac{1}{\beta})) \sim Gamma(y_i+2, t_{i+1}-t_i+\frac{1}{\beta}))$$

$$f(\beta|x,\theta_{-\beta}) \propto \beta^{-5}\exp(-\frac{(\lambda_0+\lambda_1+1)}{\beta}) \sim IG(4, \lambda_0+\lambda_1+1)$$

4

We apply a single-site MCMC to sample from the posterior distribution

- Update β by a Gibbs step
- Update λ_0 by a Gibbs step
- Update λ_1 by a Gibbs step
- Propose a new t^* using a normal distribution with variance σ^2 in a random walk MCMC

The acceptance probability for the random walk step is $\alpha = \min(1, \frac{f(\theta^*|x)}{f(\theta|x)}) = \min(1, \frac{f(t^*|x,\theta_{-t})}{f(t|x,\theta_{-t})})$ where $\theta^* = (\beta, \lambda_0, \lambda_1, t_1^*)$ and where we calculate the latter quotient in log-scale.

```
library(invgamma)
# From data
t0 = coal[1]
t2 = coal[length(coal)]
x = coal[-c(1,length(coal))]

# MCMC
# generating a chain by using aboves single site scheme

# Arguments:
# N: number of steps (including burn-in)
# sigma: tuning parameter for the RW site

MCMC <- function(N, sigma){
  # Initialisation
  beta = 0.1
  lambda0 = rgamma(1, shape=2, scale=beta)
  lambda1 = rgamma(1, shape=2, scale=beta)
  t1 = t0+(t2-t0)/2

  # Bookkeeping
  T1 = rep(t1,N)
  Beta = rep(beta,N)
  Lambda0 = rep(lambda0,N)
  Lambda1 = rep(lambda1,N)
```

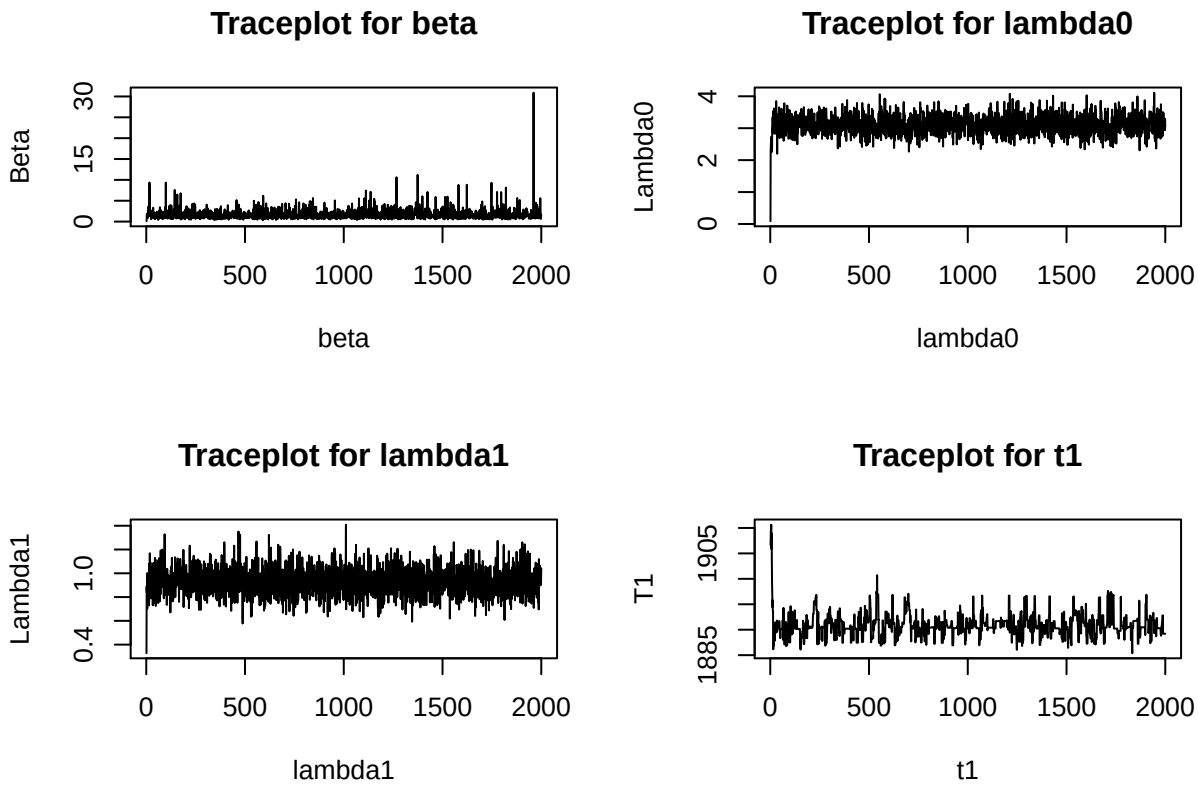
```
  # MCMC Loop
  for (i in 2:N){
    # split x
    y0 = length(x[x<t1])
    y1 = length(x)-y0

    # Update beta
    beta = rinvgamma(1, shape=4, lambda0+lambda1+1)
    Beta[i] = beta

    # Update lambda0
    lambda0 = rgamma(1, shape=y0+2, rate=-(t0-t1-1/beta))
    Lambda0[i] = lambda0

    # Update lambda1
    lambda1 = rgamma(1, shape=y1+2, rate=-(t1-t2-1/beta))
    Lambda1[i] = lambda1

    # Propose t1
    t1_prop = min(max(t0,rnorm(1, mean=t1, sd=sigma)),t2)
    y0_prop = length(x[x<t1_prop])
    y1_prop = length(x)-y0_prop

    ldens_new = y0_prop*log(lambda0) + y1_prop*log(lambda1) -(lambda0-lambda1)*t1_prop
    ldens_old = y0*log(lambda0) + y1*log(lambda1) -(lambda0-lambda1)*t1

    # Accetance or rejection of beta proposal
    alpha = min(1,exp(ldens_new-ldens_old))
    if (runif(1) < alpha) {
      t1 = t1_prop
    }
    T1[i] = t1
  }

  return(list(t1=T1,lambda0=Lambda0,lambda1=Lambda1,beta=Beta))
}
```

5

We now build Markov chains using the single-site algorithm from above and evaluate its performance. For the length of the burn-in period we inspect trace plots and for the mixing we observe the autocorrelation (there are further criteria like ESS and Geweke diagnostics, but we focus on the previously mentioned, since they are very visual).

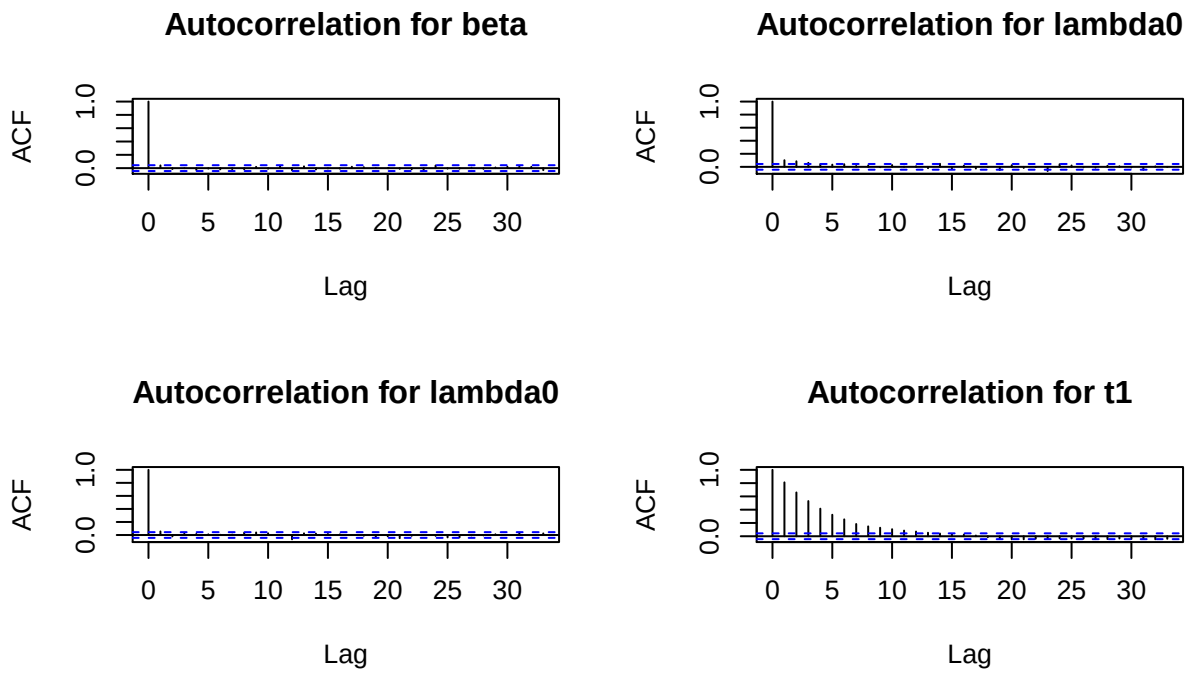
```
# Input
sigma = 3.5
N = 2000

# Extract chains from function
chains = MCMC(N, sigma)

# Arguments:
# chains: as output of MCMC function
```

```r
# Function:
# generating traceplots for all parameters
tracePlots <- function(chains, cutoff=length(chains$beta)){
  par(mfrow=c(2,2))
  # Single chains
  Beta = chains$beta
  Lambda0 = chains$lambda0
  Lambda1 = chains$lambda1
  T1 = chains$t1

  plot(Beta, type="l",
      main="Traceplot for beta", xlab="beta", xlim=c(1,cutoff))
  plot(Lambda0, type="l",
       main="Traceplot for lambda0", xlab="lambda0", xlim=c(1,cutoff))
  plot(Lambda1, type="l",
       main="Traceplot for lambda1", xlab="lambda1", xlim=c(1,cutoff))
  plot(T1, type="l",
       main="Traceplot for t1", xlab="t1", xlim=c(1,cutoff))
}

tracePlots(chains)
```




```r
# Arguments:
# chains as by MCMC fucntion

# Function:
```

```
# Generating autocorrelation
acfPlots <- function(chains){
  par(mfrow=c(2,2))
  par(oma=c(2,0,2,0))
  # Single chains
  Beta = chains$beta
  Lambda0 = chains$lambda0
  Lambda1 = chains$lambda1
  T1 = chains$t1

  acf(Beta, main="Autocorrelation for beta")
  acf(Lambda0, main="Autocorrelation for lambda0")
  acf(Lambda1, main="Autocorrelation for lambda0")
  acf(T1, main="Autocorrelation for t1")
}

acfPlots(chains)
```




```
# Reasonability of solution
t1_mean = mean(chains$t1[-c(1:100)])
x1 = x[x<t1_mean]
x2 = x[x>=t1_mean]
l0 = length(x1)/(x1[length(x1)]-x1[1])
l1 = length(x2)/(x2[length(x2)]-x2[1])

print(paste("From the MCMC we assume a split at ",t1_mean))
```

```
## [1] "From the MCMC we assume a split at  1890.90803567379"
```

```
print(paste("From the data we estimate lambda0 for aboves t1 as ", l0))
```

```
## [1] "From the data we estimate lambda0 for aboves t1 as  3.21600511254698"
```

```
print(paste("From the data we estimate lambda1 for aboves t1 as ", l1))
```

```
## [1] "From the data we estimate lambda1 for aboves t1 as  0.938797500889711"
```

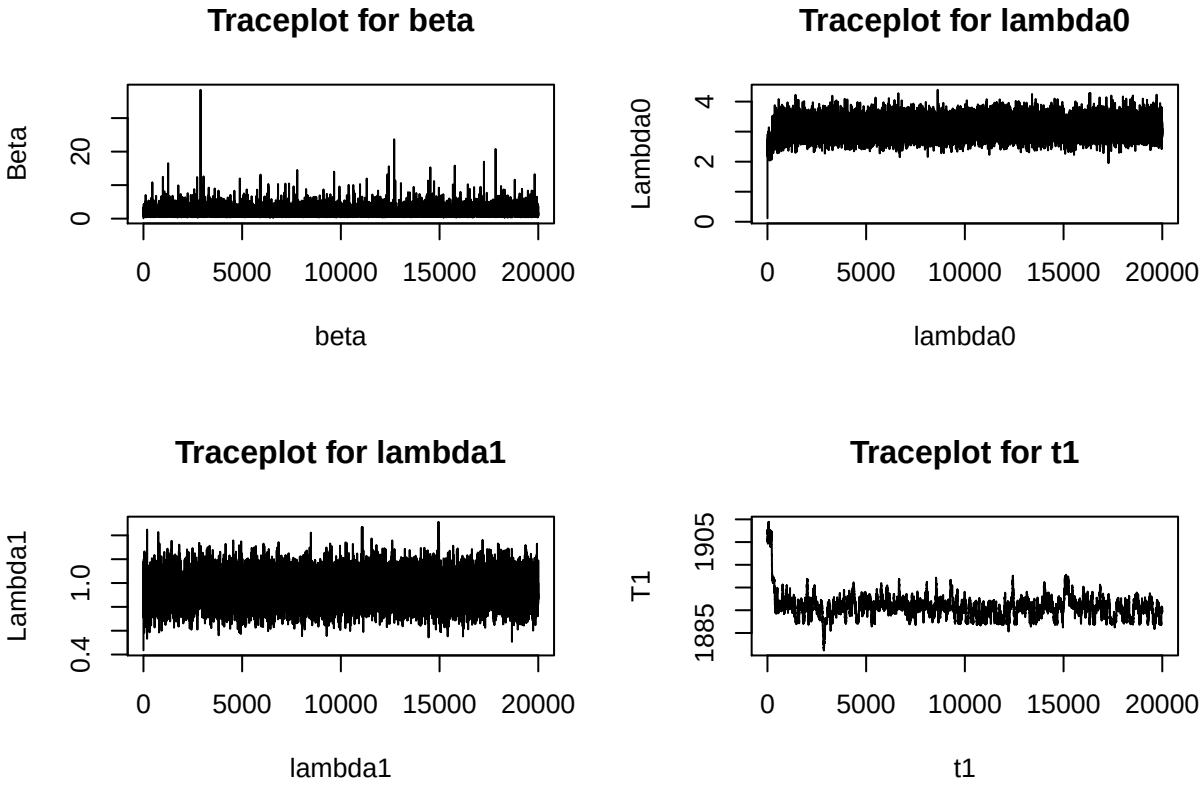
The trace plots suggest that the samples represent the target distribution after around 100 steps rather well and we cannot observe further changes of the distribution such that we choose the burn-in period of 100 steps. Further, the autocorrelation plots prove that the chains for $\beta, \lambda_0, \lambda_1$ mix very well as we expect it for Gibbs steps. For t_1 the mixing is not as good, since several steps get rejected and the chain pauses in a state for several steps.

Moreover, the distribution make a lot of sense to us since we already guessed that t_1 would be around 1890 (see sub-part 1) and in the Bayesian setting the distribution for t_1 which was generated with MCMC coincides with this intuition. Ad parameter, we can estimate the disasters per year before and after the cut at t_1, i.e. λ_0 and λ_1 from the data. For the parameter in the first interval we estimate around 3 disasters per year and this is again remodelled in the Bayesian setting of the MCMC with λ_0 having the mass of its distribution little above 3. Similarly, we observe a bit less than 1 disaster per year and λ_1 has it mass a bit below 1, such that this again coincides and we value our results as very reasonable.

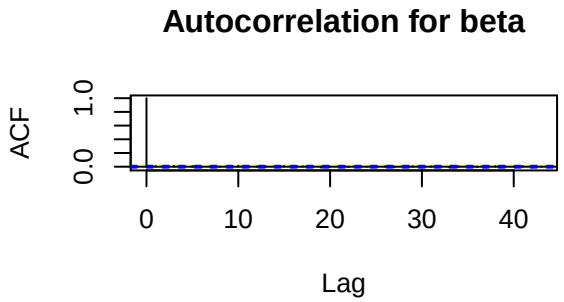
6

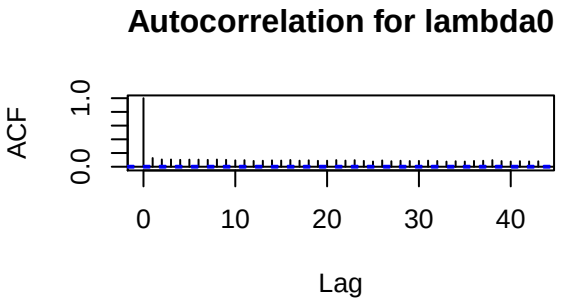
The random walk proposal step in the MCMC algorithms uses the tuning parameter σ for the standard deviation in the normal distribution. Technically the initial states can also seen as tuning parameter, but we keep them constant here to be able to compare the burn-in period with respect to one parameter and avoiding Pareto-fronts which distract our attention from the core of the tuning.

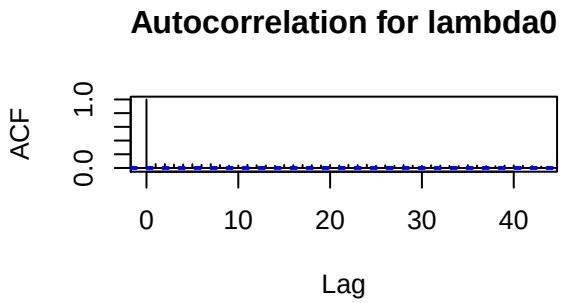
```
chains1 = MCMC(20000, sigma=0.5)
tracePlots(chains1)
```

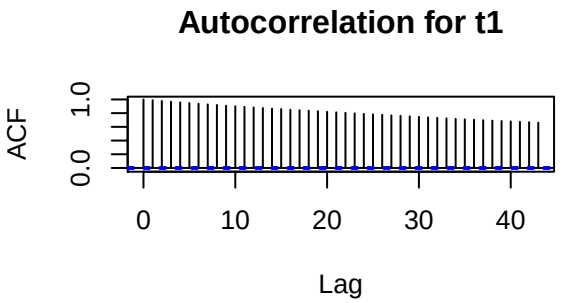



```
acfPlots(chains1)
```

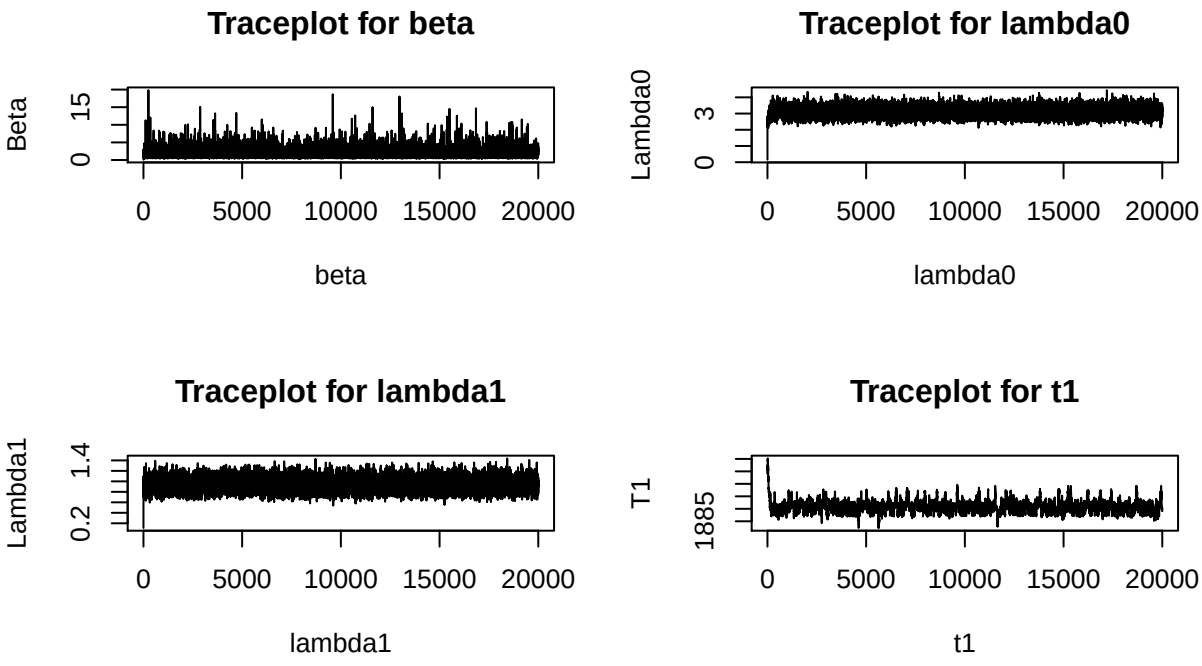




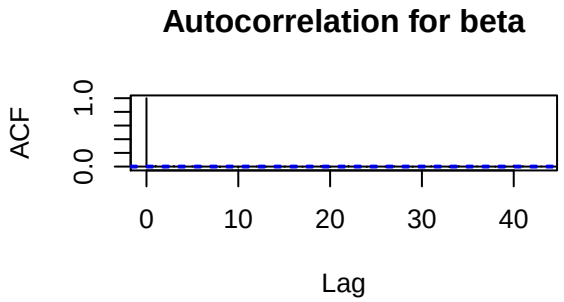


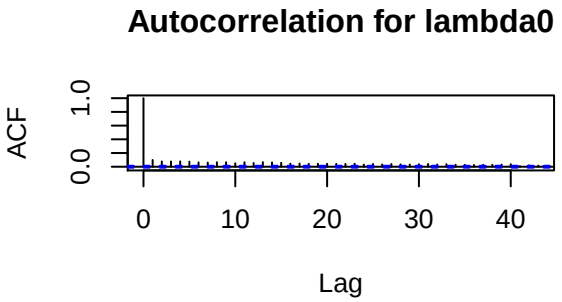


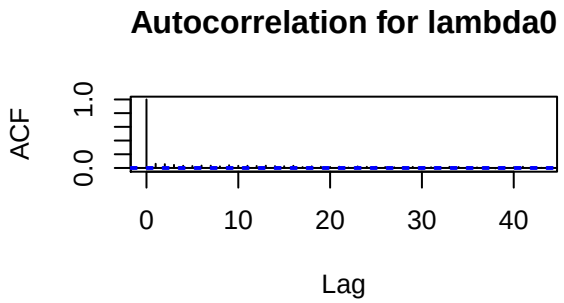

```
chains2 = MCMC(20000, sigma=1)
tracePlots(chains2)
```

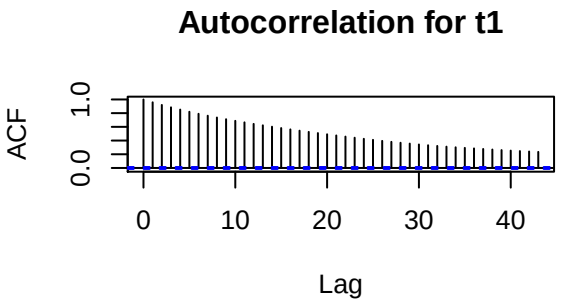



```
acfPlots(chains2)
```

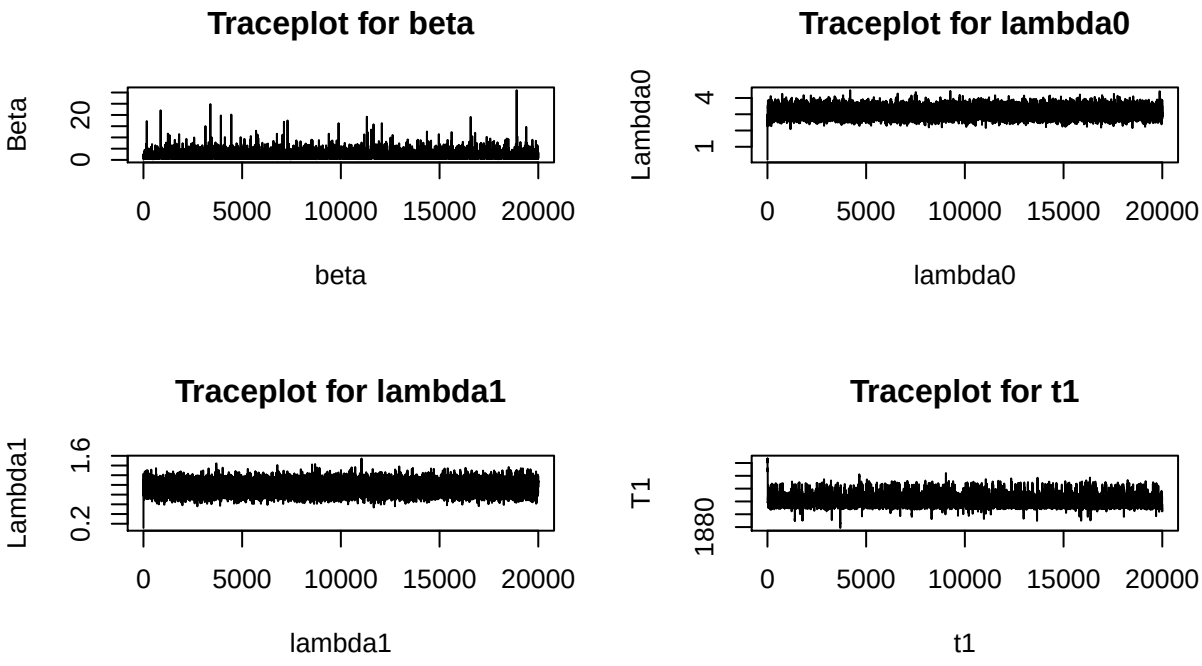




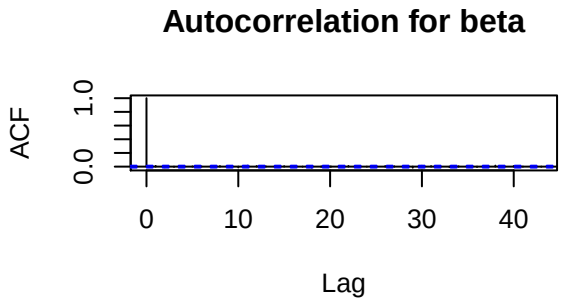


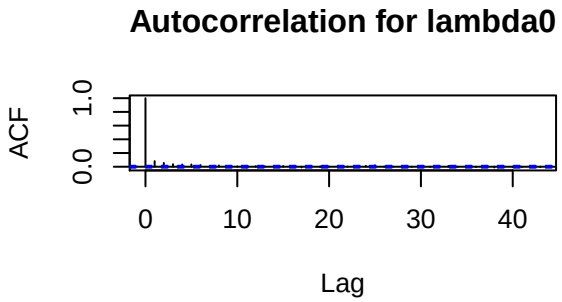


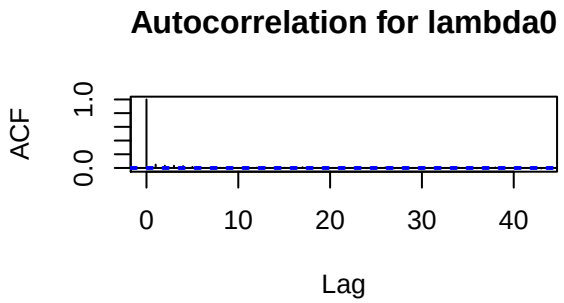

```
chains3 = MCMC(20000, sigma=3.5)
tracePlots(chains3)
```

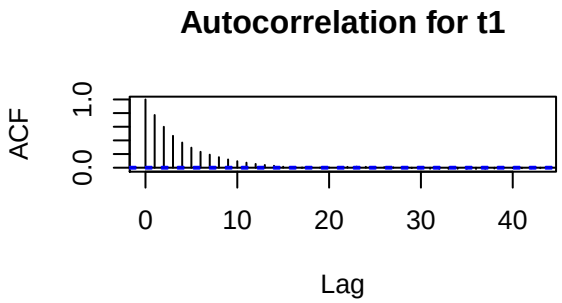



```
acfPlots(chains3)
```

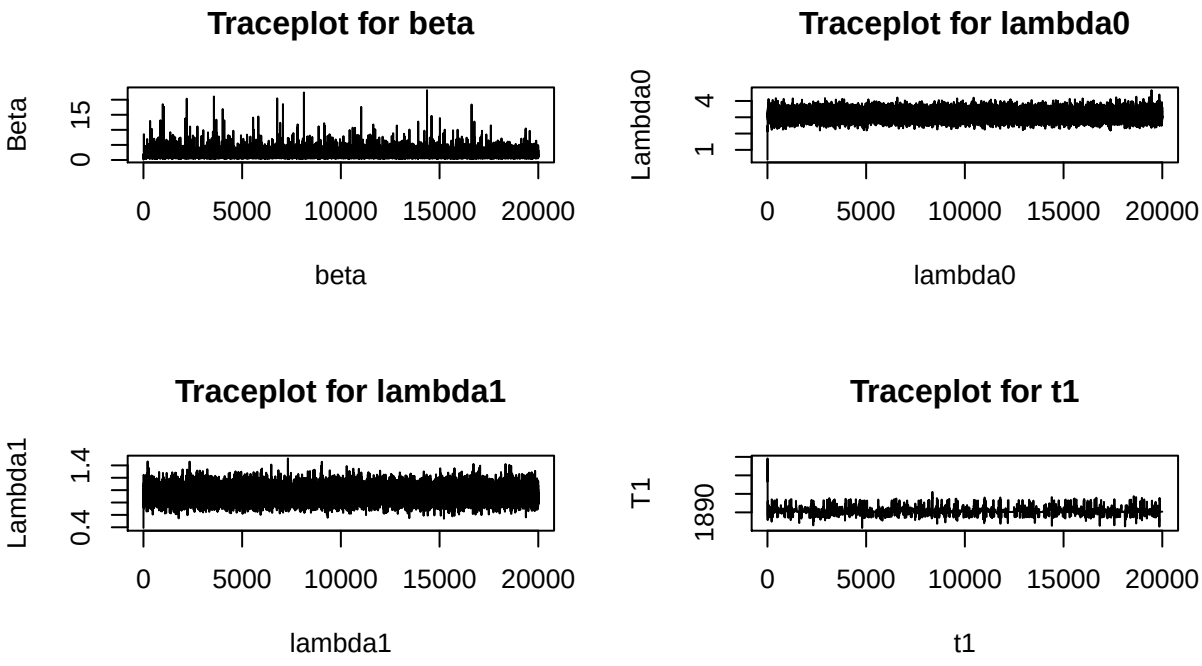




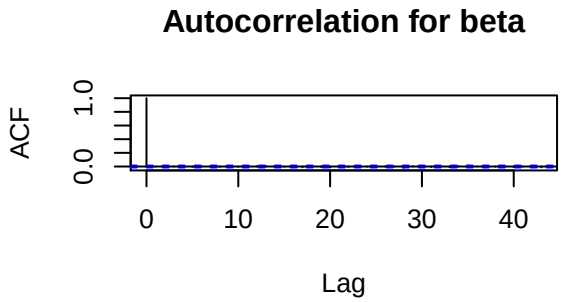


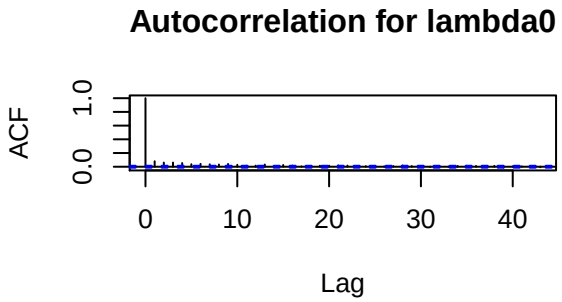


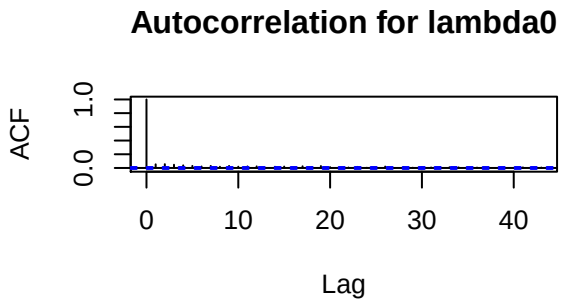

```
chains4 = MCMC(20000, sigma=25.0)
tracePlots(chains4)
```

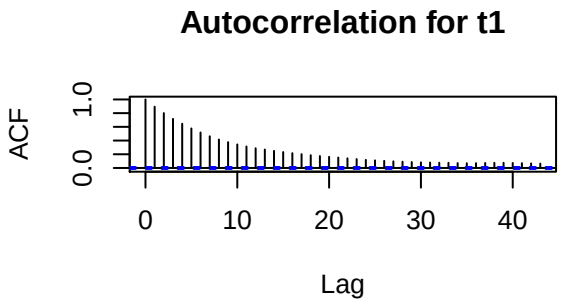



```
acfPlots(chains4)
```

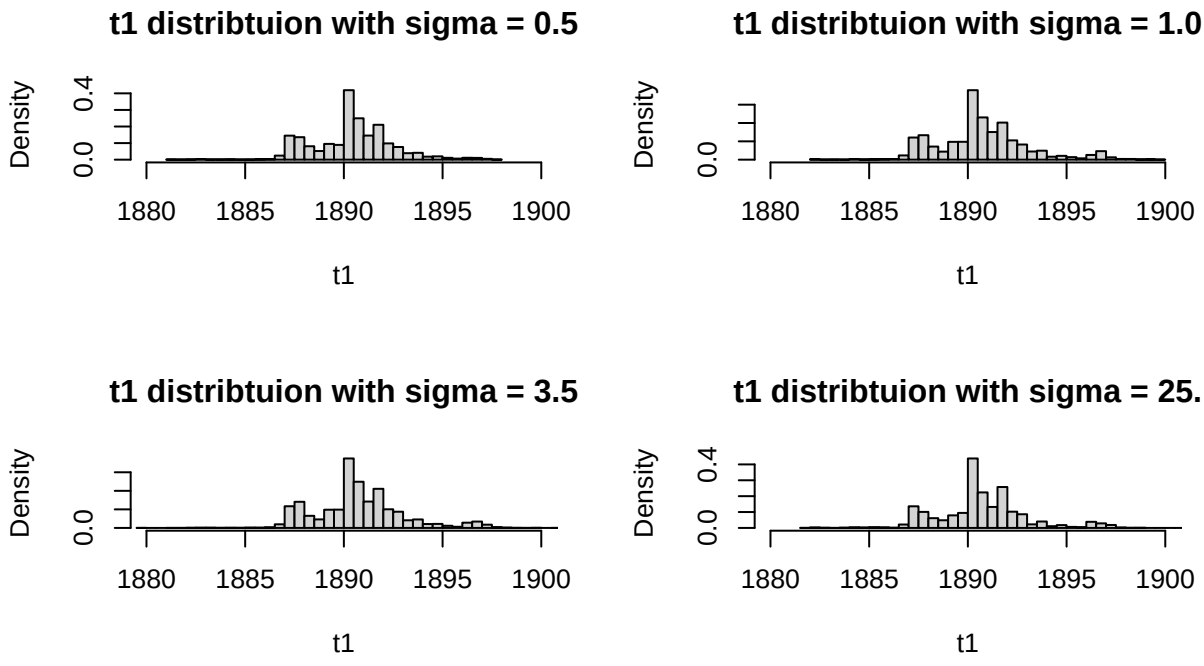









```
hist(chains1$t1[-c(1:1000)], freq=F, breaks=50,
     main="t1 distribtuion with sigma = 0.5",
     xlab="t1", xlim=c(1880,1900))
hist(chains2$t1[-c(1:1000)], freq=F, breaks=50,
     main="t1 distribtuion with sigma = 1.0",
     xlab="t1", xlim=c(1880,1900))
hist(chains3$t1[-c(1:1000)], freq=F, breaks=50,
     main="t1 distribtuion with sigma = 3.5",
     xlab="t1", xlim=c(1880,1900))
hist(chains4$t1[-c(1:1000)], freq=F, breaks=50,
     main="t1 distribtuion with sigma = 25.",
     xlab="t1", xlim=c(1880,1900))
```



Since the distributions for $\beta, \lambda_0, \lambda_1$ are generated with Gibbs sampling which does not invovle the tuning parameter, we pay here attention to the chain of t_1. For too small choices of σ the chain moves very slow, the space is explored with a very high autocorrelation for many lags and we have a long burn-in phase. The burn-in is shorter and the mixing better for moderate values of $\sigma \in [2, 5]$. Whereas for ridiculous large values values of σ the burn-in stays short, but the chain pauses too long since many steps get rejected and autocorrelation increases again. Note that this discussion addresses the performance of the MCMC in dependence of the tuning parameter. Qualitatively we can see that all chains converge to the same distribution independent of the choice of the tuning parameter! (See histograms)

7

After the single-site MCMC algorithm, we draw our attention towards a block update within the MCMC routine. Every other MCMC iteration we use one of the following blocking strategies:

1. Propose t_1^* from a random walk proposal with standard deviation σ_t and then propose $(\lambda_0^*, \lambda_1^*)$ from $f(\lambda_0, \lambda_1 | x, t_1^*, \beta)$. Keep β unchanged.
2. Propose β^* from a random walk proposal with standard deviation σ_β and then propose $(\lambda_0^*, \lambda_1^*)$ from $f(\lambda_0, \lambda_1 | x, t_1, \beta^*)$. Keep t_1 unchanged.

Remember that λ_i are jointly independent of each other by assumption, hence the joint distribution factorizes and we resamble the same Gamma distribution individually with the proposed t_1^* in the parameters. The acceptance probabilities then become

1. $\alpha_1 = \min(1, \frac{f(\theta^*|x)}{f(\theta|x)}) = \min(1, \frac{f(t^*, \lambda_0^*, \lambda_1^*|x, \beta)}{f(t, \lambda_0, \lambda_1|x, \beta)})$
2. $\alpha_2 = \min(1, \frac{f(\theta^*|x)}{f(\theta|x)}) = \min(1, \frac{f(\lambda_0^*, \lambda_1^*, \beta^*|x, t_1)}{f(\lambda_0, \lambda_1, \beta|x, t_1)})$

where again

1. $f(t, \lambda_0, \lambda_1 | x, \beta) = \lambda_0^{y_0+1} \lambda_1^{y_1+1} \exp(-\lambda_0(t_1 - t_0) - \lambda_1(t_2 - t_1) - \frac{1}{\beta}(\lambda_0 + \lambda_1))$
2. $f(\lambda_0, \lambda_1, \beta | x, t_1) = \frac{1}{\beta^5} \lambda_0^{y_0+1} \lambda_1^{y_1+1} \exp(-\lambda_0(t_1 - t_0) - \lambda_1(t_2 - t_1) - \frac{1}{\beta}(\lambda_0 + \lambda_1 + 1))$

and we calculate ratios in log-scale.

```
# From data
t0 = coal[1]
t2 = coal[length(coal)]
x = coal[-c(1,length(coal))]

# MCMC
# generating a chain by using aboves single site scheme

# Arguments:
# N: number of steps (including burn-in)
# sigma: tuning parameter for the RW site

blockMCMC <- function(N, sigma_t, sigma_b){
  # Initialisation
  beta = 0.1
  lambda0 = rgamma(1, shape=2, scale=beta)
  lambda1 = rgamma(1, shape=2, scale=beta)
  t1 = t0+(t2-t0)/2

  # Bookkeeping
  T1 = rep(t1,N)
  Beta = rep(beta,N)
  Lambda0 = rep(lambda0,N)
  Lambda1 = rep(lambda1,N)

  # MCMC Loop
  for (i in 2:N){

    ##############################
    # block 1
    if (i%%2==0){
      # split x
      y0 = length(x[x<t1])
      y1 = length(x)-y0

      # Keep beta
      Beta[i] = beta

      # Propose t1
      t1_prop = min(max(t0,rnorm(1, mean=t1, sd=sigma_t)),t2)
      y0_prop = length(x[x<t1_prop])
      y1_prop = length(x)-y0_prop

      # Propose lambda0
      lambda0_prop = rgamma(1, shape=y0_prop+2, rate=-(t0-t1_prop-1/beta))

      # Propose lambda1
      lambda1_prop = rgamma(1, shape=y1_prop+2, rate=-(t1_prop-t2-1/beta))

      # Evaluation of target densities
```

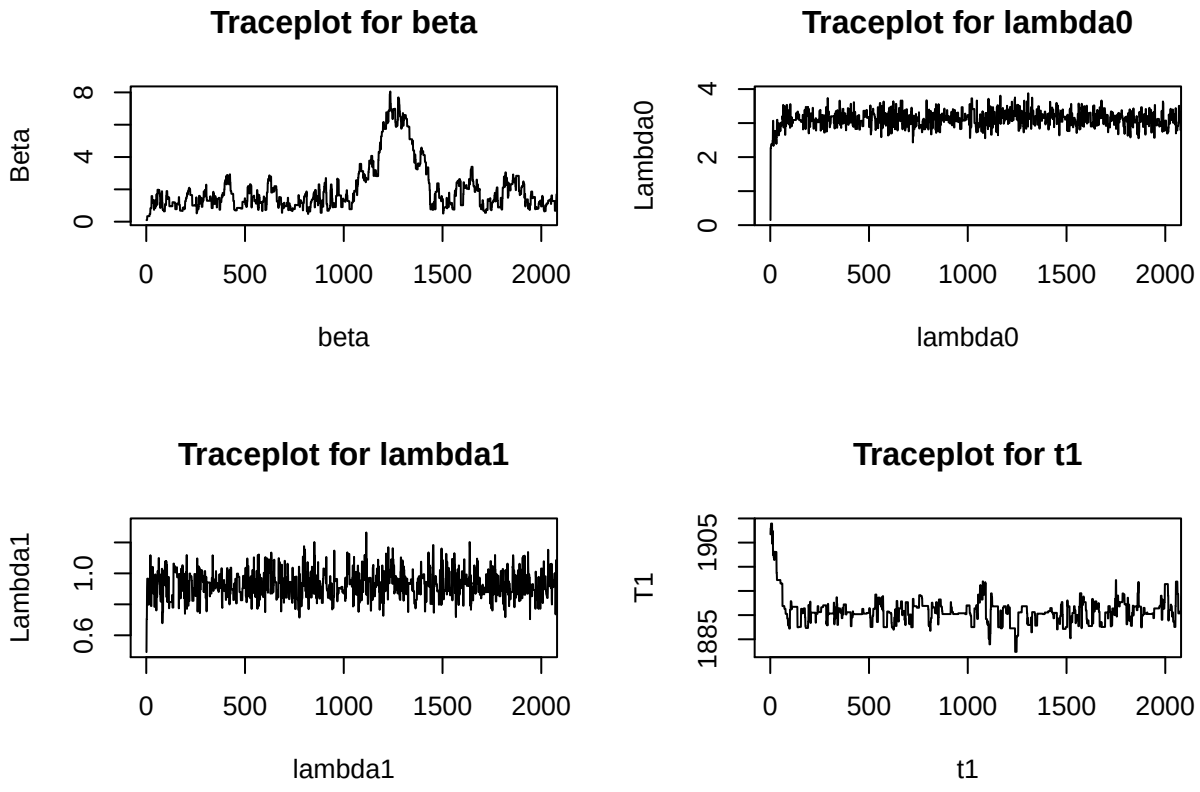
```
    ldens_new = (y0_prop+1)*log(lambda0_prop) + (y1_prop+1)*log(lambda1_prop)
    ldens_new = ldens_new - lambda0_prop*(t1_prop-t0) - lambda1_prop*(t2-t1_prop)
    ldens_new = ldens_new - (lambda0_prop+lambda1_prop)/beta

    ldens_old = (y0+1)*log(lambda0) + (y1+1)*log(lambda1)
    ldens_old = ldens_old - lambda0*(t1-t0) - lambda1*(t2-t1)
    ldens_old = ldens_old - (lambda0+lambda1)/beta

    # Accetance or rejection of beta proposal
    alpha = min(1,exp(ldens_new-ldens_old))
    if (runif(1) < alpha) {
      t1 = t1_prop
      lambda0 = lambda0_prop
      lambda1 = lambda1_prop
    }
    # Storing new values
    T1[i] = t1
    Lambda0[i] = lambda0
    Lambda1[i] = lambda1
  }

  ##############################
  # Block 2
  else{
    # split x
    y0 = length(x[x<t1])
    y1 = length(x)-y0

    # Keep t1
    T1[i] = t1

    # Propose beta
    beta_prop = max(1e-10, rnorm(1, mean=beta, sd=sigma_b))

    # Popose lambda0
    lambda0_prop = rgamma(1, shape=y0+2, rate=-(t0-t1-1/beta_prop))

    # Propose lambda1
    lambda1_prop = rgamma(1, shape=y1+2, rate=-(t1-t2-1/beta_prop))

    # Evaluation of target densities
    ldens_new = -5*log(beta_prop) + (y0+1)*log(lambda0_prop) + (y1+1)*log(lambda1_prop)
    ldens_new = ldens_new - lambda0_prop*(t1-t0) - lambda1_prop*(t2-t1)
    ldens_new = ldens_new - (lambda0_prop+lambda1_prop+1)/beta_prop

    ldens_old = -5*log(beta) + (y0+1)*log(lambda0) + (y1+1)*log(lambda1)
    ldens_old = ldens_old - lambda0*(t1-t0) - lambda1*(t2-t1)
    ldens_old = ldens_old - (lambda0+lambda1+1)/beta

    # Accetance or rejection of beta proposal
    alpha = min(1,exp(ldens_new-ldens_old))
    if (runif(1) < alpha) {
      beta = beta_prop
```

```
        lambda0 = lambda0_prop
        lambda1 = lambda1_prop
      }
      Beta[i] = beta
      Lambda0[i] = lambda0
      Lambda1[i] = lambda1
    }
  }

  return(list(t1=T1,lambda0=Lambda0,lambda1=Lambda1,beta=Beta))
}
```

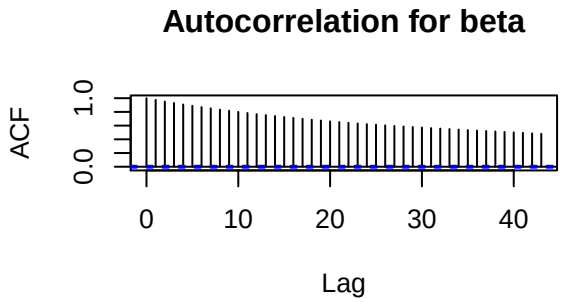
8

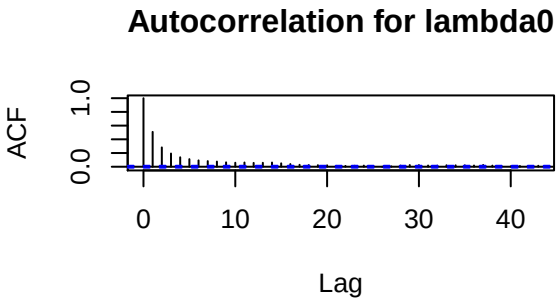
Lastly, we run the block MCMC algorithm for different tuning parameters.

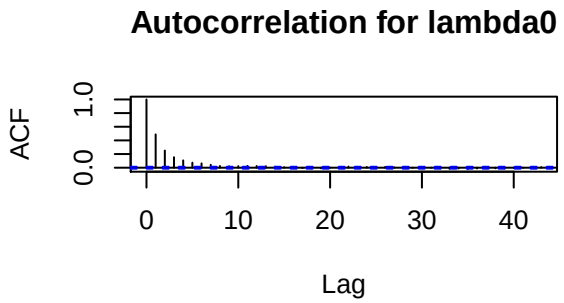
```
# Running chains with different parameters
chains1 = blockMCMC(20000, sigma_t=3.0, sigma_b=0.5)
tracePlots(chains1, 2000)
```

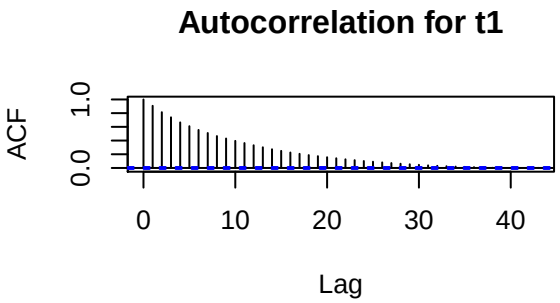



```
acfPlots(chains1)
```

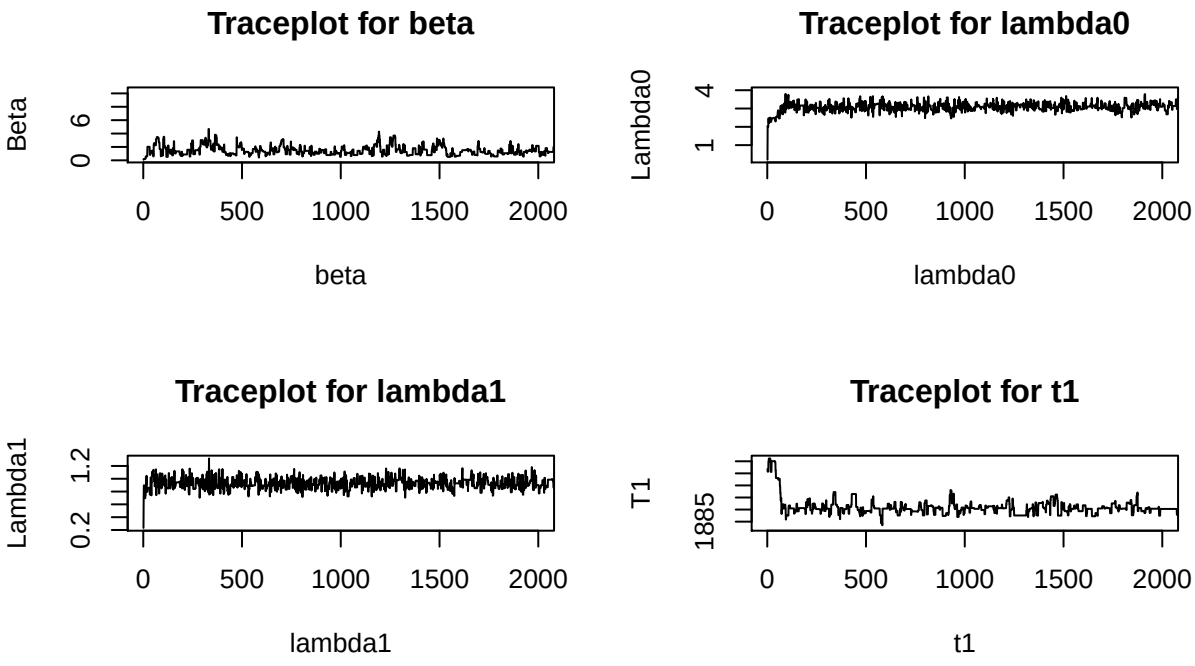




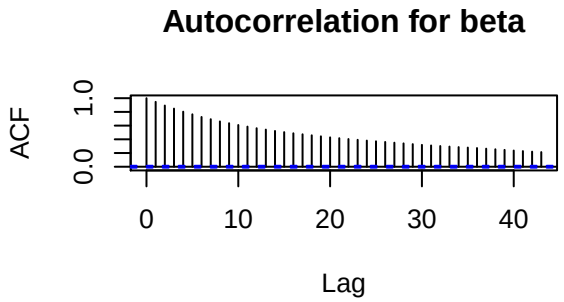


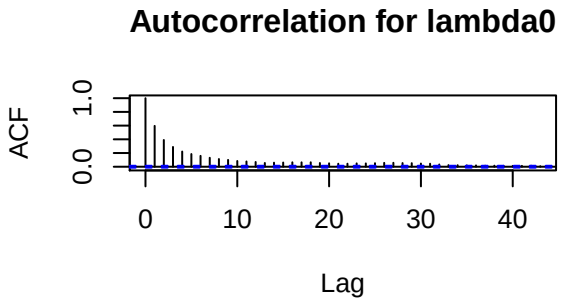


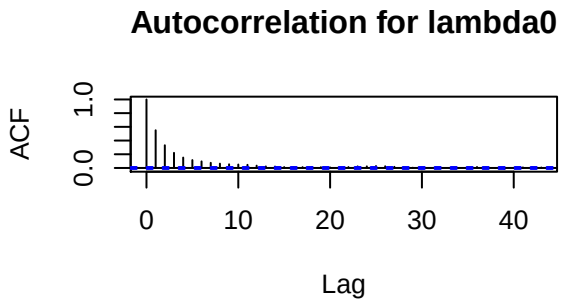

```
chains2 = blockMCMC(20000, sigma_t=3.0, sigma_b=1.0)
tracePlots(chains2, 2000)
```

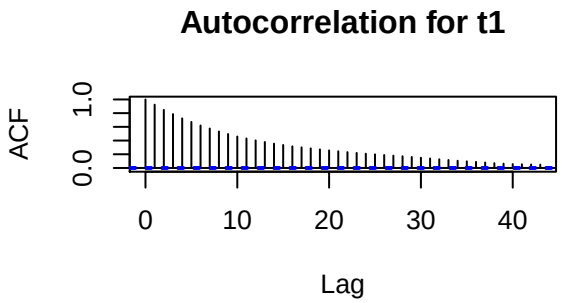



```
acfPlots(chains2)
```

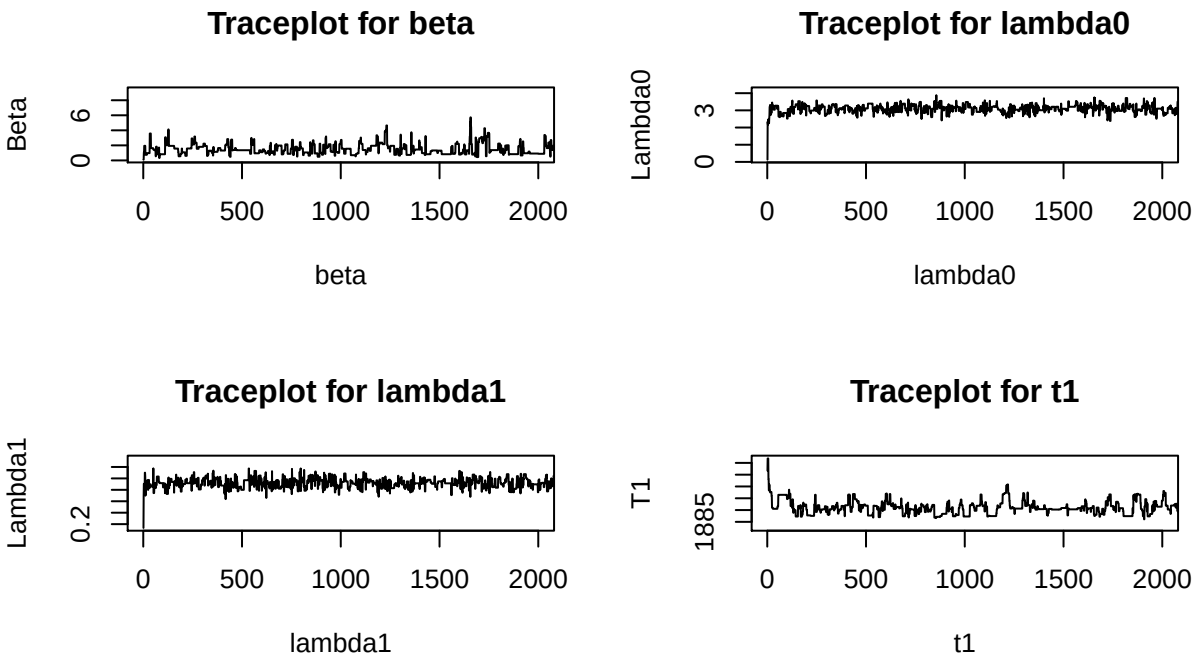




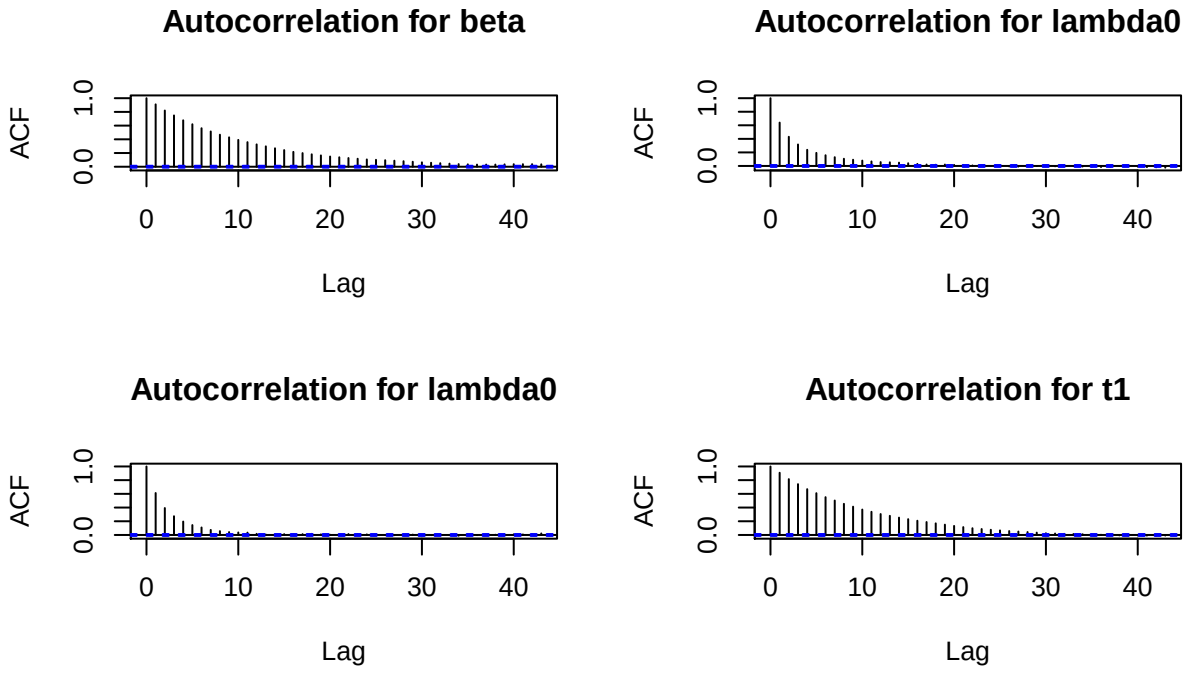




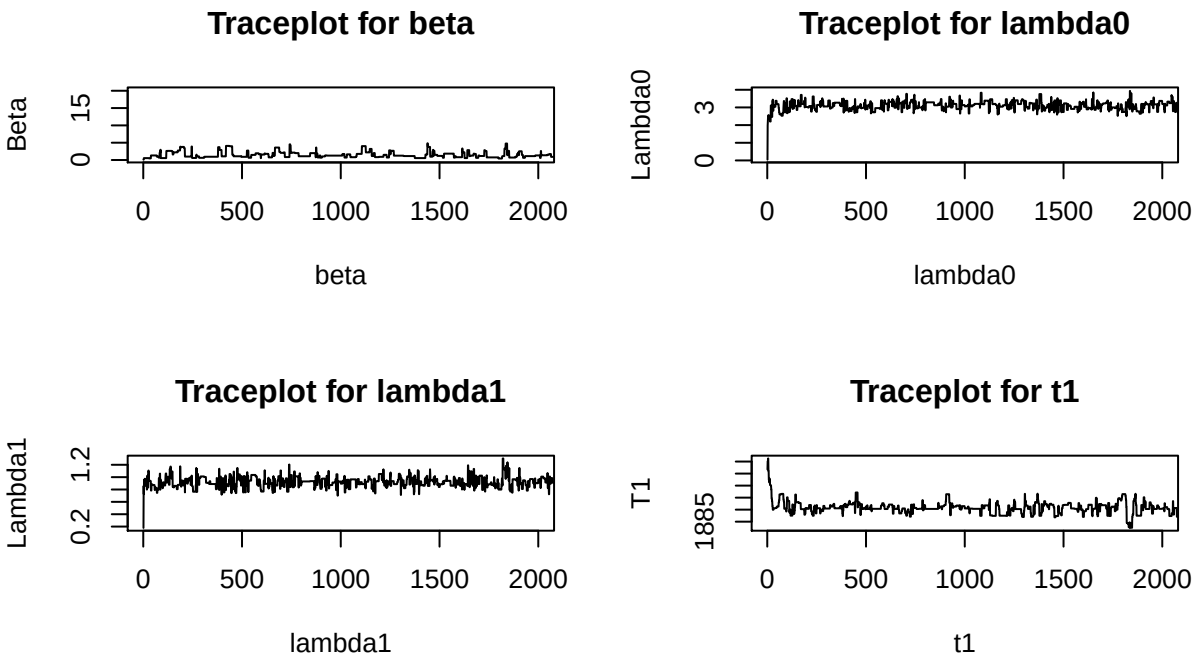

```
chains3 = blockMCMC(20000, sigma_t=3.0, sigma_b=2.0)
tracePlots(chains3, 2000)
```



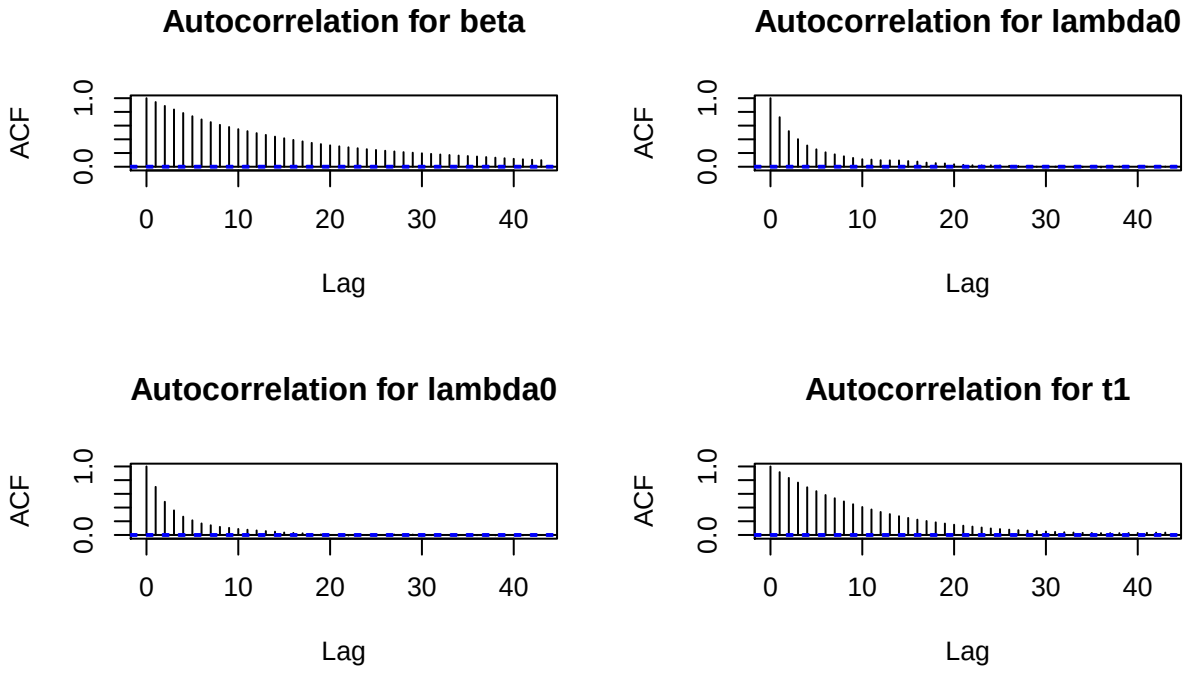

```
acfPlots(chains3)
```




```
chains4 = blockMCMC(20000, sigma_t=3.0, sigma_b=5.0)
tracePlots(chains4, 2000)
```




```
acfPlots(chains4)
```




```
# Calculating means
burnin = 1000

lambda0_mean = mean(chains3$lambda0[-c(1:burnin)])
lambda1_mean = mean(chains3$lambda1[-c(1:burnin)])
t1_mean = mean(chains3$t1[-c(1:burnin)])

print(paste("The MCMC-mean of lambda0 is", lambda0_mean))
```

```
## [1] "The MCMC-mean of lambda0 is 3.10732039830918"
```

```
print(paste("The MCMC-mean of lambda1 is", lambda1_mean))
```

```
## [1] "The MCMC-mean of lambda1 is 0.924150839418025"
```

```
print(paste("The MCMC-mean of t1 is", t1_mean))
```

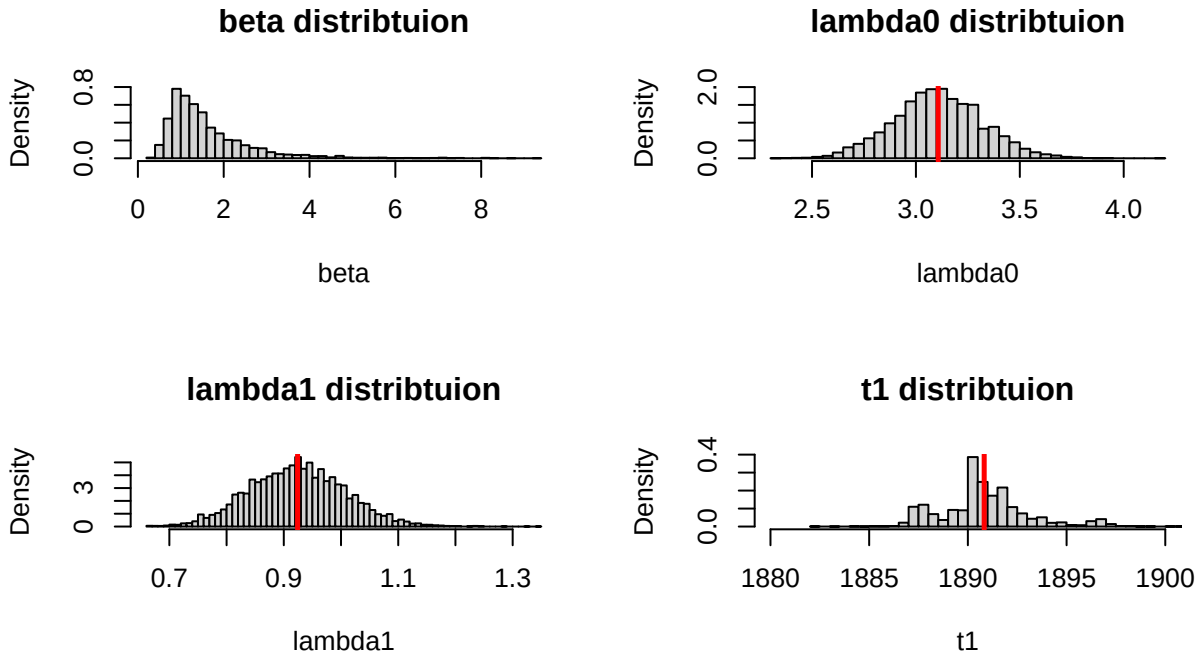
```
## [1] "The MCMC-mean of t1 is 1890.82614103381"
```

```
# Calculatinf covariance
lambda_cov = cov(chains3$lambda0[-c(1:burnin)],chains3$lambda1[-c(1:burnin)])

print(paste("The MCMC-covariance of lambda0 with lambda1 is", lambda_cov))
```

```
## [1] "The MCMC-covariance of lambda0 with lambda1 is 0.0023669928338069"
```

```
# Inspection of distributions
hist(chains3$beta[-c(1:burnin)], freq=F, breaks=50,
     main="beta distribtuion",
```

```
     xlab="beta")
hist(chains3$lambda0[-c(1:burnin)], freq=F, breaks=50,
     main="lambda0 distribtuion",
     xlab="lambda0")
abline(v=lambda0_mean, col="red", lwd=2.5)
hist(chains3$lambda1[-c(1:burnin)], freq=F, breaks=50,
     main="lambda1 distribtuion",
     xlab="lambda1")
abline(v=lambda1_mean, col="red", lwd=2.5)
hist(chains3$t1[-c(1:burnin)], freq=F, breaks=50,
     main="t1 distribtuion",
     xlab="t1", xlim=c(1880,1900))
abline(v=t1_mean, col="red", lwd=2.5)
```



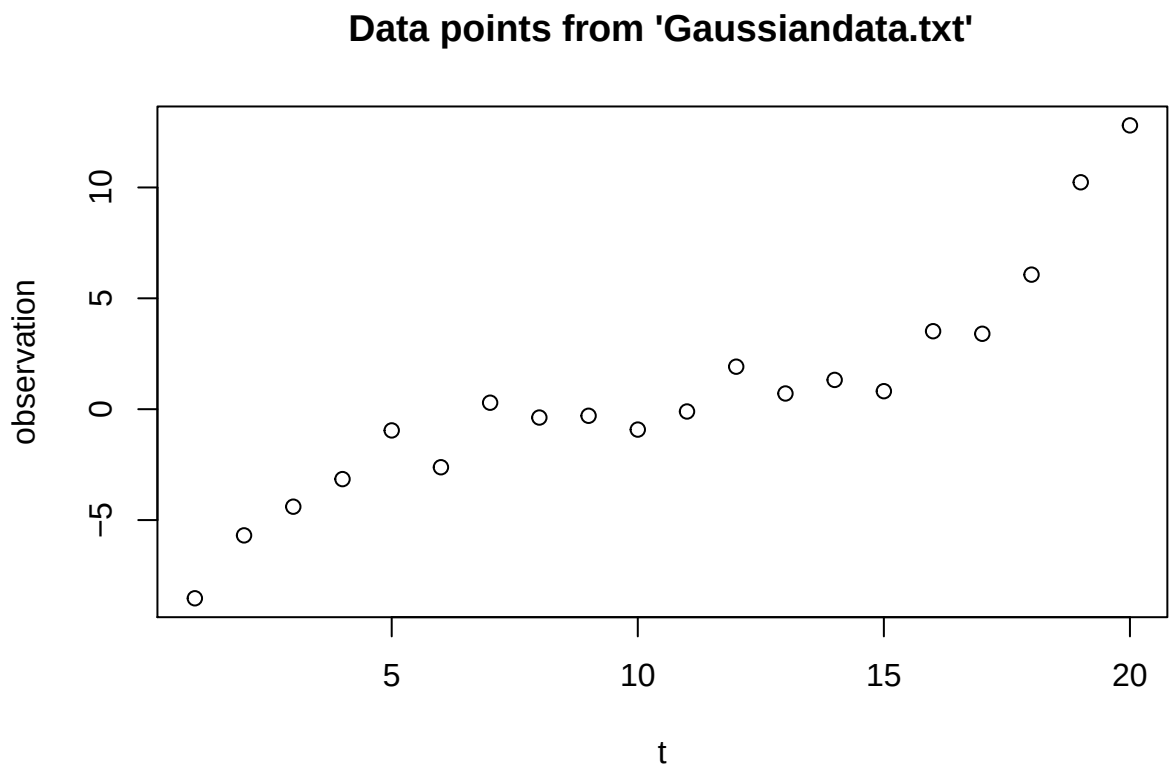
To better inspect the chain behaviour at the beginning we only inspect the traces of the first part, but all other plots involve the full-chain. Moreover, we keep σ_t constant since we have analysed a similar (not same) situation previously. For small σ_β we see that the burn-in is longer than for all others and for high σ_β that the chain starts to pause. In general the mixing is worse than for the Gibbs sampler, which can be seen by the autocorrelation which is a way higher for lag greater than 10 as before. The marginals posteriors for λ_i suggest a Gaussian shape and the one for β shows skwedness. The mean values suggest that before 1891 a little more than 3 disasters happened wheras after only less than 1 per year. The values for λ_i were assumed to be independent, but numerically they have a very small covariance, but we consider it as neglicable.

Problem B: INLA for Gaussian data

In this exercise we are going to consider the problem of smoothing for time series.

The dataset `Gaussiandata.txt` is given. We read and visualise it.

```
# Reading file and convert data type
y = as.double(as.matrix(read.table("Gaussiandata.txt")))
T = length(y)

# Visualise data points on a line
plot(seq(T),y,
     main="Data points from 'Gaussiandata.txt'",
     xlab="t", ylab="observation")
```



The model

It is assumed that the data generation is described by a Gaussian likelihood:

$$1. \quad y_t|x_t \sim \mathcal{N}(x_t, 1), \quad t = 1, \ldots, T$$

wherefrom it immediately follows that y|x $\sim \Pi\pi(y_t|x_t)$ which means that y is conditionally independent given x. Note, taht here y_t does not depend explicitly on θ but via x it still does. The latent field x which is modelled via a second order random walk with parameter θ such that

$$2. \quad \pi(\mathrm{x}|\theta) \propto \mathcal{N}(0, \mathrm{Q}(\theta)^{-1})$$

is obviously a GMRF where we will go into detail of the precision matrix a bit further below. Lastly, a Gamma prior is given for the hyperparameter θ, i.e.

$$3. \quad \theta \sim Gamma(1,1).$$

In this exercise, we utilize different methods and implementation for Bayesian inference of the marginal posteriors - we will compare the results all together at the end and not after each subpart.

1.

Therewith, the model shows the clear structure of a latent Gaussian model, since the latent field is Gaussian distributed. Moreover, the precision matrix of the GMRF is sparse, which makes the model suitable for an INLA approach.

About the precision matrix

In this paragraph, we show the form of the precision matrix which is derived from the RW2 ansatz. The part of interest in the exponent in the exponential, wherefore we only present those calculations here. In the second step those terms that allow to factor out x_t are collected and mixed terms are split symmetrically.

$$\begin{aligned}\sum_{t=3}^{T}(x_t - 2x_{t-1} + x_{t-2})^2 &= \sum_{t=3}^{T} x_t^2 - 4x_t x_{t-1} + 2x_t x_{t-2} + 4x_{t-1}^2 - 4x_{t-1}x_{t-2} + x_{t-2}^2 \\ &= x_1(x_1 - 2x_2 + x_3) + x_2(-2x_1 + 5x_2 - 4x_3 + 1x_4) + \sum_{t=3}^{T-2} x_t(x_{t-2} - 4x_{t-1} + 6x_t - 4x_{t+1} + x_{t+2}) + \\ &= \mathrm{x}^\top \mathrm{Q} \mathrm{x}\end{aligned}$$

where

$$\mathrm{Q} = \begin{bmatrix} 1 & -2 & 1 & & & & \\ -2 & 5 & -4 & 1 & & & \\ 1 & -4 & 6 & -4 & 1 & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & 1 & -4 & 6 & -4 & 1 \\ & & & 1 & -4 & 5 & -2 \\ & & & & 1 & -2 & 1 \end{bmatrix}.$$

This precision matrix is obviously sparse with bandwidth = 2. For later convenience, we introduce $\mathrm{Q}(\theta) = \theta \mathrm{Q}$.

```
# loads
library(Matrix)

# Construct the precision matrix
diags = list(rep(6,T), rep(-4,T-1), rep(1,T-2))
Q = as.matrix(bandSparse(T, T, k=c(0:2), diag=diags, symm=T))
Q[1,1] = Q[T,T] = 1
Q[1,2] = Q[2,1] = Q[T,T-1] = Q[T-1,T] = -2
Q[2,2] = Q[T-1,T-1] = 5

print(Q)
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
##  [1,]    1   -2    1    0    0    0    0    0    0     0     0     0     0
##  [2,]   -2    5   -4    1    0    0    0    0    0     0     0     0     0
```

```
##  [3,]    1   -4    6   -4    1    0    0    0    0    0    0    0    0
##  [4,]    0    1   -4    6   -4    1    0    0    0    0    0    0    0
##  [5,]    0    0    1   -4    6   -4    1    0    0    0    0    0    0
##  [6,]    0    0    0    1   -4    6   -4    1    0    0    0    0    0
##  [7,]    0    0    0    0    1   -4    6   -4    1    0    0    0    0
##  [8,]    0    0    0    0    0    1   -4    6   -4    1    0    0    0
##  [9,]    0    0    0    0    0    0    1   -4    6   -4    1    0    0
## [10,]    0    0    0    0    0    0    0    1   -4    6   -4    1    0
## [11,]    0    0    0    0    0    0    0    0    1   -4    6   -4    1
## [12,]    0    0    0    0    0    0    0    0    0    1   -4    6   -4
## [13,]    0    0    0    0    0    0    0    0    0    0    1   -4    6
## [14,]    0    0    0    0    0    0    0    0    0    0    0    1   -4
## [15,]    0    0    0    0    0    0    0    0    0    0    0    0    1
## [16,]    0    0    0    0    0    0    0    0    0    0    0    0    0
## [17,]    0    0    0    0    0    0    0    0    0    0    0    0    0
## [18,]    0    0    0    0    0    0    0    0    0    0    0    0    0
## [19,]    0    0    0    0    0    0    0    0    0    0    0    0    0
## [20,]    0    0    0    0    0    0    0    0    0    0    0    0    0
##       [,14] [,15] [,16] [,17] [,18] [,19] [,20]
##  [1,]     0     0     0     0     0     0     0
##  [2,]     0     0     0     0     0     0     0
##  [3,]     0     0     0     0     0     0     0
##  [4,]     0     0     0     0     0     0     0
##  [5,]     0     0     0     0     0     0     0
##  [6,]     0     0     0     0     0     0     0
##  [7,]     0     0     0     0     0     0     0
##  [8,]     0     0     0     0     0     0     0
##  [9,]     0     0     0     0     0     0     0
## [10,]     0     0     0     0     0     0     0
## [11,]     0     0     0     0     0     0     0
## [12,]     1     0     0     0     0     0     0
## [13,]    -4     1     0     0     0     0     0
## [14,]     6    -4     1     0     0     0     0
## [15,]    -4     6    -4     1     0     0     0
## [16,]     1    -4     6    -4     1     0     0
## [17,]     0     1    -4     6    -4     1     0
## [18,]     0     0     1    -4     6    -4     1
## [19,]     0     0     0     1    -4     5    -2
## [20,]     0     0     0     0     1    -2     1
```

2.

For the aforementioned model, the new interest lies in the joint posterior $\pi(x, \theta|y)$. By Bayes' formula it can be represented by

$$\pi(x, \theta|y) \propto \pi(y|x, \theta)\pi(x|\theta)\pi(\theta) = \pi(y|x)\pi(x|\theta)\pi(\theta)$$

where the likelihood $\pi(y|x, \theta)$, the latent Gaussian field $\pi(x|\theta)$ and the hyper-prior $\pi(\theta)$ are given from the model, such that we obtain

$$\pi(x, \theta|y) \propto \exp(-\frac{1}{2}(\mathrm{y} - \mathrm{x})^\top(\mathrm{y} - \mathrm{x}))\theta^{\frac{T-2}{2}} \exp(-\frac{1}{2}\mathrm{x}^\top \mathrm{Q}(\theta)\mathrm{x}) \exp(-\theta)$$

which is not in closed form such that direct sampling possible would be possible. This exercise employs MCMC to generate samples from the posterior, in the particular case a blocked Gibbs sampler is used. The proposal steps are split into

- new θ (single value) proposed from its full-conditional $\pi(\theta|\mathrm{x}, \mathrm{y}$
- new x (vector) proposed from its full-conditional $\pi(\mathrm{x}|\theta, \mathrm{y})$.

The full-conditionals are derived from the joint posterior be omitting factors that do not depend on the marginal variable and look like:

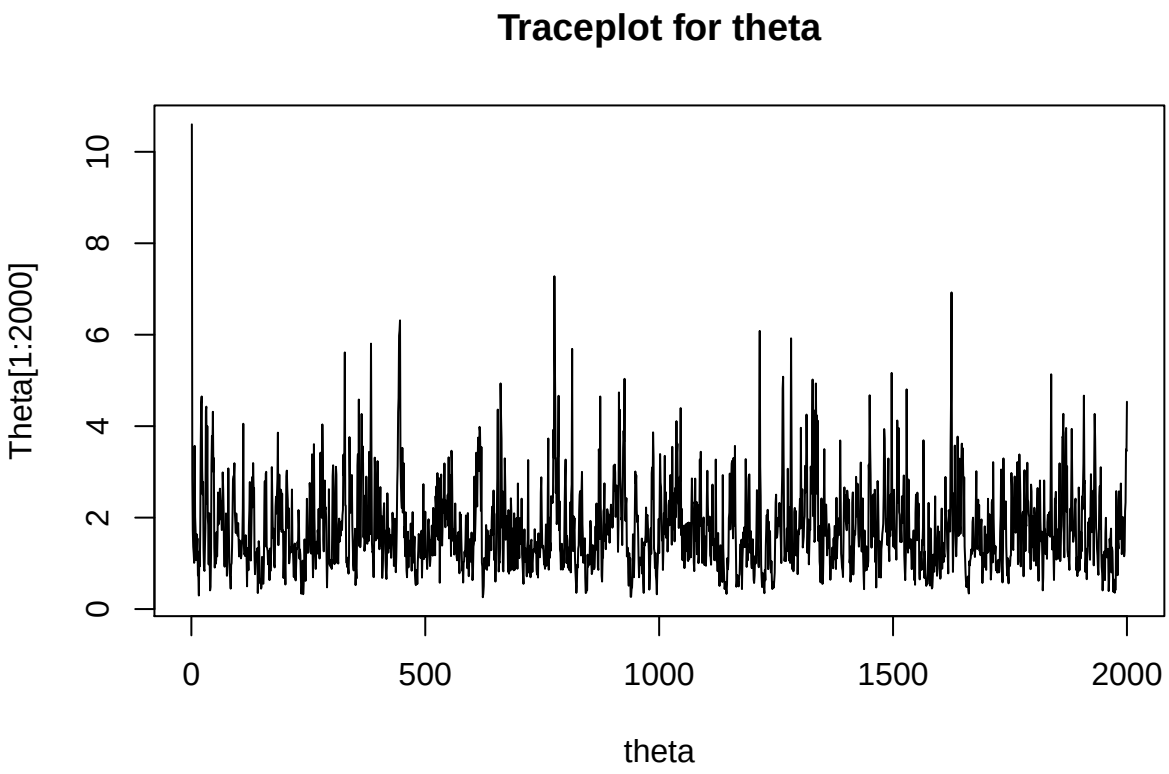
- $\pi(\theta|\mathrm{x}, \mathrm{y}) \propto \theta^{\frac{T-2}{2}} \exp(-(\frac{1}{2}\mathrm{x}^\top \mathrm{Q}\mathrm{x} + 1)\theta) \propto Gamma(\frac{T}{2}, \frac{1}{2}\mathrm{x}^\top \mathrm{Q}\mathrm{x} + 1)$
- $\pi(\mathrm{x}|\theta, \mathrm{y}) \propto \exp(-\frac{1}{2}(\mathrm{x}\mathrm{Q}(\theta)\mathrm{x} - 2\mathrm{y}^\top \mathrm{x}) \propto \mathcal{N}(\mu, \Sigma)$

where the latter is the canoical form a Gaussian distribution. However, to be able to sample from it we have to find the covariance matrix Σ and mean vector μ, which are used in the form $\exp(-\frac{1}{2}(\mathrm{x} - \mu)^\top \mathrm{C}^{-1}(\mathrm{x} - \mu))$. Note that we do not care about multiplicative constants that are independent of x. Then we find by expending the covariance-formulation and comparing to above's canoncical coefficients:

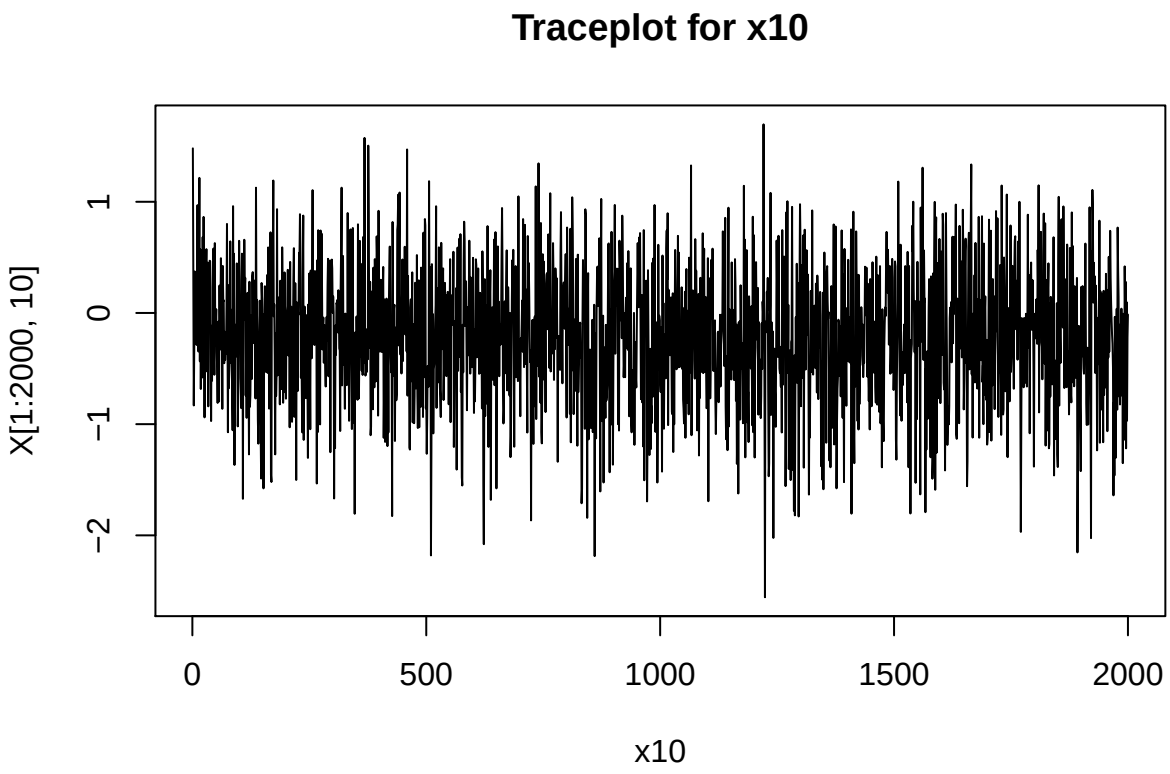
- $\Sigma = (\mathrm{Q}(\theta) + \mathrm{I})^{-1}$
- $\mu = \Sigma \mathrm{y}$

The Gibbs sampler iteratively updates θ and x from their respective proposal densities - in particular, the acceptance probability is always 1 for Gibbs.

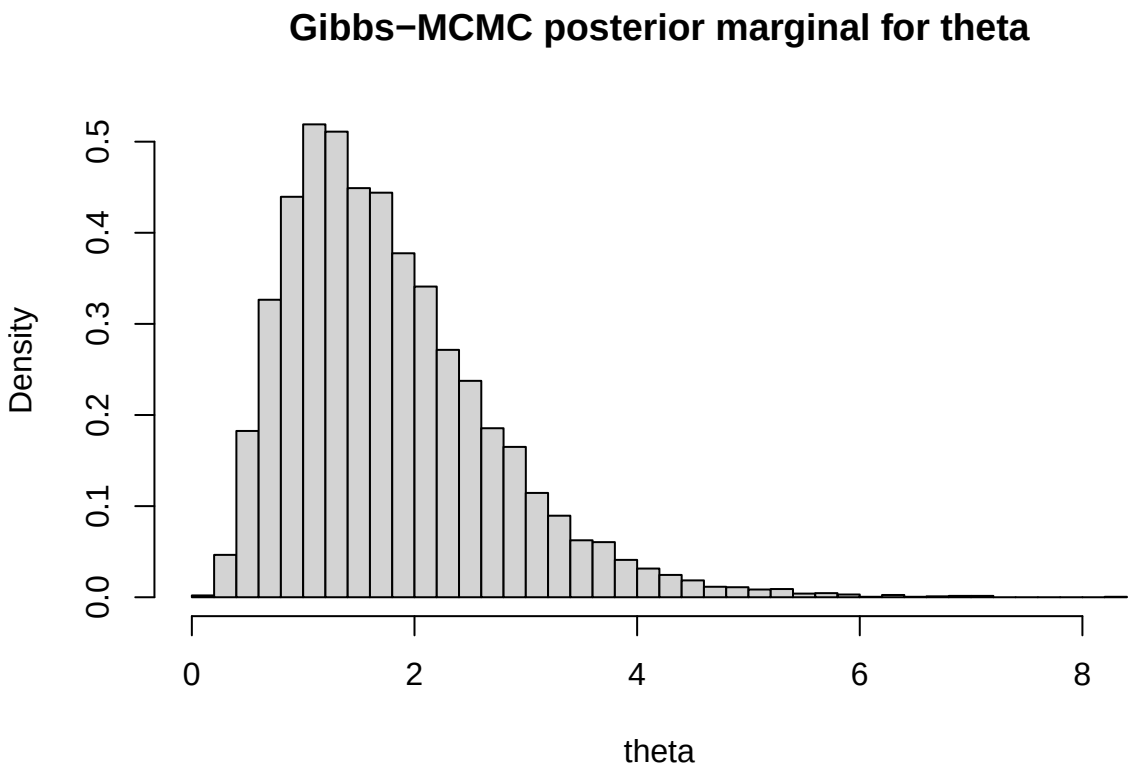
```
# loads
library(mvtnorm)

# Input parameters:
T = length(y)
x = rep(0, T) # initial x
theta = 0.0 # initial theta
N = 11000 # steps in markov chain

# Bookkeping
X = matrix(0, nrow=N, ncol=length(x))
Theta = matrix(0, nrow=N, ncol=1)

#----------------------
# MCMC loop
for (i in 1:N){
  # Update theta
  theta = rgamma(1, shape=T/2, rate=0.5*t(x)%*%Q%*%x+1)
  Theta[i] = theta
  # Update x
  Sigma = solve(theta*Q+diag(T))
  mu = Sigma%*%y
  x = c(rmvnorm(1,mean=mu, sigma=Sigma))
  X[i,] = x
}

#----------------------
# Diagnostics

plot(Theta[1:2000], type="l",
     main="Traceplot for theta",
     xlab="theta")
```



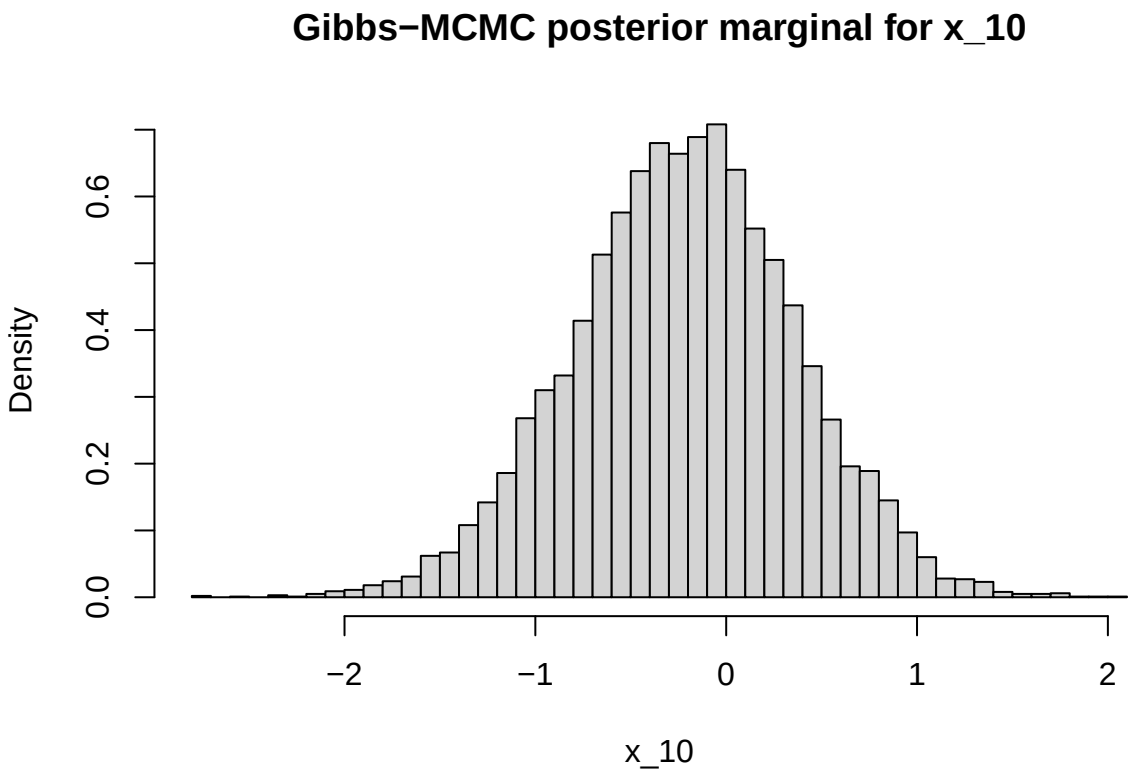

```
plot(X[1:2000,10], type="l",
     main="Traceplot for x10",
     xlab="x10")
```



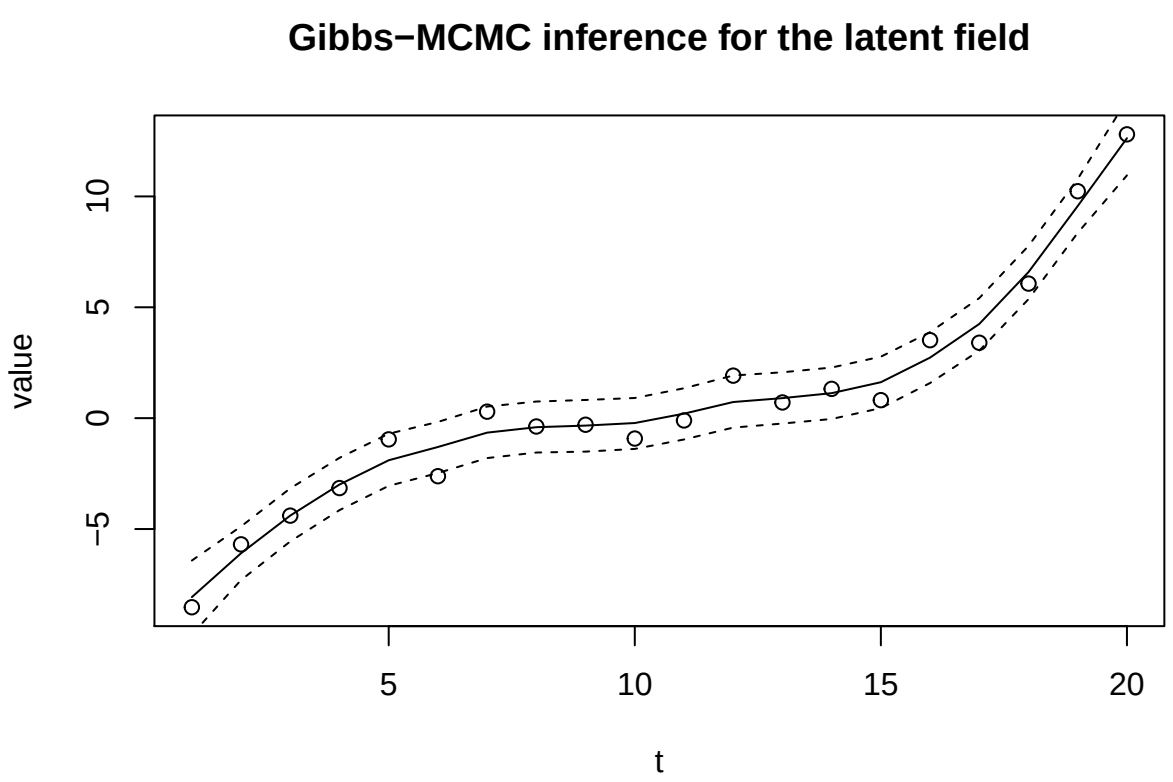

```
#-----------------------
# Inference
burnin = 1000

# hyperparameter theta
hist(Theta[-c(1:burnin)], freq=F, breaks=50,
     main="Gibbs-MCMC posterior marginal for theta",
     xlab="theta")
```




```
# latent field x_10
hist(X[-c(1:burnin),10], freq=F, breaks=50,
     main="Gibbs-MCMC posterior marginal for x_10",
     xlab="x_10")
```




```
# latent field x
Xmeans = colMeans(X[-c(1:burnin),])
Xquants = apply(X[-c(1:burnin),], 2, quantile, probs=c(0.025,0.975))

Ts = seq(T)
plot(Ts,y,
     main="Gibbs-MCMC inference for the latent field",
     xlab="t", ylab="value")
lines(Ts, Xmeans)
lines(Ts, Xquants[1,], lty=2)
lines(Ts, Xquants[2,], lty=2)
```



The histogram shows the posterior marginal for θ which is obtained by Gibbs MCMC. The plot shows again the observation data together with the linearly interpolated posterior means (solid line) and the 0.025 as well as 0.975 quantile (dashed line) from the Gibbs MCMC. We use 10.000 MCMC samples after a burn-in period of 1.000 steps - an inspection of the trace plots suggests that there is only a very short burn-in phase and ignoring the first 1000 steps is more than enough.

3.

As second approach for inference with the posterior, the exercise constrasts the MCMC results with the INLA method. The method focuses on the posterior marginals (which we actually analyzed exclusively in the previous subsection). The INLA scheme relies on numerical approximation of the posterior marginals using the identity

$$\pi(\theta|\mathrm{x}) \propto \frac{\pi(\mathrm{y}|\mathrm{x},\theta)\pi(\mathrm{x}|\theta)\pi(\theta)}{\pi(\mathrm{x}|\theta,\mathrm{y})}$$

where it is noteworthy that all distributions are known from before:

- $\pi(\mathrm{y}|\mathrm{x},\theta) = \pi(\mathrm{y}|\mathrm{x}) \sim \mathcal{N}(\mathrm{x},\mathrm{I})$
- $\pi(\mathrm{x}|\theta) \sim \mathcal{N}(0,\mathrm{Q}(\theta))$
- $\pi(\theta) \sim Gamma(1,1)$
- $\pi(\mathrm{x}|\theta,\mathrm{y}) \sim \mathcal{N}(\mu,\Sigma)$

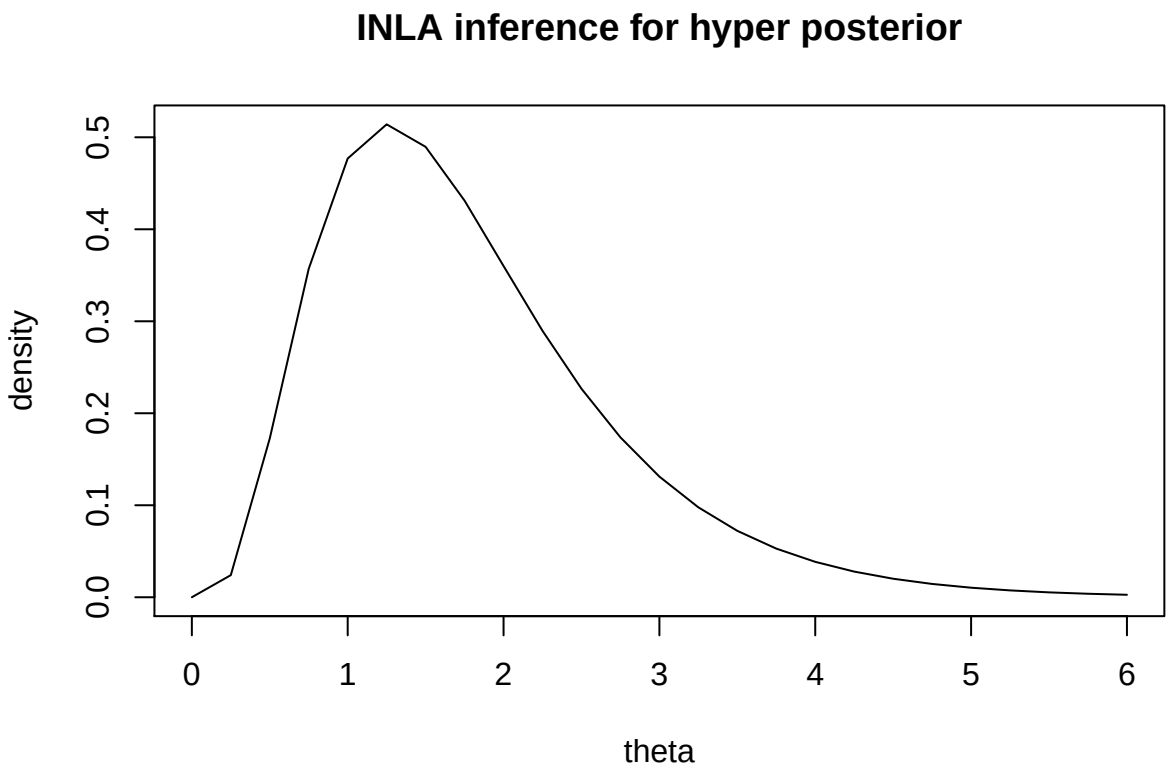
The approximation is build using a θ-grid in the interval $[0,6]$, moreover this is valid for all x wherefore we choose $\mathrm{x} = 0$ for simplicity, such that several terms will vanish.

- First, the hyperprior is independent of x, where $\pi(\theta) = \exp(-\theta)$
- For $\mathrm{x} = 0$ in the latent Gaussian field the argument in the exponential vanishes and therefore it only contributes a constant factor of 1, such that $\pi(0|\theta) = \theta^{\frac{T-2}{2}}$

- The likelihood does not depend explicitly on θ. Further, $\mathrm{x} = 0$ means that the mean in the normal distribution is 0 and we end up with the standard normal (since variances are 1 and uncorrelated) whose density value is evaluted at y, i.e. $\pi(\mathrm{y}|0) \equiv \exp(-\frac{1}{2}\mathrm{y}^\top \mathrm{y})$
- Lastly, the full conditional of the latent field is as derived above $\pi(\mathrm{x})|\mathrm{y}, \theta) = \exp(-\frac{1}{2}(\mathrm{x}-\mu(\theta))\Sigma(\theta)^{-1}(\mathrm{x}-\mu(\theta)))$

Since those calculation involve products and ratios of very small numbers, we perform those calculations in log-scale. Moreover, we normalize the result at the end.

```
# theta grid
tgrid_reso = 0.25
thetas = seq(from=0, to=6, by=tgrid_reso)

x = rep(0,T)

# pi(theta)
lnumerator1 = -thetas

# pi(x|theta)
lnumerator2 = ((T-2)/2)*log(thetas)

# pi(y|x)
lnumerator3 = rep(-1/2*t(y)%*%y, length(thetas))

# pi(x|theta, y)
denominator = rep(0, length(thetas))
for (j in 1:length(thetas)){
  Sigma = solve(thetas[j]*Q+diag(T))
  mu = Sigma%*%y
  denominator[j] = dmvnorm(x,mean=mu, sigma=Sigma)
}
ldenominator = log(denominator)

lpost_marginal_theta = lnumerator1 + lnumerator2 + lnumerator3 - ldenominator
post_marginal_theta = exp(lpost_marginal_theta)
post_marginal_theta = post_marginal_theta/(sum(post_marginal_theta)*tgrid_reso)

plot(thetas, post_marginal_theta,
     main="INLA inference for hyper posterior",
     xlab="theta", ylab="density",
     type="l")
```

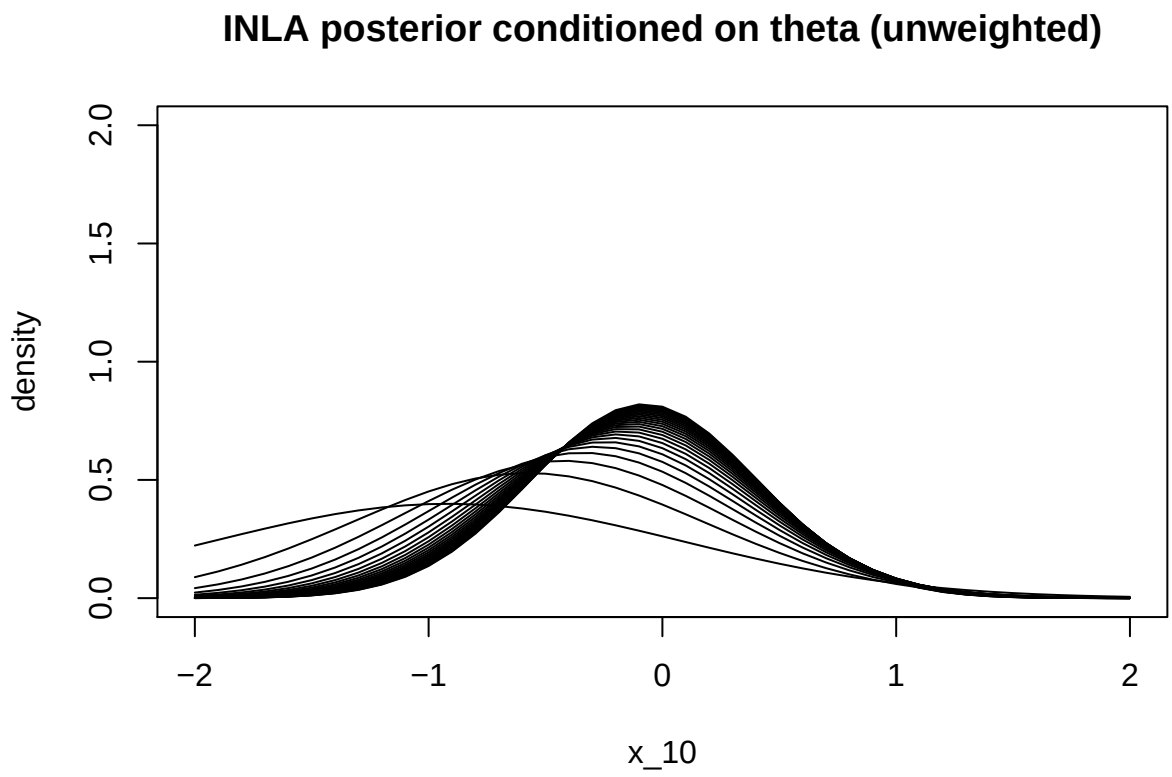


4.

Since $\pi(\mathrm{x}|\theta, \mathrm{y})$ is Gaussian also its component $\pi(x_1 0|\theta, \mathrm{y})$ is Gaussian with distribution $\mathcal{N}(\mu_1 0, \Sigma_{10,10})$.

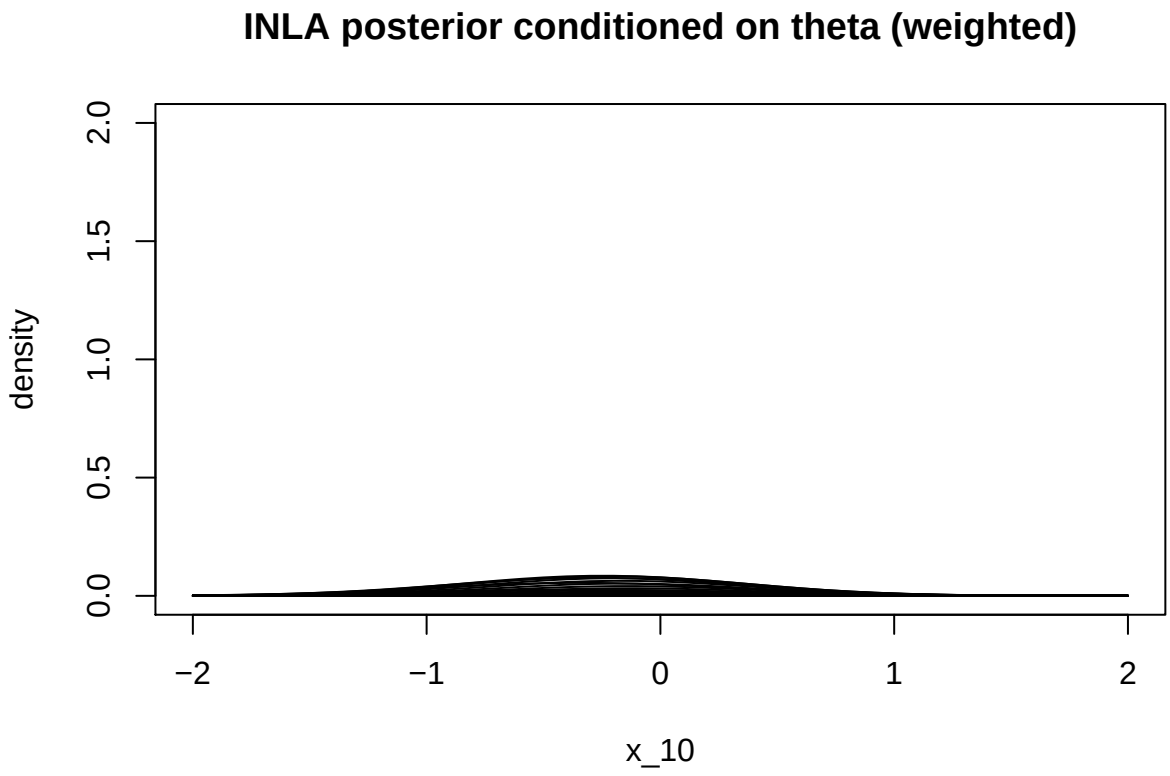
```
index=10

# x10 grid
xgrid_reso = 0.1
xs = seq(from=-2, to=2, by=xgrid_reso)

post_marginal_xs <- function(index, xs){
  # bookkeeping
  post_marginal_x = rep(0,length(xs))
  post_fullcond_x = matrix(0, nrow=length(thetas), ncol=length(xs))
  post_fullcond_x_weighted = matrix(0, nrow=length(thetas), ncol=length(xs))

  # Numerical intregration of integral on x grid
  for (k in 1:length(thetas)){
    # dependencies of theta
    Sigma = solve(thetas[k]*Q+diag(T))
    mu = Sigma%*%y

    # pi(x|theta,y)
    post_fullcond_x[k,] = dnorm(xs, mu[index], sqrt(Sigma[index,index]))
    # pi(x|theta,y) pi(theta|y) Delta theta
    post_fullcond_x_weighted[k,] = post_fullcond_x[k,] * post_marginal_theta[k] * tgrid_reso
```

```
    # Update
    post_marginal_x = post_marginal_x + post_fullcond_x_weighted[k,]
  }

  return(list(post_marginal_x, post_fullcond_x, post_fullcond_x_weighted))

}

# Evaluating function and storing separately
post_marginal_x_result = post_marginal_xs(index, xs)
post_marginal_x = post_marginal_x_result[[1]]
post_fullcond_x = post_marginal_x_result[[2]]
post_fullcond_x_weighted = post_marginal_x_result[[3]]

# plotting pi(x|theta,y)
plot(x=xs,y=post_fullcond_x[1,], type="l",
     main="INLA posterior conditioned on theta (unweighted)",
     ylim=c(0,2), xlab="x_10", ylab="density")
for (i in 2:length(thetas)){
  lines(xs, post_fullcond_x[i,])
}
```

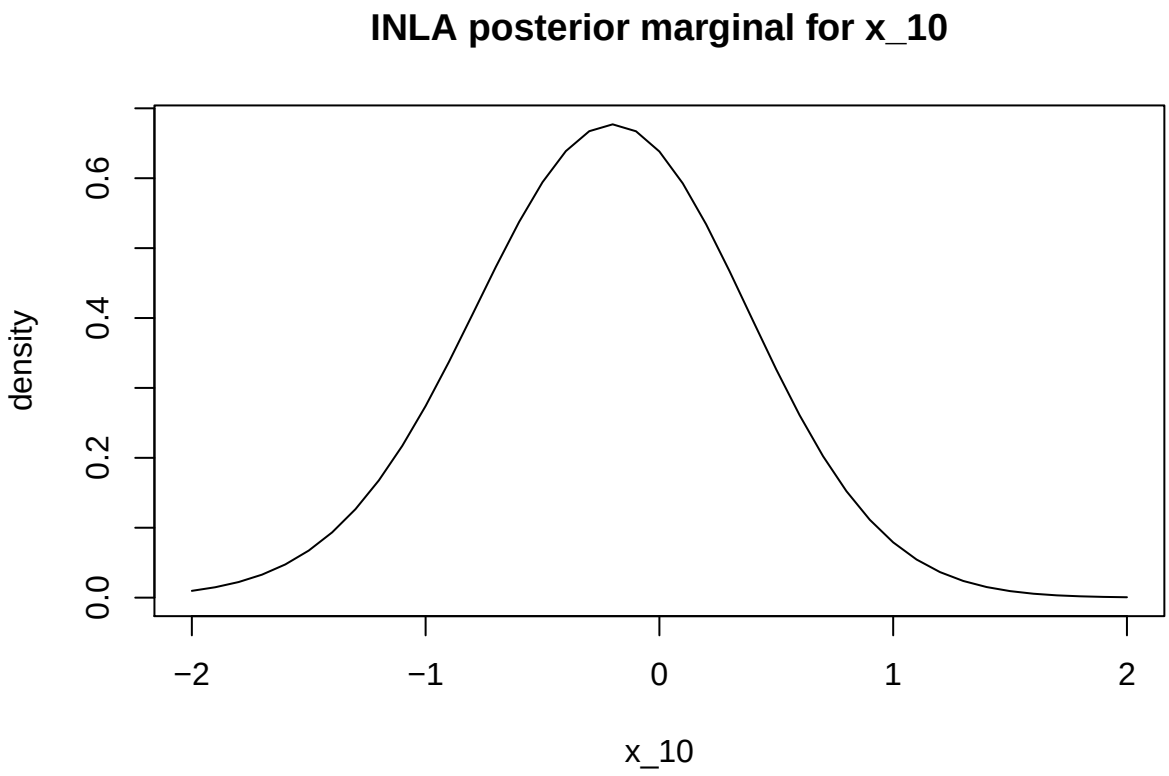



```
# plotting pi(x|theta,y) pi(theta|y) Delta theta
plot(x=xs,y=post_fullcond_x_weighted[1,], type="l",
     main="INLA posterior conditioned on theta (weighted)",
     ylim=c(0,2), xlab="x_10", ylab="density")
```

```
for (i in 2:length(thetas)){
  lines(xs, post_fullcond_x_weighted[i,])
}
```




```
# Plotting pi(x|y)
plot(xs, post_marginal_x, type="l",
     main="INLA posterior marginal for x_10",
     xlab="x_10", ylab="density")
```



5.

In addition to the previous self-implemented INLA routines, we utilize now the `R-INLA` package for exactly the same model.

```
library(INLA)
```

```
## Warning: package 'INLA' was built under R version 4.0.4
```

```
## Loading required package: foreach
```

```
## Loading required package: parallel
```

```
## Loading required package: sp
```

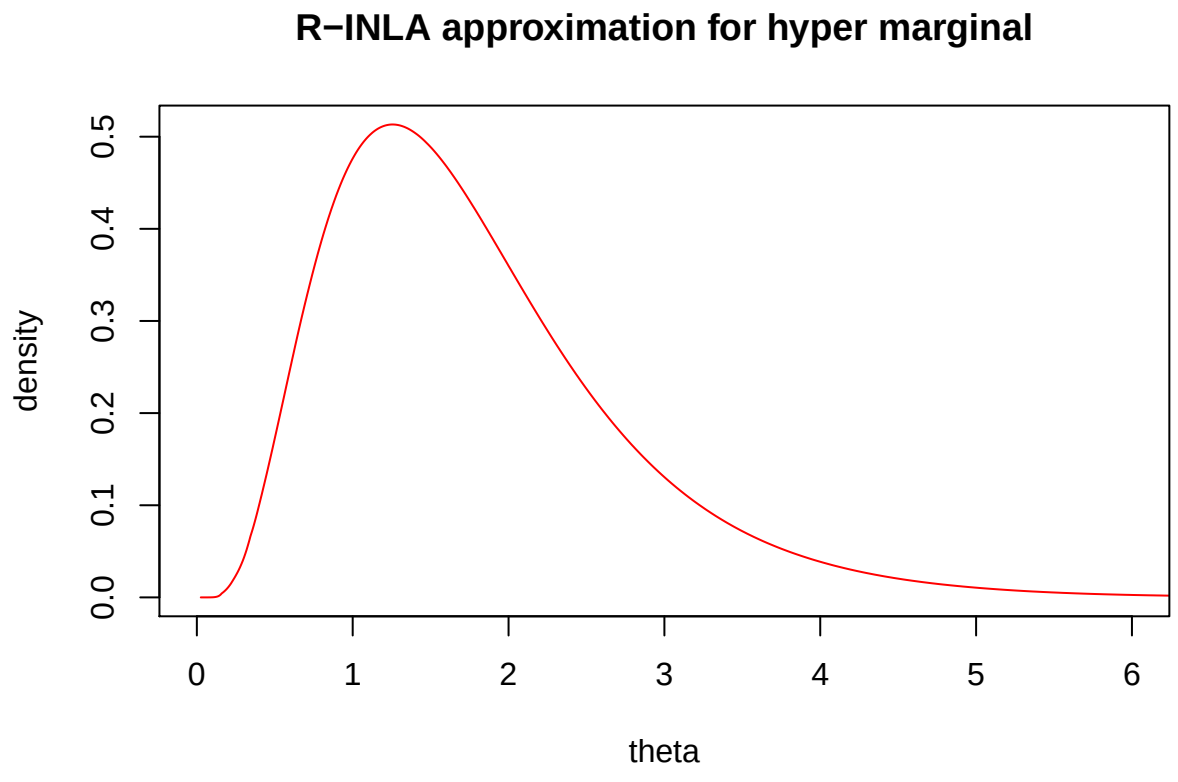
```
## This is INLA_21.02.23 built 2021-02-22 21:11:05 UTC.
##  - See www.r-inla.org/contact-us for how to get help.
##  - Save 379.7Mb of storage running 'inla.prune()'
```

```
data = list(y=y, t=seq(length(y)) )

# Define model
formula = y ~ f( t,
                 model = "rw2",
                 hyper = list(prec=list(prior="loggamma", param=c(1,1))),
                 cyclic=F,
                 constr=T
                 )
```

```r
# Calculate results
result = inla(formula = formula,
              data = data,
              family = "gaussian",
              control.family = list(hyper=list(prec=list(initial=0, fixed=TRUE))),
              control.predictor = list(compute = TRUE, link = 1)
              )

result$summary.hyperpar
```

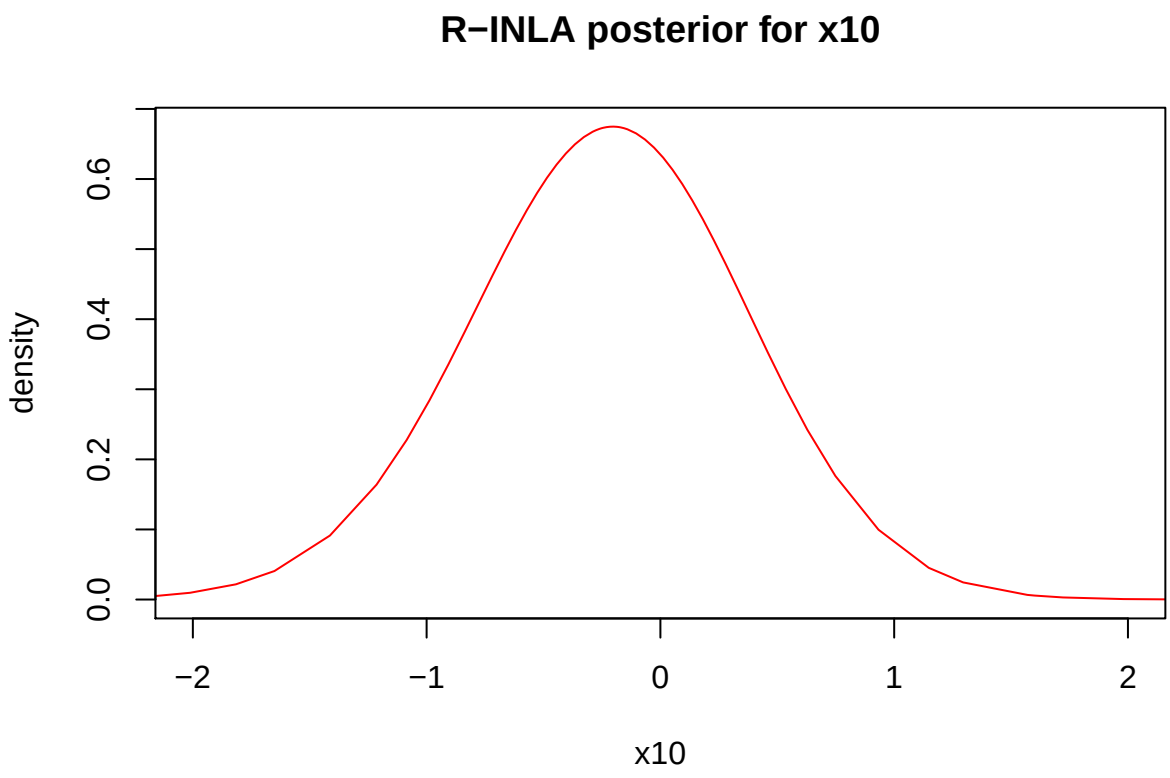
```
##                       mean        sd 0.025quant 0.5quant 0.975quant     mode
## Precision for t 1.792155 0.9408234  0.5090661 1.609802   4.120677 1.256134
```

```r
#--------------------
# Inference

# marginal hyperparam
INLAtheta = inla.smarginal(result$marginals.hyperpar[[1]])

plot(INLAtheta, type="l", col="red",
     main="R-INLA approximation for hyper marginal",
     xlab="theta", ylab="density",
     xlim=c(0,6))
```

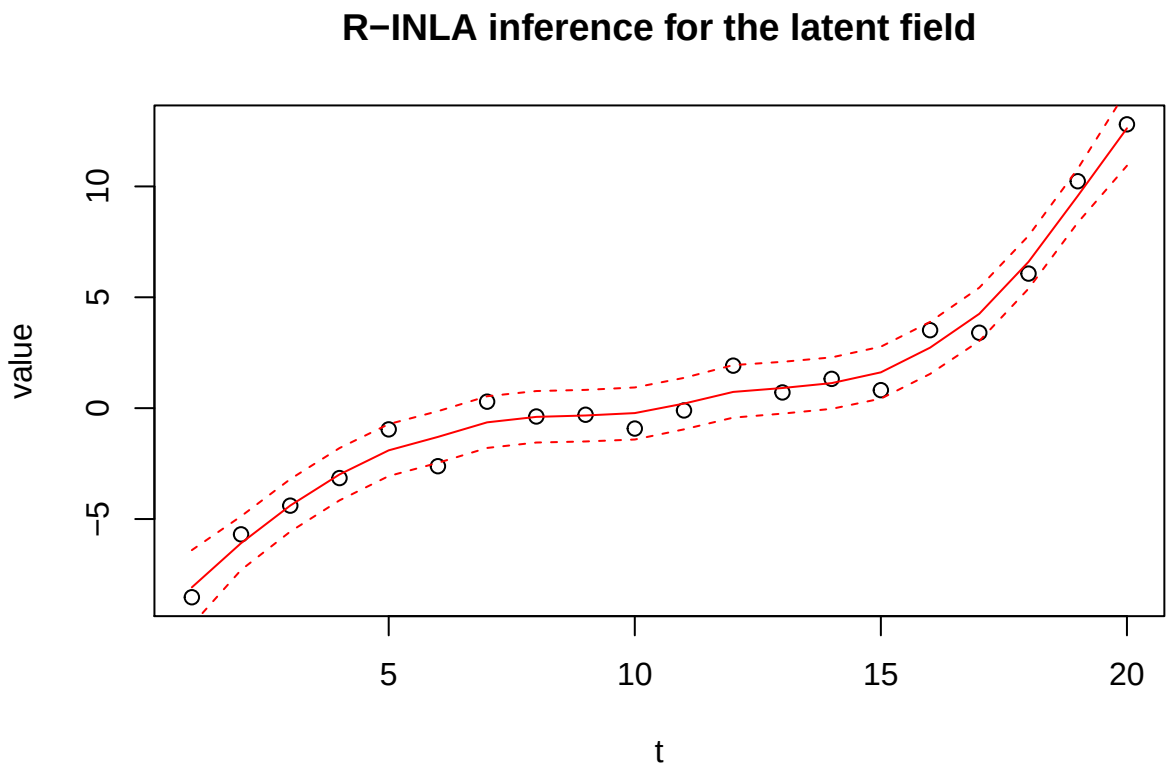



```r
# marginal x10
INLAx10 = result$marginals.linear.predictor$Predictor.10
```

```
plot(INLAx10, type="l", col="red",
     main="R-INLA posterior for x10",
     xlab="x10", xlim=c(-2,2), ylab="density")
```




```
# marginal latent field
INLAx = result$summary.linear.predictor

plot(Ts,y,
     main="R-INLA inference for the latent field",
     xlab="t", ylab="value")
lines(Ts, INLAx$mean, col="red")
lines(Ts, INLAx$`0.025quant`, lty=2, col="red")
lines(Ts, INLAx$`0.975quant`, lty=2, col="red")
```



Comparison of results

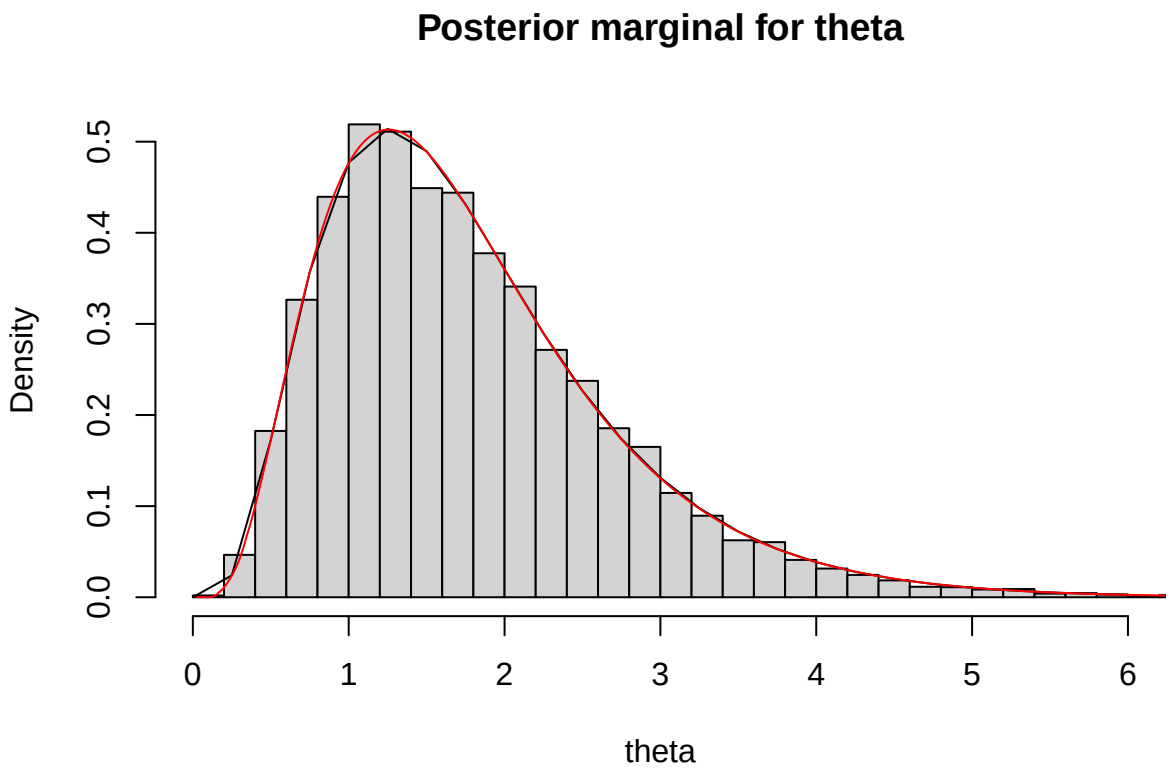
To put the quality of all methods into a nutshell, we compare the properties of the resulting distributions.

Posterior marginal for θ

We compare the result of Gibbs-MCMC, (user implemented) INLA and R-INLA.

- Histrogram from Gibbs-MCMC
- Black line from INLA
- Red line from R-INLA

```
# Plotting as above
hist(Theta[-c(1:burnin)], freq=F, breaks=50,
     main="Posterior marginal for theta",
     xlab="theta", xlim=c(0,6))
lines(thetas, post_marginal_theta)
lines(INLAtheta, col="red")
```



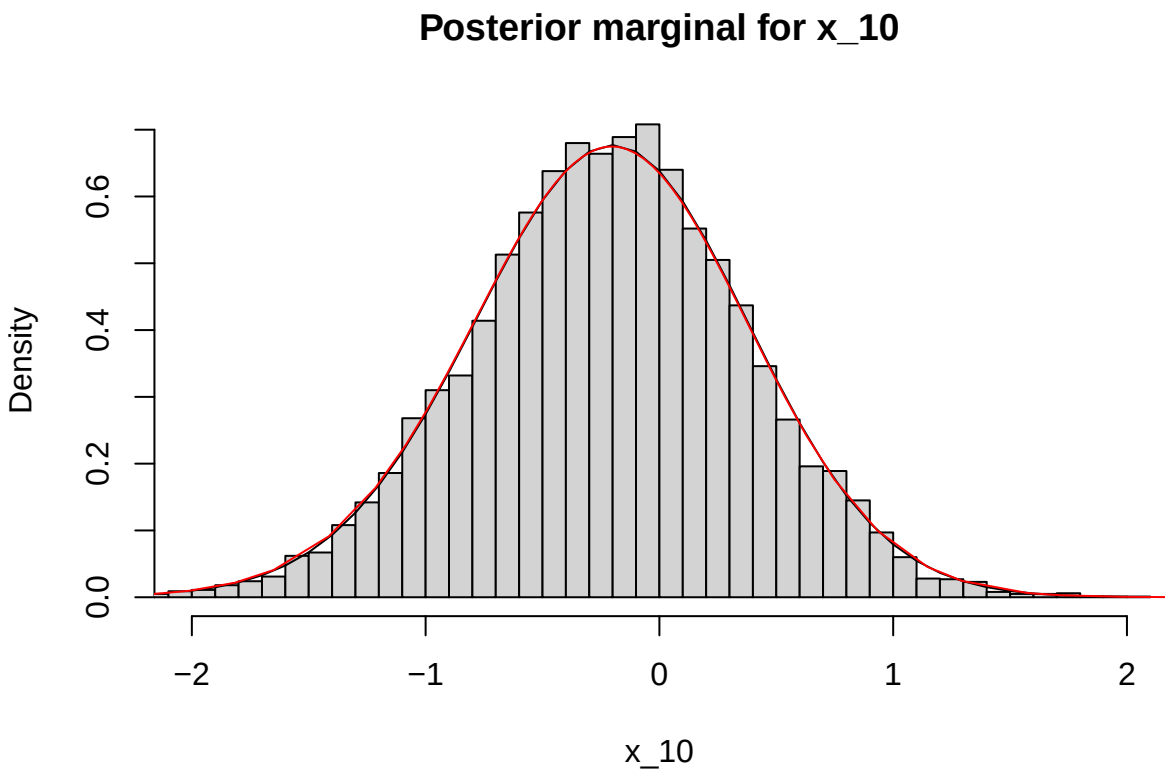
We see that all methods agree (with respect to the chosen precision of calculations).

Posterior marginal for x_10

We compare the result of Gibbs-MCMC, (user implemented) INLA and R-INLA.

- Histrogram from Gibbs-MCMC
- Black line from INLA
- Red line from R-INLA

```
# Plotting as above
hist(X[-c(1:burnin),10], freq=F, breaks=50,
     main="Posterior marginal for x_10",
     xlab="x_10", xlim=c(-2,2))
lines(xs, post_marginal_x)
lines(INLAx10, col="red")
```



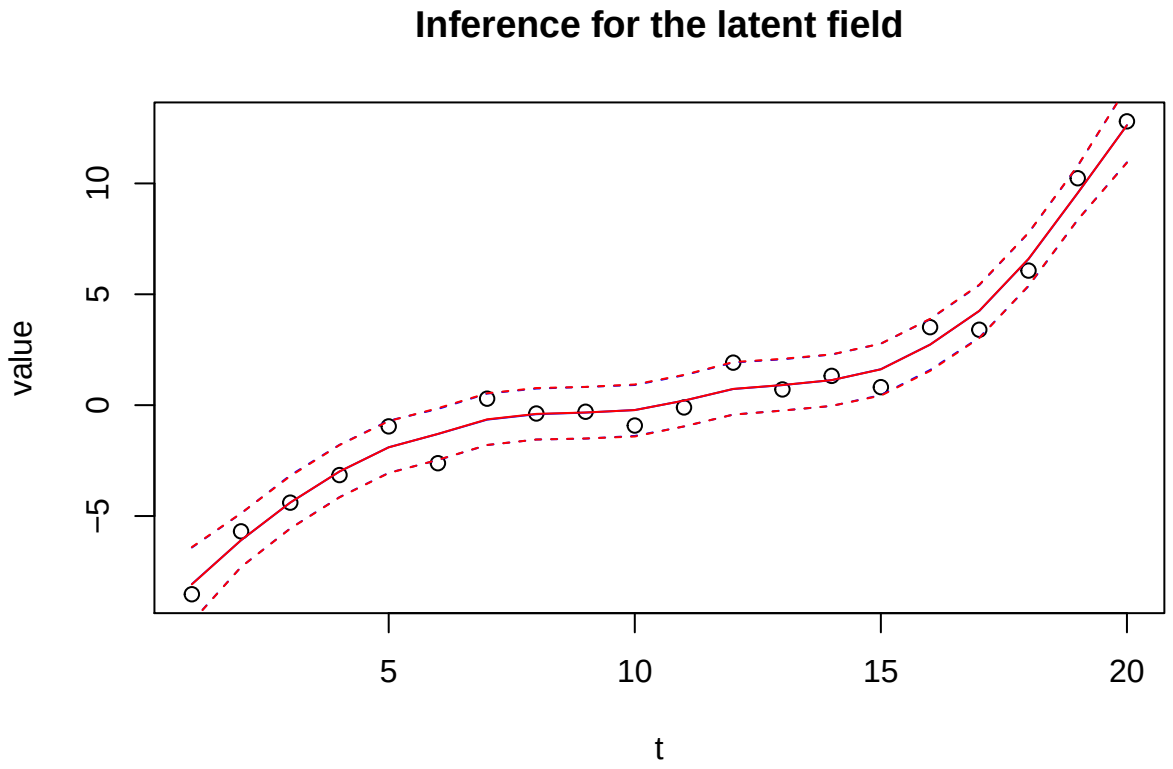
Again all methods agree on the distribution. Note that INLA does not rely on sampling and therefore the numerical approximations of INLA are finer.

Posterior marginal for x

We compare the result of Gibbs-MCMC, (user implemented) INLA and R-INLA.

- Green from Gibbs-MCMC
- Red line from R-INLA

```
# Plotting as above
plot(Ts,y,
     main="Inference for the latent field",
     xlab="t", ylab="value")
lines(Ts, Xmeans, col="blue")
lines(Ts, Xquants[1,], lty=2, col="blue")
lines(Ts, Xquants[2,], lty=2, col="blue")

lines(Ts, INLAx$mean, col="red")
lines(Ts, INLAx$`0.025quant`, lty=2, col="red")
lines(Ts, INLAx$`0.975quant`, lty=2, col="red")
```



Note that we have not included INLA in this comparison since we have only insepected the 10th index before but all indices would be needed here. Nevertheless, the Gibbs-MCMC and R-INLA results coincide very well here.