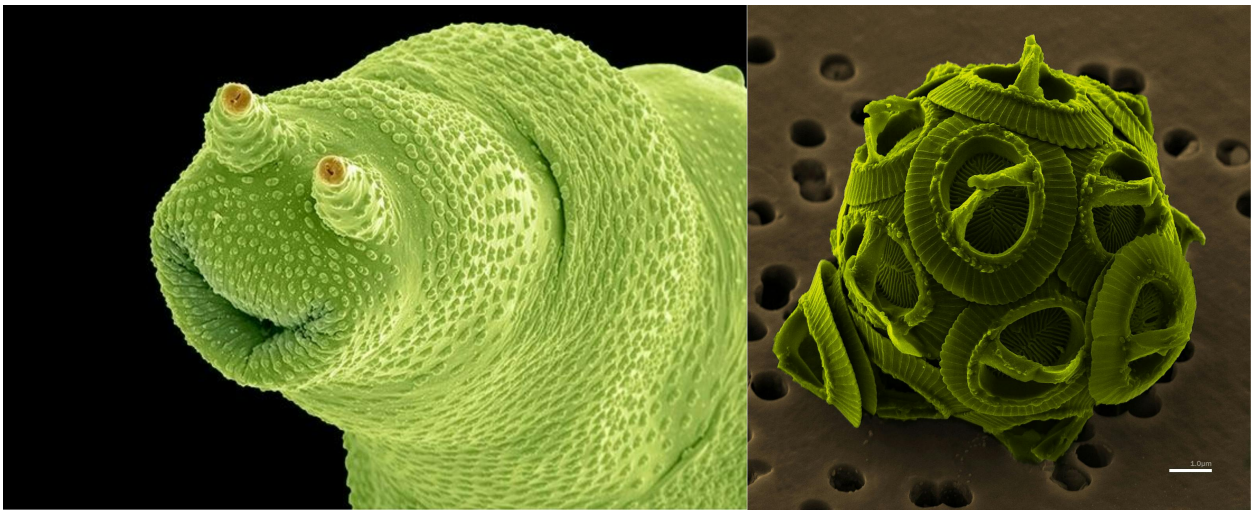
University of Edinburgh

School of Mathematics

Bayesian Data Analysis, 2021/2022, Semester 2

Lecturer: Daniel Paulin

Solutions to Assignment 1



The first picture is a rotifier (by Steve Gschmeissner), the second is a unicellular algae (by NEON ja, colored by Richard Bartz).

Problem 1 - Rotifier and algae data

In this problem, we study an experimental dataset (Blasius et al. 2020, https://doi.org/10.1038/s41586-019-1857-0) about predator-prey relationship between two microscopic organism: rotifier (predator) and unicellular green algae (prey). These were studied in a controlled environment (water tank) in a laboratory over 375 days. The dataset contains daily observations of the concentration of algae and rotifiers. The units of measurement in the algae column is 10^6 algae cells per ml of water, while in the rotifier column it is the number of rotifiers per ml of water.

We are going to apply a simple two dimensional state space model on this data using JAGS. The first step is to load JAGS and the dataset.

```
# We load JAGS
library(rjags)
```

```
## Loading required package: coda
```

```
## Linked to JAGS 4.3.0
```

```
## Loaded modules: basemod,bugs
```

```
#You may need to set the working directory first before loading the dataset
#setwd("/Users/dpaulin/Dropbox/BDA_2021_22/Assignments/Assignment1")
rotifier_algae=read.csv("rotifier_algae.csv")
#The first 6 rows of the dataframe
print.data.frame(rotifier_algae[1:6,])
```

```
##   day algae rotifier
## 1   1  1.50       NA
## 2   2  0.82     6.58
```

```
## 3   3  0.77    17.94
## 4   4  0.36    17.99
## 5   5  0.41    21.12
## 6   6  0.41    17.06
```

As we can see, some values in the dataset are missing (NA).

We are going to model the true log concentrations $\mathbf{x}_t$ by the state space model

$$\mathbf{x}_t = \mathbf{A}\mathbf{x}_{t-1} + \mathbf{b} + \mathbf{w}_t; \quad \mathbf{w}_t \sim N\left(0, \begin{pmatrix} \sigma_R^2 & 0 \\ 0 & \sigma_A^2 \end{pmatrix}\right)$$

where $\mathbf{A}$, $\mathbf{b}$, σ_R^2 and σ_A^2 are model parameters, and t denotes the time point. In particular, $t = 0$ corresponds to day 0, and $t = 1, 2, \ldots, 375$ correspond to days 1-375.

Here $\mathbf{x}_t$ is a two dimensional vector. The first component denotes the logarithm of the rotifier concentration measured in number of rotifiers per ml of water, and the second component denotes the logarithm of the algae concentration measured in 10^6 algae per ml (these units are the same as in the dataset). $\mathbf{A} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$ is a two times two matrix, and $\mathbf{b}$ is a two dimensional vector.

The observation process is described as

$$\mathbf{y}_t = \mathbf{x}_t + \mathbf{v}_t, \quad \mathbf{v}_t \sim N\left(0, \begin{pmatrix} \eta_R^2 & 0 \\ 0 & \eta_A^2 \end{pmatrix}\right),$$

where $\mathbf{y}_t$ is the observed log concentration on day t (for example, $\mathbf{y}_2 = \begin{pmatrix} \log(6.58) \\ \log(0.82) \end{pmatrix}$ in our dataset), while η_R^2 and η_R^2 are additional model parameters.

a)[10 marks] Create a JAGS model that fits the above state space model on the rotifier-algae dataset for the whole 375 days period.

Use 10000 burn-in steps and obtain 50000 samples from the model parameters $\mathbf{A}, \mathbf{b}, \sigma_R^2, \sigma_A^2, \eta_R^2, \eta_A^2$ (4+2+4=10 parameters in total).

Use a Gaussian prior $N\left(\begin{pmatrix} \log(6) \\ \log(1.5) \end{pmatrix}, \begin{pmatrix} 4 & 0 \\ 0 & 4 \end{pmatrix}\right)$ for the initial state $\mathbf{x}_0$, independent Gaussian $N(0, 1)$ priors for each 4 elements of $\mathbf{A}$, Gaussian prior $N\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\right)$ for $\mathbf{b}$, and inverse Gamma (0.1,0.1) prior for the variance parameters $\sigma_R^2, \sigma_A^2, \eta_R^2, \eta_A^2$.

Explain how did you handle the fact that some of the observations are missing (NA) in the dataset.

Explanation: We have implemented the model in JAGS as shown below. The number of elements we model is $n = 375 + 1 = 376$, as day 0 is also included (as we have put a prior on the initial state at day 0). The 2x2 matrix A is denoted by A, the hidden states are stored in the $n \times 2$ matrix x, the observations are stored in the $n \times 2$ matrix y. We pass along a variable called y1 to JAGS as y when compiling the model. In this y1 variable (which is a $n \times 2$ matrix), we select the first column (corresponding to day 0) as NA, and also all other data points with missing data as NA. JAGS is able to handle these NA values automatically by creating stochastic nodes for them (and we could even obtain MCMC samples from them). We ran 5 chains with 50000 samples after 10000 burn-in iterations, starting from an initial position for x based on an interpolated version of y and some random noise.

```
model_string <-
  "
  #Declaring the size of the variables before the model using the var command
```

```
  #This step is not essential, and the code will run without it, but it can improve the performance

  var x[n,2], y[n,2], A[2,2], b[2], c[2], prec.sigma2[2], prec.eta2[2], sigma2[2], eta2[2];

  model{
  #We start by setting the priors
  for(i in 1:2) {
  # prior on x_0, denoted by x[1,] in R
  x[1,i] ~ dnorm(mean.x0[i],prec.x0[i])
  #priors on b, sigma2 and eta2
  b[i]~dnorm(mean.b[i],prec.b[i])
  prec.sigma2[i]~dgamma(alpha.sigma2[i],beta.sigma2[i])
  prec.eta2[i]~dgamma(alpha.eta2[i],beta.eta2[i])
  }

  sigma2<-1/prec.sigma2
  eta2<-1/prec.eta2

  for(i in 1:2) {
  for(j in 1:2) {
  A[i,j]~dnorm(mean.A[i,j],prec.A[i,j])
  }
  }

  #The evolution of the hidden states x according to the model
  for(i in 2:n) {
    x[i,1] ~ dnorm(A[1,1]*x[i-1,1]+A[1,2]*x[i-1,2]+b[1],prec.sigma2[1])
    x[i,2] ~ dnorm(A[2,1]*x[i-1,1]+A[2,2]*x[i-1,2]+b[2],prec.sigma2[2])
  }


  for(i in 1:n) {
    for(j in 1:2) {
      #observations
      y[i,j] ~ dnorm(x[i,j],prec.eta2[j])
      #replicates
      y.rep[i,j] ~ dnorm(x[i,j],prec.eta2[j])
    }
  }
}"
```

```
mean.x0=c(log(6),log(1.5))
prec.x0=c(0.25,0.25)
alpha.sigma2=c(0.1,0.1)
beta.sigma2=c(0.1,0.1)
alpha.eta2=c(0.1,0.1)
beta.eta2=c(0.1,0.1)
mean.b=c(0,0)
prec.b=c(1,1)


mean.A=matrix(0,nrow=2,ncol=2) #Identity matrix
prec.A=matrix(1,nrow=2,ncol=2)
```

```
n=nrow(rotifier_algae)+1

y1=matrix(NA,nrow=n,ncol=2)

y1[2:n,1]=log(rotifier_algae$rotifier)
y1[2:n,2]=log(rotifier_algae$algae)



x.init=y1
require(imputeTS)
```

```
## Loading required package: imputeTS
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
x.init[,1]=na_interpolation(x.init[,1])
x.init[,2]=na_interpolation(x.init[,2])
#Initializing the Markov chain based on interpolation


A.init=matrix(0,nrow=2,ncol=2)
b.init=rep(0.5,2)
prec.sigma2.init=rep(1,2)
prec.eta2.init=rep(1,2)


model1.inits=list()
for(it in 1:5)
{
  model1.inits[[it]]<-list(A=A.init+rnorm(4,sd=1),b=b.init+rnorm(2,sd=1),
                           prec.sigma2=prec.sigma2.init+abs(rnorm(2,sd=1)),
                           prec.eta2=prec.eta2.init+abs(rnorm(2,sd=1)),
                           x=x.init+rnorm(2*n,sd=1))
}



model1.data <- list(n=n,y=y1,mean.x0=mean.x0,prec.x0=prec.x0,
                    mean.b=mean.b,prec.b=prec.b,
                    mean.A=mean.A,prec.A=prec.A,
                              alpha.sigma2=alpha.sigma2,beta.sigma2=beta.sigma2,
                              alpha.eta2=alpha.eta2,beta.eta2=beta.eta2)

# #compiling model
model1=jags.model(textConnection(model_string),data = model1.data,n.chains=5, inits=model1.inits)
```

```
## Compiling model graph
##    Declaring variables
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
```

```
##    Observed stochastic nodes: 735
##    Unobserved stochastic nodes: 1531
##    Total graph size: 4546
##
## Initializing model
```

```
# # Burnin for 10000 samples
update(model1,10000,progress.bar="none")

# # Running the model for 50000 iterations, monitoring the variables A, b, sigma2, and eta2

res.model1=coda.samples(model1,variable.names=c("A","b","sigma2","eta2"), n.iter=50000,progress.bar="no
```

b)[10 marks]

Based on your MCMC samples, compute the Gelman-Rubin convergence diagnostics (Hint: you need to run multiple chains in parallel for this by setting the n.chains parameter). Discuss how well has the chain converged to the stationary distribution based on the results.

Print out the summary of the fitted JAGS model. Do autocorrelation plots for the 4 components of the model parameter A.

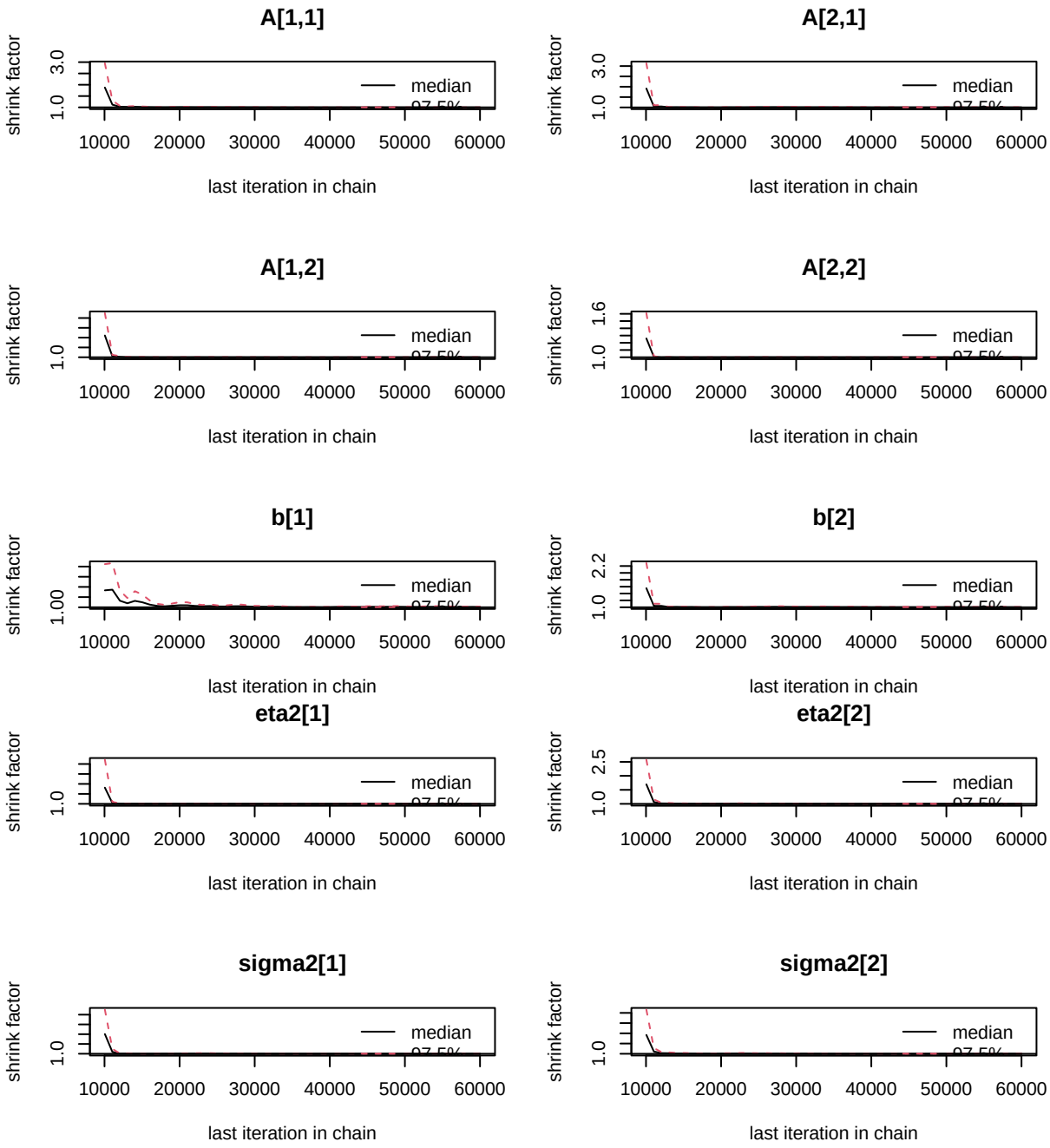
Compute and print out the effective sample sizes (ESS) for each of the model parameters $\mathbf{A}, \mathbf{b}, \sigma_R^2, \sigma_A^2, \eta_R^2, \eta_A^2$.

If the ESS is below 1000 for any of these 10 parameters, increase the sample size/number of chains until the ESS is above 1000 for all 10 parameters.

```
gelman.diag(res.model1)
```

```
## Potential scale reduction factors:
##
##           Point est. Upper C.I.
## A[1,1]             1       1.00
## A[2,1]             1       1.01
## A[1,2]             1       1.01
## A[2,2]             1       1.00
## b[1]               1       1.00
## b[2]               1       1.01
## eta2[1]            1       1.00
## eta2[2]            1       1.01
## sigma2[1]          1       1.00
## sigma2[2]          1       1.00
##
## Multivariate psrf
##
## 1
```

```
gelman.plot(res.model1)
```



Explanation:[3 marks] As we can see from either the gelman.diag or gelman.plot results, the Gelman-Rubin diagnostics values are below 1.05 for each of the parameters, indicating that we have approximated the stationary distribution quite well.

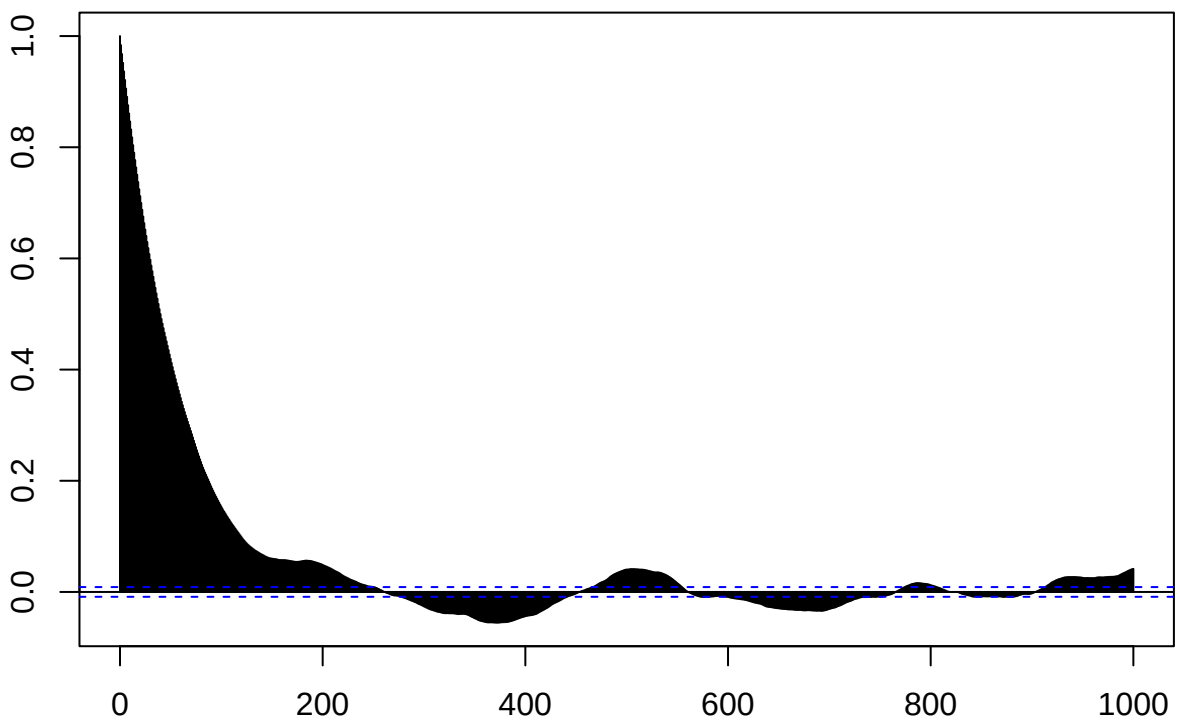
```
summary(res.model1)
```

```
##
## Iterations = 10001:60000
## Thinning interval = 1
## Number of chains = 5
## Sample size per chain = 50000
##
## 1. Empirical mean and standard deviation for each variable,
```

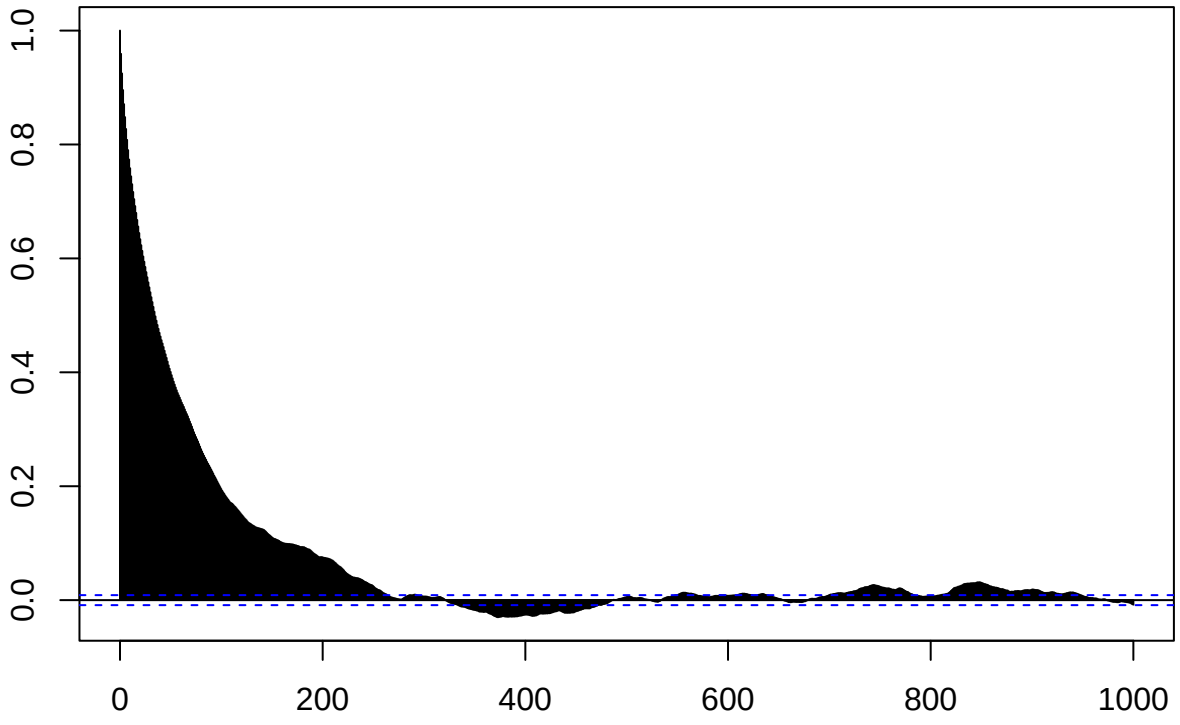
```
##    plus standard error of the mean:
##
##               Mean       SD  Naive SE Time-series SE
## A[1,1]     0.57454 0.045278 9.056e-05      0.0010024
## A[2,1]    -0.42145 0.049640 9.928e-05      0.0010639
## A[1,2]     0.60257 0.089987 1.800e-04      0.0016662
## A[2,2]     0.25682 0.058680 1.174e-04      0.0004739
## b[1]       1.85025 0.139820 2.796e-04      0.0029545
## b[2]       0.73246 0.160455 3.209e-04      0.0034399
## eta2[1]    0.02727 0.008427 1.685e-05      0.0001023
## eta2[2]    0.09084 0.030873 6.175e-05      0.0005920
## sigma2[1]  0.12941 0.019331 3.866e-05      0.0002710
## sigma2[2]  0.16564 0.032401 6.480e-05      0.0005488
##
## 2. Quantiles for each variable:
##
##               2.5%      25%      50%     75%    97.5%
## A[1,1]     0.48842  0.54369  0.57352  0.6044  0.66564
## A[2,1]    -0.52063 -0.45474 -0.42041 -0.3874 -0.32680
## A[1,2]     0.45191  0.53881  0.59410  0.6568  0.80162
## A[2,2]     0.13956  0.21820  0.25746  0.2964  0.36980
## b[1]       1.57630  1.75679  1.85055  1.9440  2.12483
## b[2]       0.42375  0.62262  0.73082  0.8405  1.05130
## eta2[1]    0.01382  0.02116  0.02624  0.0323  0.04644
## eta2[2]    0.03495  0.06833  0.08992  0.1119  0.15347
## sigma2[1]  0.09069  0.11663  0.12971  0.1424  0.16660
## sigma2[2]  0.10597  0.14248  0.16468  0.1876  0.23083
```

Explanation:[2 marks] We printed out the posterior summaries for the model parameters A, b, sigma2 and eta2 using the summary(res.model1) command. We can see that the standard deviations are typically much smaller than the means in absolute value, indicating that the the amount of data is sufficient to fit the model parameters with some degree of confidence. The time series SE is much smaller than the posterior means (in absolute value), showing that our estimates are rather accurate and the chain was mixing reasonably well.

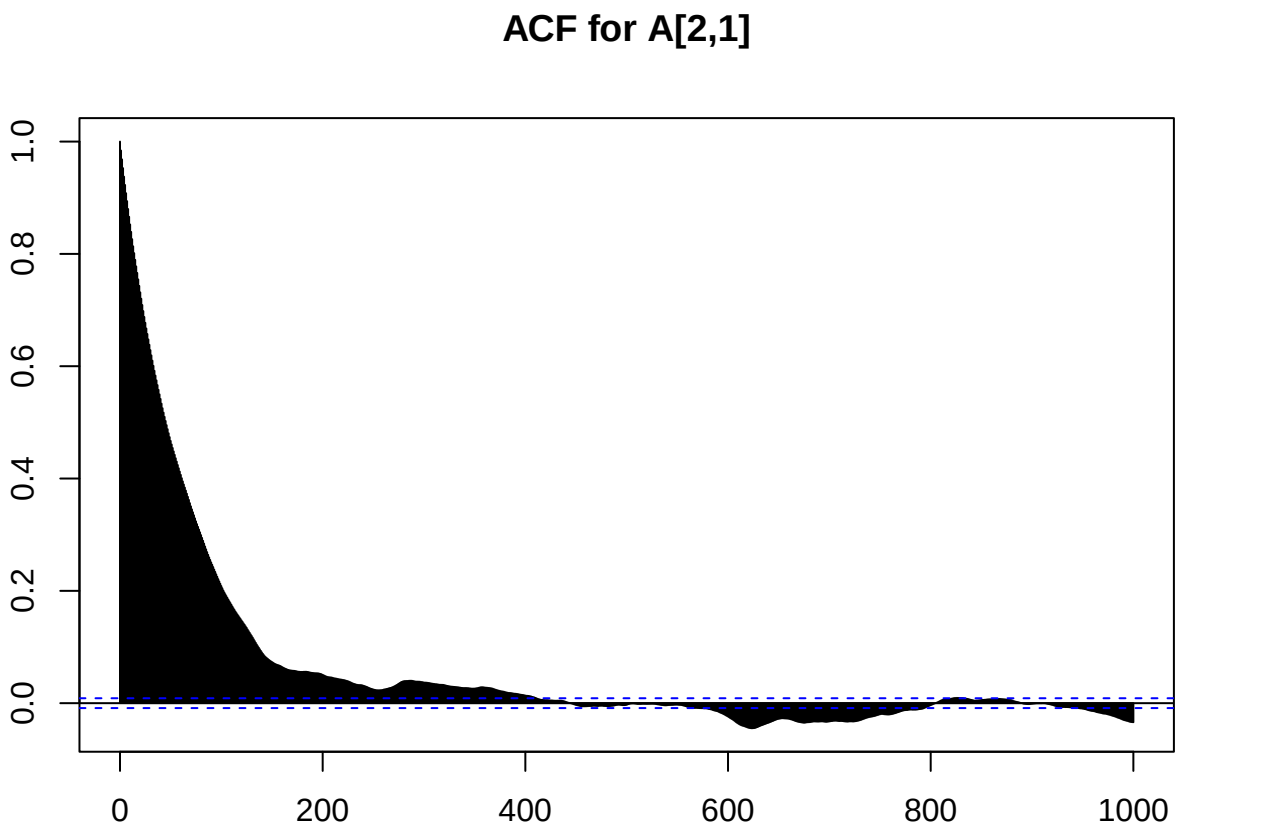
```
par(mar = c(2, 2, 4, 2))
acf(res.model1[[1]][,"A[1,1]"],lag.max=1000,main="ACF for A[1,1]")
```



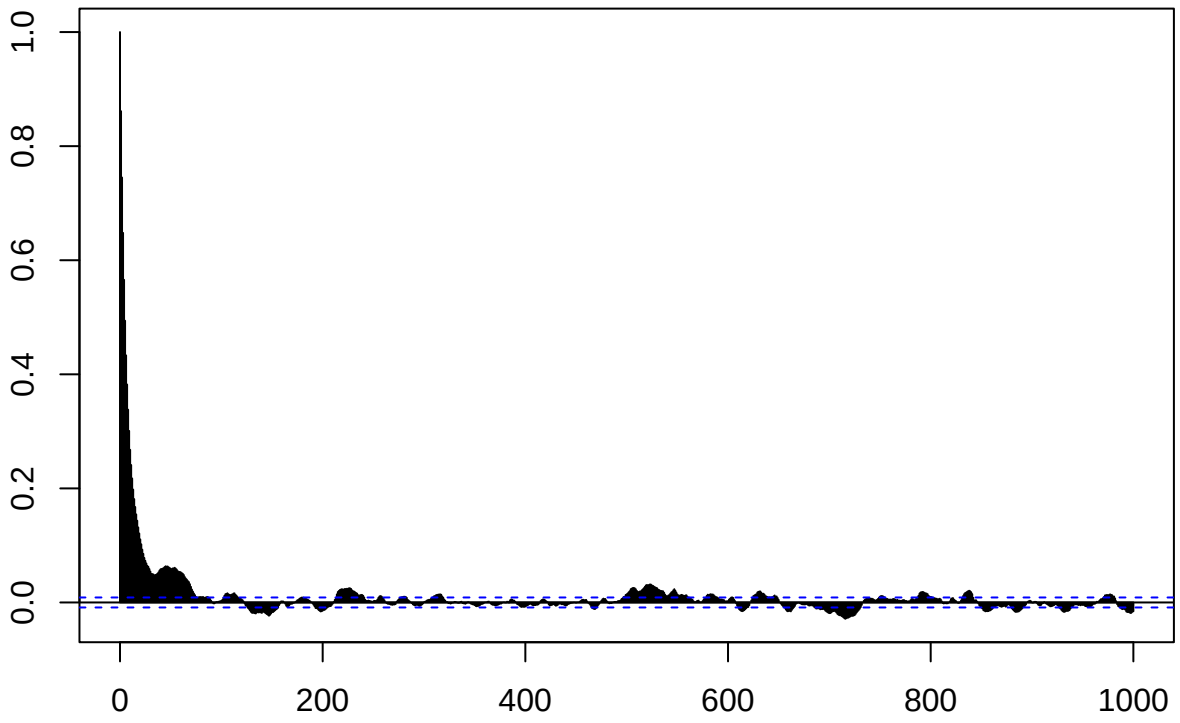

```
acf(res.model1[[1]][,"A[1,2]"],lag.max=1000,main="ACF for A[1,2]")
```



```
acf(res.model1[[1]][,"A[2,1]"],lag.max=1000,main="ACF for A[2,1]")
```




```
acf(res.model1[[1]][,"A[2,2]"],lag.max=1000,main="ACF for A[2,2]")
```



Explanation: We plotted the autocorrelation functions for the 4 components of A using the acf function. We set the lag.max parameter to 500 to clearly display the plots. We can see that the samples from A[1,1] suffer from the most autocorrelation, but it's mostly gone after a lag of 100, and almost completely gone after a lag of 300.

```
effectiveSize(res.model1)
```

```
##    A[1,1]    A[2,1]    A[1,2]    A[2,2]      b[1]      b[2]   eta2[1]   eta2[2]
##  2043.362  2187.434  2954.427 15460.525  2245.090  2187.721  6806.732  2730.201
## sigma2[1] sigma2[2]
##  5209.922  3500.911
```

Explanation: In our code in part a), we ran 5 chains with 50000 samples after 10000 burn-in iterations. We have computed the ESS for the model parameters A, b, sigma2 and eta2 using the effectiveSize function. The ESS is above 2000 for each of them, so no further samples are needed.

c)[10 marks]

We are going to perform posterior predictive checks to evaluate the fit of this model on the data (using the priors stated in question a). First, create replicate observations from the posterior predictive using JAGS. The number of replicate observations should be at least 1000.

Compute the minimum, maximum, and median for both log-concentrations (i.e. both for rotifier and algae, $3 \cdot 2 = 6$ in total).

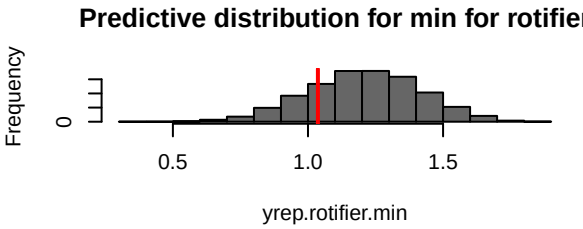
Plot the histograms for these quantities together with a line that shows the value of the function considered on the actual dataset (see the R code for Lecture 2 for an example). Compute the DIC score for the model (Hint: you can use the `dic.samples` function for this).

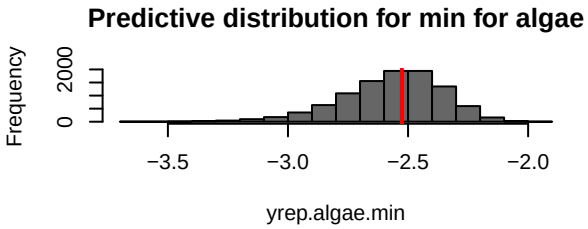
Discuss the results.

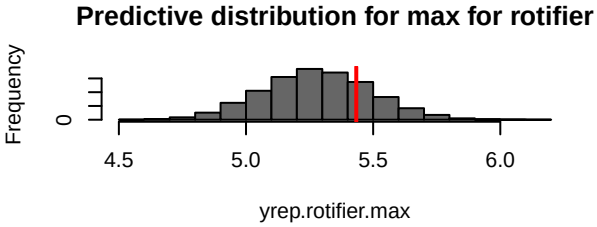
Explanation: We have created replicates by including them in the model. We obtain samples from the

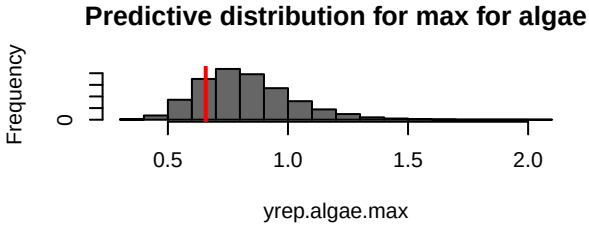
replicates by coda.samples, with y.rep included in variable.names. We have pruned out the observations with NA values from the replicates (as they would distort the results), and compared the values of the functions min, max, median on the rotifier and algae data with the histograms of the replicates.

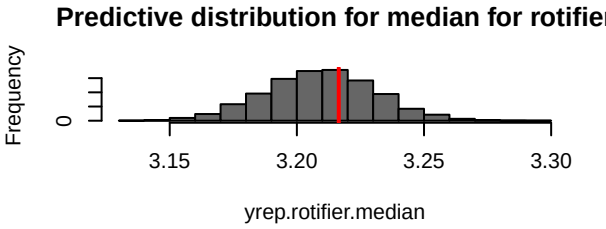
```
res.replicates=coda.samples(model1,variable.names=c("y.rep"), n.iter=2000,progress.bar="none")
yrep=as.matrix(res.replicates)
yrep.rotifier=yrep[,1:n]
yrep.algae=yrep[,(n+1):(2*n)]

ind.rotifier.not.NA=which(!is.na(y1[,1]))
ind.algae.not.NA=which(!is.na(y1[,2]))
rotifier.not.NA=y1[ind.rotifier.not.NA,1]
algae.not.NA=y1[ind.algae.not.NA,2]

yrep.rotifier.not.NA=yrep.rotifier[,ind.rotifier.not.NA]
yrep.algae.not.NA=yrep.algae[,ind.algae.not.NA]

yrep.rotifier.min=apply(yrep.rotifier.not.NA,1,min)
yrep.rotifier.max=apply(yrep.rotifier.not.NA,1,max)
yrep.rotifier.median=apply(yrep.rotifier.not.NA,1,median)

yrep.algae.min=apply(yrep.algae.not.NA,1,min)
yrep.algae.max=apply(yrep.algae.not.NA,1,max)
yrep.algae.median=apply(yrep.algae.not.NA,1,median)

par(mfrow=c(3,2))
hist(yrep.rotifier.min,col="gray40",main="Predictive distribution for min for rotifier")
abline(v=min(rotifier.not.NA),col="red",lwd=2)

hist(yrep.algae.min,col="gray40",main="Predictive distribution for min for algae")
abline(v=min(algae.not.NA),col="red",lwd=2)


hist(yrep.rotifier.max,col="gray40",main="Predictive distribution for max for rotifier")
abline(v=max(rotifier.not.NA),col="red",lwd=2)

hist(yrep.algae.max,col="gray40",main="Predictive distribution for max for algae")
abline(v=max(algae.not.NA),col="red",lwd=2)

hist(yrep.rotifier.median,col="gray40",main="Predictive distribution for median for rotifier")
abline(v=median(rotifier.not.NA),col="red",lwd=2)

hist(yrep.algae.median,col="gray40",main="Predictive distribution for median for algae")
abline(v=median(algae.not.NA),col="red",lwd=2)
```

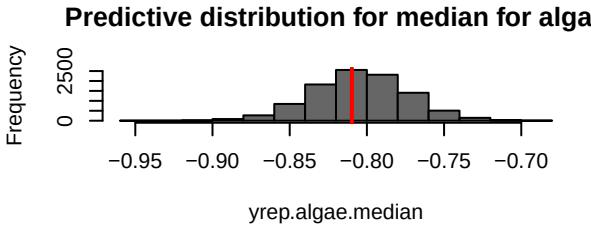












As we can see, the posterior predictive checks did not detect any issues with the model. The values of the functions min, max and median applied on the original data for algae and rotifier were in the typical range of the realizations of the replicates.

```
dic.samples(model1,n.iter=10000)
```

```
## Mean deviance:  -171
## penalty 602.4
## Penalized deviance: 431.4
```

Explanation: We have computed the DIC value using the dic.samples function applied on the JAGS model. The penalized deviance value is 433.7.

d)[10 marks]

Discuss the meaning of the model parameters $\mathbf{A}, \mathbf{b}, \sigma_R^2, \sigma_A^2, \eta_R^2, \eta_A^2$. Find a website or paper that that contains information about rotifiers and unicellular algae (Hint: you can use Google search for this). Using your understanding of the meaning of model parameters and the biological information about these organisms, construct more informative prior distributions for the model parameters. State in your report the source of information and the rationale for your choices of priors.

Re-implement the JAGS model with these new priors. Perform the same posterior predictive checks as in part c) to evaluate the fit of this new model on the data.

Compute the DIC score for the model as well (Hint: lower DIC score indicates better fit on the data).

Discuss whether your new priors have improved the model fit compared to the original prior from a).

Explanation: We kept the same priors as in a) for the initial concentrations x0 (i.e. mean.x0 and prec.x0).

sigma2 corresponds to the variance of the evolution equations for x (i.e. model noise). We decided to change the prior of the precision prec.sigma2=1/sigma2 to Gamma(0.05,0.05), which is more spread out.

eta2 corresponds to the measurement noise (i.e. how precisely the measured concentration y corresponds to the true concentration x). Since this experiment was done very recently (2020) in a German laboratory in controlled conditions, and the paper was published in Nature, it seems plausible that the measurements were rather accurate (i.e. eta2 might be rather small). For this reason we choose a prior Gamma(0.01,0.01) for prec.eta2=1/eta2. This density is more spread out than Gamma(0.1,0.1), meaning that larger precision values are not penalized as much. You can plot the log-density of a Gamma distribution using the following lines,

```
x=(1:1000)/10

plot(x,dgamma(x,shape=0.1,rate=0.1,log=TRUE))
```

b is an additive term in the model equation $\mathbf{x}_t = \mathbf{A}\mathbf{x}_{t-1} + \mathbf{b} + \mathbf{w}_t$, where $\mathbf{x}_t$ contains the log-concentrations. Hence if the two species were not interacting, and $\mathbf{A}$ would be identity, then $\mathbf{b}$ would correspond to the logarithm of the multiplication rate per day.

In the original paper (), it was mentioned that the rotifier is of species B. calyciflorus, while the algae are of species M. minutum or C. vulgaris. Based on a quick Google search, we found the following resource about B. calyciflorus https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1365-2427.1990.tb00295.x This states that the maximal growth rate is 0.8/day. We assume that the typical growth rate is 0.65/day. Regarding green algae, we found the following resource: http://www.jlakes.org/config/hpkx/news_category/2016-03-22/1-s2.0-S1364032115004839-main.pdf This states that some algae species can grow up to a rate of 1.73/day under ideal conditions, but there is a lot of variability based on the conditions. Hence we assume that the typical growth rate is 0.65/day. For this reason, we set the mean of b mean.b as log(1.65)~0.5 for both algae and rotifier. We set the precision of b as 2 for both components.

Finally, the A matrix corresponds to the evolution of x and the predator-prey interaction between rotifier and algae. We assume that A[1,1] and A[2,2] has mean 1, as in an identity matrix (in order to have some level of smoothness in the evolution of log-concentrations at consecutive days). A[1,2] is expressing how much the rotifier concentration is impacted by the algae concentration. We assume that this is positive since the rotifiers are eating the algae. So we set mean 0.2 for it. A[2,1] expresses how much is the algae concentration impacted by the rotifier concentration. We assume that this is negative since the algae are being consumed by the rotifiers. So we set mean -0.2 for it. Finally, we set the precisions at 1 for each element of A to account for the vagueness of the prior information. We re-implement the model with the new priors and do the posterior predictive checks and compute the DIC score.

```
mean.x0=c(log(6),log(1.5))
prec.x0=c(0.25,0.25)
alpha.sigma2=c(0.05,0.05)
beta.sigma2=c(0.05,0.05)
alpha.eta2=c(0.01,0.01)
beta.eta2=c(0.01,0.01)
mean.b=c(0.5,0.5)
prec.b=c(2,2)


mean.A=matrix(c(1,-0.2,0.2,1),nrow=2,ncol=2)
prec.A=matrix(1,nrow=2,ncol=2)


model2.inits=list()
for(it in 1:5)
{
  model2.inits[[it]]<-list(A=A.init+rnorm(4,sd=1),b=b.init+rnorm(2,sd=1),
                           prec.sigma2=prec.sigma2.init+abs(rnorm(2,sd=1)),
```

```
                          prec.eta2=prec.eta2.init+abs(rnorm(2,sd=1)),
                          x=x.init+rnorm(2*n,sd=1))
}



model2.data <- list(n=n,y=y1,mean.x0=mean.x0,prec.x0=prec.x0,
                    mean.b=mean.b,prec.b=prec.b,
                    mean.A=mean.A,prec.A=prec.A,
                              alpha.sigma2=alpha.sigma2,beta.sigma2=beta.sigma2,
                              alpha.eta2=alpha.eta2,beta.eta2=beta.eta2)

# #compiling model
model2=jags.model(textConnection(model_string),data = model2.data,n.chains=5, inits=model2.inits)
```

```
## Compiling model graph
##    Declaring variables
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 735
##    Unobserved stochastic nodes: 1531
##    Total graph size: 4546
##
## Initializing model
```

```
# # Burnin for 10000 samples
update(model2,10000,progress.bar="none")

# # Running the model, monitoring the variable theta

res.model2=coda.samples(model2,variable.names=c("A","b","sigma2","eta2"), n.iter=50000,progress.bar="no
```

```
effectiveSize(res.model2)
```

```
##    A[1,1]    A[2,1]    A[1,2]    A[2,2]      b[1]      b[2]   eta2[1]   eta2[2]
##  2970.258  3074.761  2230.580 23263.609  3013.347  2975.799  2328.081  1396.947
## sigma2[1] sigma2[2]
##  3295.825  1752.001
```

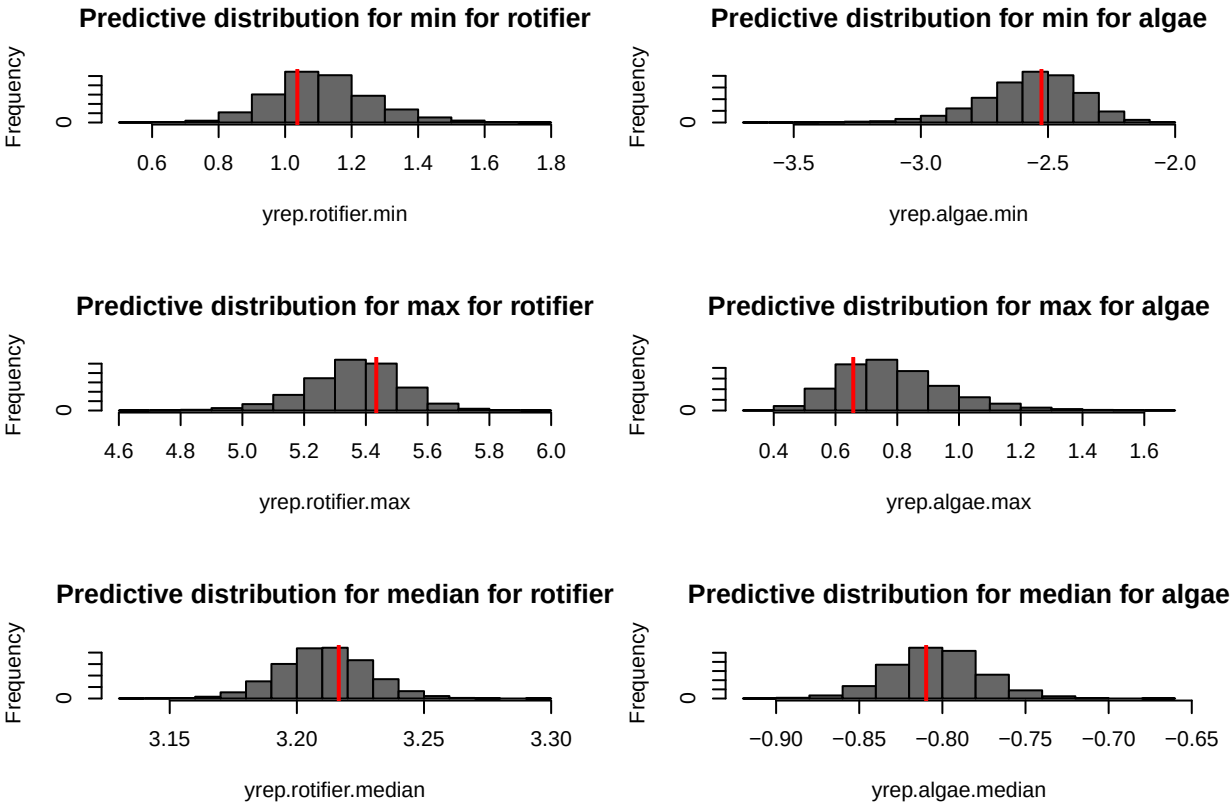
```
summary(res.model2)
```

```
##
## Iterations = 10001:60000
## Thinning interval = 1
## Number of chains = 5
## Sample size per chain = 50000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##               Mean       SD  Naive SE Time-series SE
## A[1,1]     0.55481 0.041001 8.200e-05      0.0007529
## A[2,1]    -0.40573 0.046200 9.240e-05      0.0008337
## A[1,2]     0.53756 0.082880 1.658e-04      0.0017694
## A[2,2]     0.26167 0.054289 1.086e-04      0.0003571
```

```
## b[1]       1.85967 0.132120 2.642e-04      0.0024081
## b[2]       0.68611 0.150977 3.020e-04      0.0027672
## eta2[1]    0.01170 0.006767 1.353e-05      0.0001403
## eta2[2]    0.05963 0.033739 6.748e-05      0.0009088
## sigma2[1]  0.15346 0.018738 3.748e-05      0.0003283
## sigma2[2]  0.19643 0.037260 7.452e-05      0.0008926
##
## 2. Quantiles for each variable:
##
##                2.5%       25%      50%      75%    97.5%
## A[1,1]     0.475468  0.527109  0.55418  0.58220  0.63569
## A[2,1]    -0.497046 -0.436513 -0.40559 -0.37470 -0.31557
## A[1,2]     0.403871  0.477768  0.52776  0.58753  0.72427
## A[2,2]     0.155121  0.225571  0.26140  0.29764  0.36900
## b[1]       1.603315  1.769933  1.85933  1.94926  2.11919
## b[2]       0.389480  0.584734  0.68671  0.78695  0.98295
## eta2[1]    0.003018  0.006703  0.01023  0.01515  0.02855
## eta2[2]    0.007591  0.032079  0.05717  0.08310  0.13050
## sigma2[1]  0.115164  0.141419  0.15390  0.16603  0.18904
## sigma2[2]  0.124877  0.169538  0.19678  0.22380  0.26564
```

```
res.replicates=coda.samples(model2,variable.names=c("y.rep"), n.iter=2000,progress.bar="none")
yrep=as.matrix(res.replicates)
yrep.rotifier=yrep[,1:n]
yrep.algae=yrep[,(n+1):(2*n)]

ind.rotifier.not.NA=which(!is.na(y1[,1]))
ind.algae.not.NA=which(!is.na(y1[,2]))
rotifier.not.NA=y1[ind.rotifier.not.NA,1]
algae.not.NA=y1[ind.algae.not.NA,2]

yrep.rotifier.not.NA=yrep.rotifier[,ind.rotifier.not.NA]
yrep.algae.not.NA=yrep.algae[,ind.algae.not.NA]

yrep.rotifier.min=apply(yrep.rotifier.not.NA,1,min)
yrep.rotifier.max=apply(yrep.rotifier.not.NA,1,max)
yrep.rotifier.median=apply(yrep.rotifier.not.NA,1,median)

yrep.algae.min=apply(yrep.algae.not.NA,1,min)
yrep.algae.max=apply(yrep.algae.not.NA,1,max)
yrep.algae.median=apply(yrep.algae.not.NA,1,median)

par(mfrow=c(3,2))
hist(yrep.rotifier.min,col="gray40",main="Predictive distribution for min for rotifier")
abline(v=min(rotifier.not.NA),col="red",lwd=2)

hist(yrep.algae.min,col="gray40",main="Predictive distribution for min for algae")
abline(v=min(algae.not.NA),col="red",lwd=2)


hist(yrep.rotifier.max,col="gray40",main="Predictive distribution for max for rotifier")
abline(v=max(rotifier.not.NA),col="red",lwd=2)

hist(yrep.algae.max,col="gray40",main="Predictive distribution for max for algae")
```

```
abline(v=max(algae.not.NA),col="red",lwd=2)

hist(yrep.rotifier.median,col="gray40",main="Predictive distribution for median for rotifier")
abline(v=median(rotifier.not.NA),col="red",lwd=2)

hist(yrep.algae.median,col="gray40",main="Predictive distribution for median for algae")
abline(v=median(algae.not.NA),col="red",lwd=2)
```



As we can see, as with the prior in a), no issues with the model have been detected by these posterior predictive checks.

```
dic.samples(model2,n.iter=10000)
```

```
## Mean deviance:  -705.6
## penalty 1020
## Penalized deviance: 314.2
```

We have obtained a significantly smaller value (371.6) for the DIC score of this model compared to the original model in a), indicating that the informative prior distributions resulted in better fit on the data.

e)[10 marks] Update the model with your informative prior in part d) to compute the posterior distribution of the log concentrations sizes (x_t) on the days 376-395 (20 additional days).

Plot the evolution of the posterior mean of the log concentrations for rotifier and algae during days 376-395 on a single plot, along with curves that correspond to the [2.5%, 97.5%] credible interval of the log concentration size (x_t) **according to the posterior distribution at each year [Hint: you need**** $2 + 2 \cdot 2 = 6$ **curves in total, use different colours for the curves for rotifier and algae].

Finally, estimate the posterior probability that the concentration of algae (measured in 10^6 algae/ml, as in the data) becomes smaller than 0.1 **at any time during these 20 additional days (days 376-395).**

[8 marks] Explanation: We included the additional days in the dataset as NAs (y3 variable). We re-run the model from part d) on this new dataset, and obtained the samples from the log-concentration x, which contain the posterior samples for the 20 additional days. We have plotted the evolution of the mean and the 95% credible intervals.

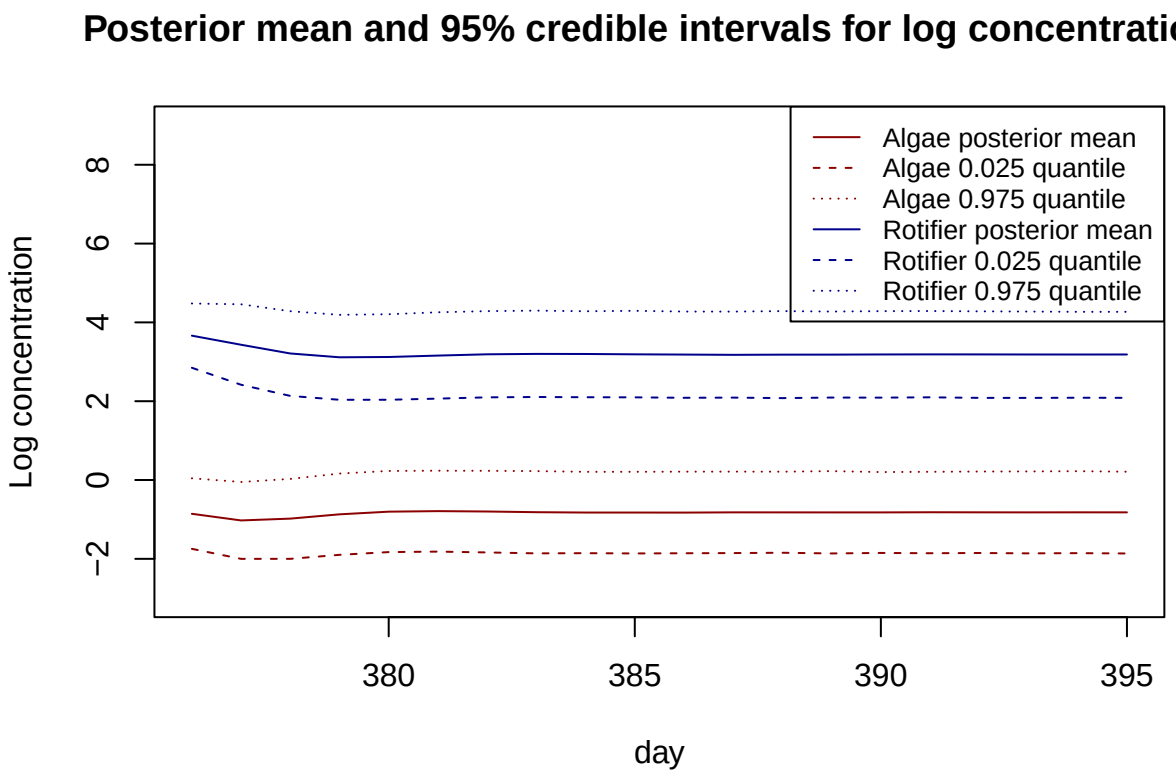
```
mean.x0=c(log(6),log(1.5))
prec.x0=c(0.25,0.25)
alpha.sigma2=c(0.05,0.05)
beta.sigma2=c(0.05,0.05)
alpha.eta2=c(0.01,0.01)
beta.eta2=c(0.01,0.01)
mean.b=c(0.5,0.5)
prec.b=c(2,2)


mean.A=matrix(c(1,-0.2,0.2,1),nrow=2,ncol=2)
prec.A=matrix(1,nrow=2,ncol=2)



n=nrow(rotifier_algae)+1
npred=20

y3=matrix(NA,nrow=(n+npred),ncol=2)

y3[2:n,1]=log(rotifier_algae$rotifier)
y3[2:n,2]=log(rotifier_algae$algae)


x.init=y3
require(imputeTS)
x.init[,1]=na_interpolation(x.init[,1])
x.init[,2]=na_interpolation(x.init[,2])
#Initializing the Markov chain based on interpolation


A.init=matrix(0,nrow=2,ncol=2)
b.init=rep(0.5,2)
prec.sigma2.init=rep(1,2)
prec.eta2.init=rep(1,2)


model3.inits=list()
for(it in 1:5)
{
  model3.inits[[it]]<-list(A=A.init+rnorm(4,sd=1),b=b.init+rnorm(2,sd=1),
                           prec.sigma2=prec.sigma2.init+abs(rnorm(2,sd=1)),
                           prec.eta2=prec.eta2.init+abs(rnorm(2,sd=1)),
                           x=x.init+rnorm(2*(n+npred),sd=1))
}



model3.data <- list(n=(n+npred),y=y3,mean.x0=mean.x0,prec.x0=prec.x0,
```

```
                    mean.b=mean.b,prec.b=prec.b,
                    mean.A=mean.A,prec.A=prec.A,
                              alpha.sigma2=alpha.sigma2,beta.sigma2=beta.sigma2,
                              alpha.eta2=alpha.eta2,beta.eta2=beta.eta2)

# #compiling model
model3=jags.model(textConnection(model_string),data = model3.data,n.chains=5, inits=model3.inits)
```

```
## Compiling model graph
##    Declaring variables
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 735
##    Unobserved stochastic nodes: 1651
##    Total graph size: 4786
##
## Initializing model
```

```
# # Burnin for 10000 samples
update(model3,10000,progress.bar="none")

# # Running the model for 50000 iterations, monitoring x

res.model3=coda.samples(model3,variable.names=c("x"), n.iter=10000,progress.bar="none")
```

```
xres=as.matrix(res.model3)
rotifier.predict=xres[,(n+1):(n+npred)]
algae.predict=xres[,(n+npred+n+1):(2*(n+npred))]

#We combine the results from all chains into a single dataframe

algae.mean=apply(algae.predict,MARGIN=2, FUN=mean)
algae.q025=apply(algae.predict, MARGIN=2, FUN=function(x) quantile(x,prob=0.025))
algae.q975=apply(algae.predict, MARGIN=2, FUN=function(x) quantile(x,prob=0.975))

rotifier.mean=apply(rotifier.predict,MARGIN=2, FUN=mean)
rotifier.q025=apply(rotifier.predict, MARGIN=2, FUN=function(x) quantile(x,prob=0.025))
rotifier.q975=apply(rotifier.predict, MARGIN=2, FUN=function(x) quantile(x,prob=0.975))


plot((n):(n+npred-1), algae.mean,type="l",main="Posterior mean and 95% credible intervals for log concer
lines((n):(n+npred-1), algae.q025,lty=2,col="dark red")
lines((n):(n+npred-1), algae.q975,lty=3,col="dark red")

lines((n):(n+npred-1), rotifier.mean,lty=1,col="dark blue")
lines((n):(n+npred-1), rotifier.q025,lty=2,col="dark blue")
lines((n):(n+npred-1), rotifier.q975,lty=3,col="dark blue")
legend("topright", legend=c("Algae posterior mean", "Algae 0.025 quantile", "Algae 0.975 quantile", "Ro
       col=c("dark red", "dark red", "dark red", "dark blue", "dark blue", "dark blue"), lty=c(1,2,3,1,
```



[2 marks] Explanation: We computed the posterior probability that algae concentration drops below 0.1e6 algae/ml in the next 20 days using the MCMC samples from x. This probability is approximately 0.05.

```
algae.min=apply(algae.predict,MARGIN=1, FUN=min)
prob.below.0.1=mean(algae.min<log(0.1))
cat("Probability of algae concentration dropping below 0.1e6 algae/ml during next 20 days:",prob.below.
```

```
## Probability of algae concentration dropping below 0.1e6 algae/ml during next 20 days: 0.05454
```

Problem 2 - Horse racing data

In this problem, we are going to construct a predictive model for horse races. The dataset (races.csv and runs.csv) contains the information about 1000 horse races in Hong Kong during the years 1997-1998 (originally from https://www.kaggle.com/gdaley/hkracing). Races.csv contains information about each race (such as distance, venue, track conditions, etc.), while runs.csv contains information about each horse participating in each race (such as finish time in the race). Detailed description of all columns in these files is available in the file horse_racing_data_info.txt.

Our goal is to model the mean speed of each horse during the races based on covariates available before the race begins.

We are going to use INLA to fit several different regression models to this dataset. First, we load ILNA and the datasets and display the first few rows.

```
library(INLA)
```

```
## Loading required package: Matrix
```

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

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

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

```
## This is INLA_21.12.18 built 2021-12-18 19:24:10 UTC.
##  - See www.r-inla.org/contact-us for how to get help.
##  - To enable PARDISO sparse library; see inla.pardiso()
```

```
#If it loaded correctly, you should see this in the output:
#Loading required package: Matrix
#Loading required package: foreach
#Loading required package: parallel
#Loading required package: sp
#This is INLA_21.11.22 built 2021-11-21 16:13:28 UTC.
# - See www.r-inla.org/contact-us for how to get help.
# - To enable PARDISO sparse library; see inla.pardiso()

#The following code does the full installation. You can try it if INLA has not been installed.
#First installing some of the dependencies
#install.packages("BiocManager")
#BiocManager::install("Rgraphviz")
#if (!requireNamespace("BiocManager", quietly = TRUE))
#    install.packages("BiocManager")
#BiocManager::install("graph")
#Installing INLA
#install.packages("INLA",repos=c(getOption("repos"),INLA="https://inla.r-inla-download.org/R/stable"),
#library(INLA)
```

```
runs <- read.csv(file = 'runs.csv')
head(runs)
```

```
##   race_id horse_no horse_id result won lengths_behind horse_age horse_country
## 1       0        1     3917     10   0           8.00         3           AUS
## 2       0        2     2157      8   0           5.75         3            NZ
## 3       0        3      858      7   0           4.75         3            NZ
## 4       0        4     1853      9   0           6.25         3           SAF
## 5       0        5     2796      6   0           3.75         3            GB
## 6       0        6     3296      3   0           1.25         3            NZ
##   horse_type horse_rating horse_gear declared_weight actual_weight draw
```

```
## 1    Gelding           60         --            1020           133    7
## 2    Gelding           60         --             980           133   12
## 3    Gelding           60         --            1082           132    8
## 4    Gelding           60         --            1118           127   13
## 5    Gelding           60         --             972           131   14
## 6    Gelding           60         --            1114           127    5
##   position_sec1 position_sec2 position_sec3 position_sec4 position_sec5
## 1             6             4             6            10            NA
## 2            12            13            13             8            NA
## 3             3             2             2             7            NA
## 4             8             8            11             9            NA
## 5            13            12            12             6            NA
## 6            11            11             5             3            NA
##   position_sec6 behind_sec1 behind_sec2 behind_sec3 behind_sec4 behind_sec5
## 1            NA        2.00        2.00        1.50        8.00          NA
## 2            NA        6.50        9.00        5.00        5.75          NA
## 3            NA        1.00        1.00        0.75        4.75          NA
## 4            NA        3.50        5.00        3.50        6.25          NA
## 5            NA        7.75        8.75        4.25        3.75          NA
## 6            NA        5.00        7.75        1.25        1.25          NA
##   behind_sec6 time1 time2 time3 time4 time5 time6 finish_time win_odds
## 1          NA 13.85 21.59 23.86 24.62    NA    NA       83.92      9.7
## 2          NA 14.57 21.99 23.30 23.70    NA    NA       83.56     16.0
## 3          NA 13.69 21.59 23.90 24.22    NA    NA       83.40      3.5
## 4          NA 14.09 21.83 23.70 24.00    NA    NA       83.62     39.0
## 5          NA 14.77 21.75 23.22 23.50    NA    NA       83.24     50.0
## 6          NA 14.33 22.03 22.90 23.57    NA    NA       82.83      7.0
##   place_odds trainer_id jockey_id
## 1        3.7        118         2
## 2        4.9        164        57
## 3        1.5        137        18
## 4       11.0         80        59
## 5       14.0          9       154
## 6        1.8         54        34
```

```
races<- read.csv(file = 'races.csv')
head(races)
```

```
##   race_id       date venue race_no config surface distance        going
## 1       0 1997-06-02    ST       1      A       0     1400 GOOD TO FIRM
## 2       1 1997-06-02    ST       2      A       0     1200 GOOD TO FIRM
## 3       2 1997-06-02    ST       3      A       0     1400 GOOD TO FIRM
## 4       3 1997-06-02    ST       4      A       0     1200 GOOD TO FIRM
## 5       4 1997-06-02    ST       5      A       0     1600 GOOD TO FIRM
## 6       5 1997-06-02    ST       6      A       0     1200 GOOD TO FIRM
##   horse_ratings   prize race_class sec_time1 sec_time2 sec_time3 sec_time4
## 1         40-15  485000          5     13.53     21.59     23.94     23.58
## 2         40-15  485000          5     24.05     22.64     23.70        NA
## 3         60-40  625000          4     13.77     22.22     24.88     22.82
## 4        120-95 1750000          1     24.33     22.47     22.09        NA
## 5         60-40  625000          4     25.45     23.52     23.31     23.56
## 6         60-40  625000          4     23.47     22.48     23.25        NA
##   sec_time5 sec_time6 sec_time7 time1 time2 time3 time4 time5 time6 time7
## 1        NA        NA        NA 13.53 35.12 59.06 82.64    NA    NA    NA
## 2        NA        NA        NA 24.05 46.69 70.39    NA    NA    NA    NA
```

```
## 3        NA        NA        NA 13.77 35.99 60.87 83.69    NA    NA    NA
## 4        NA        NA        NA 24.33 46.80 68.89    NA    NA    NA    NA
## 5        NA        NA        NA 25.45 48.97 72.28 95.84    NA    NA    NA
## 6        NA        NA        NA 23.47 45.95 69.20    NA    NA    NA    NA
##   place_combination1 place_combination2 place_combination3 place_combination4
## 1                  8                 11                  6                 NA
## 2                  5                 13                  4                 NA
## 3                 11                  1                 13                 NA
## 4                  5                  3                 10                 NA
## 5                  2                 10                  1                 NA
## 6                  9                 14                  8                 NA
##   place_dividend1 place_dividend2 place_dividend3 place_dividend4
## 1            36.5            25.5            18.0              NA
## 2            12.5            47.0            33.5              NA
## 3            23.0            23.0            59.5              NA
## 4            14.0            24.5            16.0              NA
## 5            15.5            28.0            17.5              NA
## 6            16.5           408.0            70.0              NA
##   win_combination1 win_dividend1 win_combination2 win_dividend2
## 1                8         121.0               NA            NA
## 2                5          23.5               NA            NA
## 3               11          70.0               NA            NA
## 4                5          52.0               NA            NA
## 5                2          36.5               NA            NA
## 6                9          61.0               NA            NA
```

a)[10 marks] Create a dataframe that includes the mean speed of each horse in each race and the distance of the race in a column [Hint: you can do this adding two extra columns to the runs dataframe].

Fit a linear regression model (lm) with the mean speed as a response variable. The covariates should be the horse id as a categorical variable, and the race distance, horse rating, and horse age as standard variable. Scale the non-categorical covariates before fitting the model (i.e. center and divide by their standard deviation, you can use the `scale` function in R for this).

Print out the summary of the lm model, discuss the quality of the fit.

Explanation: The runs dataframe contains information about each horse in each race, while the races dataframe contains the information about the race (such as distance, location, etc.) In order to be able to compute the mean speed=finish_time/distance, we first create a distance column in the runs dataframe using the line

```
runs$distance=races$distance[runs$race_id+1]
```

The variable `runs$race_id`is the id of the race that particular run happened, and since the id is starting from 0, we need to add 1 to get the row number in the races dataframe. Once the distance has been saved as a column, the mean speed can be easily computed. We fit the linear model as required in the question.

```
#creating a distance column by the distance of the race for that particular run
runs$distance=races$distance[runs$race_id+1]
#creating a column for mean speed
runs$speed=runs$distance/runs$finish_time

#Creating the dataframe with y as speed, horse_id as a factor variable, and the scaled version of the d
horse=data.frame(y=runs$speed, horse_id=as.factor(runs$horse_id), distance=scale(runs$distance), rating=
#Fitting the linear model
```

```r
m.lm=lm(y~horse_id+distance+rating+age,data=horse)

options(max.print=100)
summary(m.lm)
```

```
##
## Call:
## lm(formula = y ~ horse_id + distance + rating + age, data = horse)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.64378 -0.09259  0.01438  0.11849  0.59029
##
## Coefficients:
##                Estimate Std. Error  t value Pr(>|t|)
## (Intercept)   1.669e+01  4.461e-02  374.039  < 2e-16 ***
## horse_id29   -1.449e-01  7.197e-02   -2.013 0.044162 *
## horse_id61   -2.604e-01  1.114e-01   -2.338 0.019404 *
## horse_id62   -9.859e-02  1.016e-01   -0.970 0.331919
## horse_id63   -8.021e-02  8.924e-02   -0.899 0.368752
## horse_id64    6.383e-02  1.510e-01    0.423 0.672519
## horse_id65   -1.070e-01  1.015e-01   -1.054 0.292048
## horse_id66   -6.092e-02  8.905e-02   -0.684 0.493931
## horse_id67   -7.032e-02  8.490e-02   -0.828 0.407527
## horse_id69   -1.243e+00  2.087e-01   -5.956 2.67e-09 ***
## horse_id70    1.308e-01  1.041e-01    1.257 0.208848
## horse_id72   -8.189e-03  7.200e-02   -0.114 0.909452
## horse_id79   -2.675e-01  1.258e-01   -2.127 0.033463 *
## horse_id80    3.551e-02  7.831e-02    0.453 0.650228
## horse_id82    1.316e-01  6.547e-02    2.010 0.044455 *
## horse_id91   -2.694e-02  1.016e-01   -0.265 0.790933
## horse_id92    2.178e-01  1.016e-01    2.144 0.032079 *
## horse_id93    7.120e-02  6.650e-02    1.071 0.284323
## horse_id94   -2.626e-02  8.896e-02   -0.295 0.767818
## horse_id95   -1.136e-01  6.665e-02   -1.704 0.088333 .
##  [ reached getOption("max.print") -- omitted 1415 rows ]
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2038 on 11109 degrees of freedom
## Multiple R-squared:  0.7883, Adjusted R-squared:  0.761
## F-statistic: 28.85 on 1434 and 11109 DF,  p-value: < 2.2e-16
```

The model fit is quite good, with multiple R-squared being 0.7883. This means that 78.83% of the variance of the response variable is explained by this linear model.

b)[10 marks] Fit the same model in INLA (i.e. Bayesian linear regression with Gaussian likelihood, mean speed is the response variable, and the same covariates used with scaling for the non-categorical covariates). Set a Gamma (0.1,0.1) prior for the precision, and Gaussian priors with mean zero and variance 1000000 for all of the regression coefficients (including the intercept).

Print out the summary of the INLA model. Compute the posterior mean of the variance parameter σ^2. Plot the posterior density for the variance parameter σ^2. Compute the negative sum log CPO (NSLCPO) and DIC values for this model (smaller values indicate better fit).

Compute the standard deviation of the mean residuals (i.e. the differences between the posterior mean of the fitted values and the true response variable).

Discuss the results.

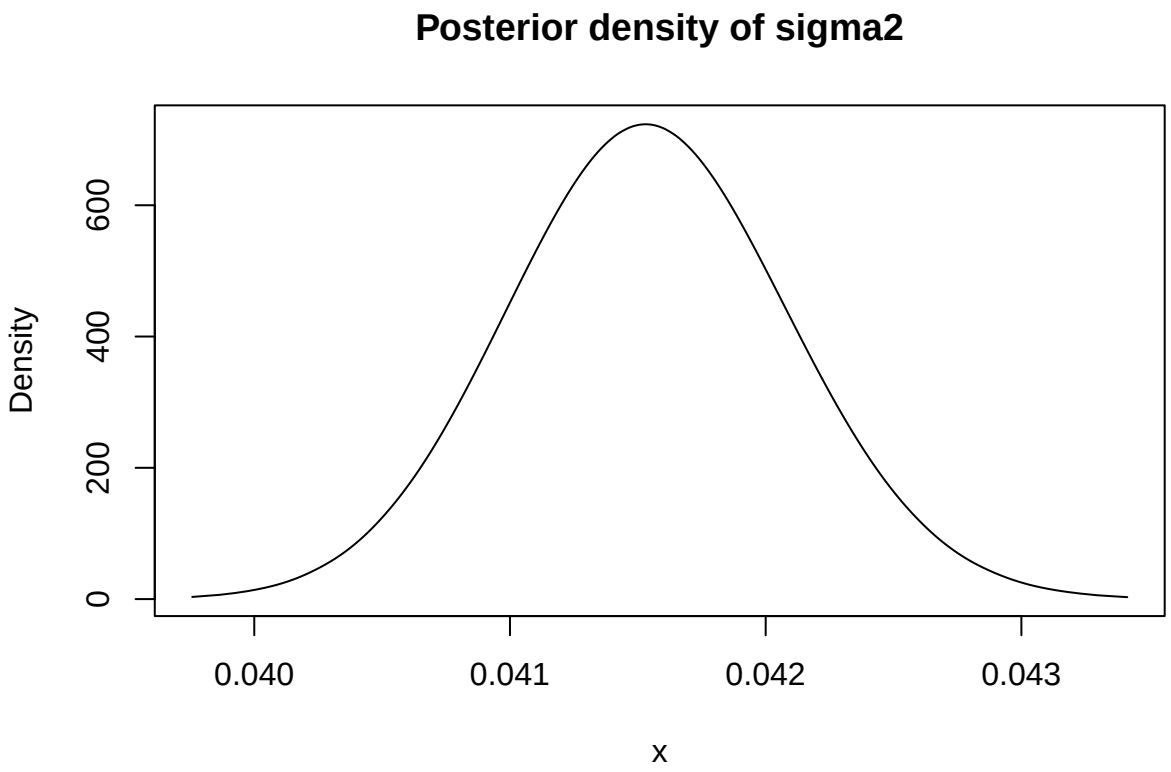
[5 marks] Explanation: We have fitted the same model in INLA with default priors. We ensured that CPO and DIC are computed using the control.compute option. We printed out he summary of the model below.

```
horse=data.frame(y=runs$speed, horse_id=as.factor(runs$horse_id), distance=scale(runs$distance), rating=
m.I=inla(y~horse_id+distance+rating+age,data=horse,
         family="Gaussian",control.predictor = list(compute = TRUE),
         control.compute = list(cpo=TRUE,dic=TRUE,config = TRUE))
options(max.print=100)
summary(m.I)
```

```
##
## Call:
##    c("inla(formula = y ~ horse_id + distance + rating + age, family =
##    \"Gaussian\", ", " data = horse, control.compute = list(cpo = TRUE, dic
##    = TRUE, ", " config = TRUE), control.predictor = list(compute = TRUE))"
##    )
## Time used:
##     Pre = 20.9, Running = 32.4, Post = 17.6, Total = 70.8
## Fixed effects:
##                mean    sd 0.025quant 0.5quant 0.975quant   mode kld
## (Intercept)  16.687 0.045     16.599   16.687     16.774 16.687   0
## horse_id29   -0.145 0.072     -0.286   -0.145     -0.004 -0.145   0
## horse_id61   -0.260 0.111     -0.479   -0.260     -0.042 -0.260   0
## horse_id62   -0.099 0.102     -0.298   -0.099      0.101 -0.099   0
## horse_id63   -0.080 0.089     -0.255   -0.080      0.095 -0.080   0
## horse_id64    0.064 0.151     -0.233    0.064      0.360  0.064   0
## horse_id65   -0.107 0.101     -0.306   -0.107      0.092 -0.107   0
## horse_id66   -0.061 0.089     -0.236   -0.061      0.114 -0.061   0
## horse_id67   -0.070 0.085     -0.237   -0.070      0.096 -0.070   0
## horse_id69   -1.243 0.209     -1.653   -1.243     -0.833 -1.243   0
## horse_id70    0.131 0.104     -0.073    0.131      0.335  0.131   0
## horse_id72   -0.008 0.072     -0.149   -0.008      0.133 -0.008   0
## horse_id79   -0.267 0.126     -0.514   -0.267     -0.021 -0.267   0
## horse_id80    0.036 0.078     -0.118    0.036      0.189  0.036   0
##  [ reached getOption("max.print") -- omitted 1421 rows ]
##
## Model hyperparameters:
##                                          mean    sd 0.025quant 0.5quant
## Precision for the Gaussian observations 24.08 0.318      23.45    24.07
##                                         0.975quant  mode
## Precision for the Gaussian observations      24.71 24.07
##
## Deviance Information Criterion (DIC) ...............: -2942.84
## Deviance Information Criterion (DIC, saturated) ....: 415240.76
## Effective number of parameters .....................: 1398.81
##
## Marginal log-Likelihood:  -5718.98
## CPO and PIT are computed
##
## Posterior summaries for the linear predictor and the fitted values are computed
## (Posterior marginals needs also 'control.compute=list(return.marginals.predictor=TRUE)')
```

[2 marks] After this, we have plotted the density of sigma2 (using inla.tmarginal). The posterior mean is 0.0415397 based on the summary created by inla.zmarginal.

```
marg.sigma2 <- inla.tmarginal(function(tau) tau^(-1),
  m.I$marginals.hyperpar$`Precision for the Gaussian observations`)
plot(marg.sigma2,type ="l",xlab="x",ylab="Density",
main='Posterior density of sigma2')
```




```
#Summary statistics of sigma
cat("Summary statistics of sigma2:\n")
```

```
## Summary statistics of sigma2:
```

```
inla.zmarginal(marg.sigma2)
```

```
## Mean            0.0415424
## Stdev           0.00055328
## Quantile  0.025 0.0404667
## Quantile  0.25  0.0411651
## Quantile  0.5   0.0415372
## Quantile  0.75  0.041913
## Quantile  0.975 0.0426411
```

[3 marks] Finally, we have evaluated the DIC, NLSCPO and the standard deviation of the mean residuals.

```
cat("DIC of model 1:",m.I$dic$dic,"\n")
```

```
## DIC of model 1: -2942.836
```

```
cat("NSLCPO of model 1:",-sum(log(m.I$cpo$cpo)),"\n")
```

```
## NSLCPO of model 1: -1231.789
```

```
cat("Standard deviation of mean residuals for model 1:",sd(horse$y-m.I$summary.fitted.values$mean),"\n")
```

```
## Standard deviation of mean residuals for model 1: 0.1918068
```

c)[10 marks] In this question, we are going to improve the model in b) by using more informative priors and more columns from the dataset.

First, using some publicly available information from the internet (Hint: use Google search) find out about the typical speed of race horses in Hong Kong, and use this information to construct a prior for the intercept. Explain the rationale for your choice.

Second, look through all of the information in the datasets that is available before the race (Hint: you need to read the description horse_racing_data_info.txt for information about the columns. position, behind, result, won, and time related columns are not available before the race). Discuss your rationale for including some of these in the dataset (make sure to scale them if they are non-categorical).

Feel free to try creating additional covariates such as polynomial or interaction terms (Hint: this can be done using I() in the formula), and you can also try to use a different likelihood (such as Student-t distribution).

Fit your new model in INLA (i.e. Bayesian linear regression, mean speed is the response variable, and scaling done for the non-categorical covariates).

Print out the summary of the INLA model. Compute the negative sum log CPO (NSLCPO) and DIC values for this model (smaller values indicate better fit).

Compute the standard deviation of the mean residuals (i.e. the differences between the posterior mean of the fitted values and the true response variable).

Discuss the results and compare your model to the model from b).

Please only include your best performing model in the report.

Explanation: After doing a Google search about Hong Kong horse races, we have found the paper http://www.bjll.org/index.php/jpm/article/view/590. In Table 2, it is mentioned that the average speed of race horses during the period 2005-2009 in Hong Kong was 16.6 m/s. Hence we use this value as the prior mean for our intercept, and set the precision as 1. We set the prior mean of the other regression variables at 0, and the precision of as 10 (i.e. our expectation is that the regression coefficients are not more than 1 in absolute value, since we are working with standardized or categorical covariates). Our model has been changed compared to part b) by using a categorical version of the distance covariate (because this only takes 9 different values in the dataset, see length(unique(races$distance))). In addition to this change, we also included the venue, surface, going, race_class, config and the logarithm of the prize covariates from the races dataset.

```
prior.prec <- list(prec=list(prior = "loggamma", param = c(0.1, 0.1)))

prior.beta <- list(mean.intercept = 16.6, prec.intercept = 1,
                   mean = 0, prec = 10)
runs$venue=races$venue[runs$race_id+1]
runs$surface=races$surface[runs$race_id+1]
runs$going=races$going[runs$race_id+1]
runs$race_class=races$race_class[runs$race_id+1]
runs$config=races$config[runs$race_id+1]
runs$prize=races$prize[runs$race_id+1]
horse=data.frame(y=runs$speed, horse_id=as.factor(runs$horse_id), distance=as.factor(runs$distance),
                 rating=scale(runs$horse_rating), age=scale(runs$horse_age), surface=as.factor(runs$sur
                 race_class=as.factor(runs$race_class), config=as.factor(runs$config),log.prize=scale(l
m.I2=inla(y~horse_id+distance+rating+age+surface+going+race_class+config+log.prize,data=horse,
         family="Gaussian",control.predictor = list(compute = TRUE),
```

```
        control.compute = list(cpo=TRUE,dic=TRUE,config = TRUE),
        control.family=list(hyper=prior.prec),control.fixed=prior.beta)
options(max.print=100)
summary(m.I2)
```

```
##
## Call:
##    c("inla(formula = y ~ horse_id + distance + rating + age + surface + ",
##    " going + race_class + config + log.prize, family = \"Gaussian\", ", "
##    data = horse, control.compute = list(cpo = TRUE, dic = TRUE, ", "
##    config = TRUE), control.predictor = list(compute = TRUE), ", "
##    control.family = list(hyper = prior.prec), control.fixed = prior.beta)"
##    )
## Time used:
##     Pre = 20.9, Running = 26, Post = 18, Total = 64.9
## Fixed effects:
##                          mean    sd 0.025quant 0.5quant 0.975quant   mode kld
## (Intercept)           17.227 0.030     17.168   17.227     17.287 17.227   0
## horse_id29            -0.134 0.047     -0.227   -0.134     -0.042 -0.134   0
## horse_id61            -0.129 0.084     -0.293   -0.129      0.035 -0.129   0
## horse_id62             0.024 0.075     -0.123    0.024      0.171  0.024   0
## horse_id63             0.094 0.065     -0.033    0.094      0.221  0.094   0
## horse_id64             0.111 0.112     -0.109    0.111      0.330  0.111   0
## horse_id65             0.046 0.074     -0.100    0.046      0.191  0.046   0
## horse_id66             0.017 0.063     -0.107    0.017      0.141  0.017   0
## horse_id67             0.023 0.060     -0.095    0.023      0.141  0.023   0
## horse_id69            -0.874 0.149     -1.166   -0.874     -0.581 -0.874   0
## horse_id70             0.151 0.075      0.004    0.151      0.298  0.151   0
## horse_id72             0.030 0.047     -0.062    0.030      0.121  0.030   0
## horse_id79            -0.019 0.093     -0.200   -0.019      0.163 -0.019   0
## horse_id80             0.001 0.053     -0.104    0.001      0.105  0.001   0
##  [ reached getOption("max.print") -- omitted 1451 rows ]
##
## Model hyperparameters:
##                                          mean    sd 0.025quant 0.5quant
## Precision for the Gaussian observations 36.16 0.478      35.22    36.15
##                                         0.975quant  mode
## Precision for the Gaussian observations      37.11 36.14
##
## Deviance Information Criterion (DIC) ...............: -8209.29
## Deviance Information Criterion (DIC, saturated) ....: 409974.31
## Effective number of parameters .....................: 1291.71
##
## Marginal log-Likelihood:  2768.46
## CPO and PIT are computed
##
## Posterior summaries for the linear predictor and the fitted values are computed
## (Posterior marginals needs also 'control.compute=list(return.marginals.predictor=TRUE)')
```

```
cat("DIC of model 1:",m.I$dic$dic,"\n")
```

```
## DIC of model 1: -2942.836
```

```
cat("NSLCPO of model 1:",-sum(log(m.I$cpo$cpo)),"\n")
```

```
## NSLCPO of model 1: -1231.789
```

```
cat("Standard deviation of mean residuals for model 2:",sd(horse$y-m.I2$summary.fitted.values$mean),"\n
```

```
## Standard deviation of mean residuals for model 2: 0.1568633
```

We have computed the DIC, NLSCPO and the standard deviation of the mean residuals. All of these indicate better model fit and higher accuracy compared to the original model.

d)[10 marks] We are going to perform model checks to evaluate the fit the two models in parts b) and c) on the data.

Compute the studentized residuals for the Bayesian regression model from parts b) and c). Perform a simple Q-Q plot on the studentized residuals. Plot the studentized residuals versus their index, and also plot the studentized residuals against the posterior mean of the fitted value (see Lecture 2). Discuss the results.

Explanation: We have implemented the studentization using dummy variables to encode categorical covariates. This is possible to implement using the dummies library in R. We did perform the required plots firstly for model from part b). We only computed the diagonal elements for the H matrix as these are the only ones required for this problem. We do have some elements in the diagonal of H that are 1, due to those categorical variables (horses) only appearing once in the dataset. Studentization is not appropriate in such cases (as it would mean division by 0), so we change H[i,i] from 1 to 0 for these positions.

```
library(dummies)
```

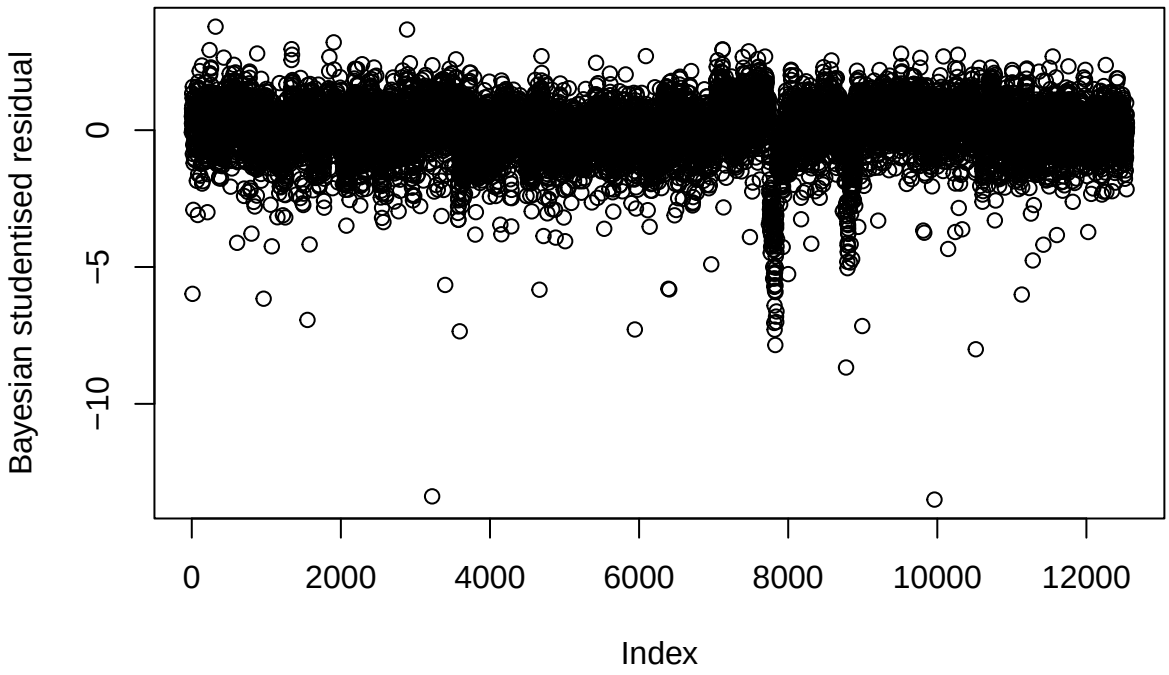
```
## dummies-1.5.6 provided by Decision Patterns
```

```
#If not installed, run install.packages(dummies) first
library(Matrix)

horse_id_dummies=dummy(horse$horse_id)
```

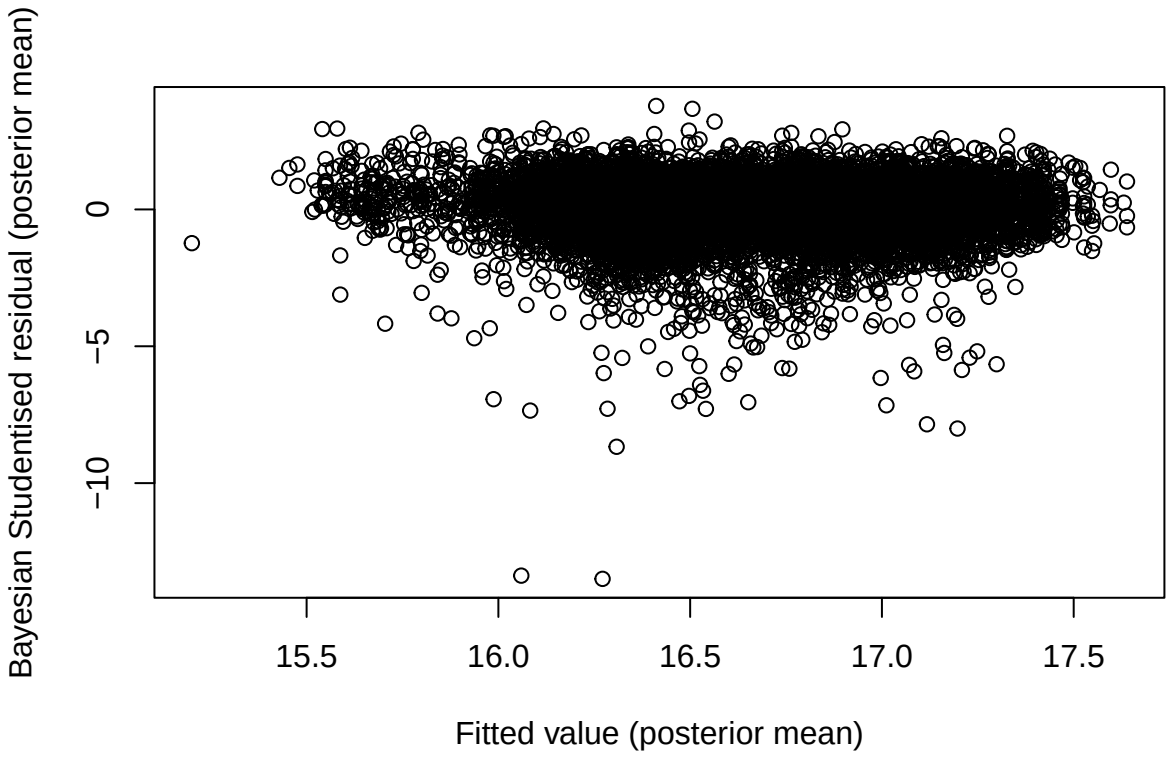
```
## Warning in model.matrix.default(~x - 1, model.frame(~x - 1), contrasts = FALSE):
## non-list contrasts argument ignored
```

```
horse_id_dummies_except_first=horse_id_dummies[,2:ncol(horse_id_dummies)]

nbsamp=1000
samp <- inla.posterior.sample(nbsamp, m.I)


sigma=1/sqrt(inla.posterior.sample.eval(function(...) {theta},
  samp))

#In this model the link function is the identity, so fitted values are the same as the linear predictor
#(E(y_i|x,theta)=mu_i=eta_i)

fittedvalues=inla.posterior.sample.eval(function(...) {Predictor},
samp)

n=nrow(horse)

#Storing x as a sparse matrix, with categorical variables encoded as dummies
#Using sparse matrices can speed up calculations in this example
x=Matrix(cbind(rep(1,n), horse_id_dummies_except_first,horse$distance,horse$rating,horse$age),sparse=TRU
A=solve(t(x)%*%x,t(x))
Hdiag=apply(x*t(A),MARGIN=1,sum)
```

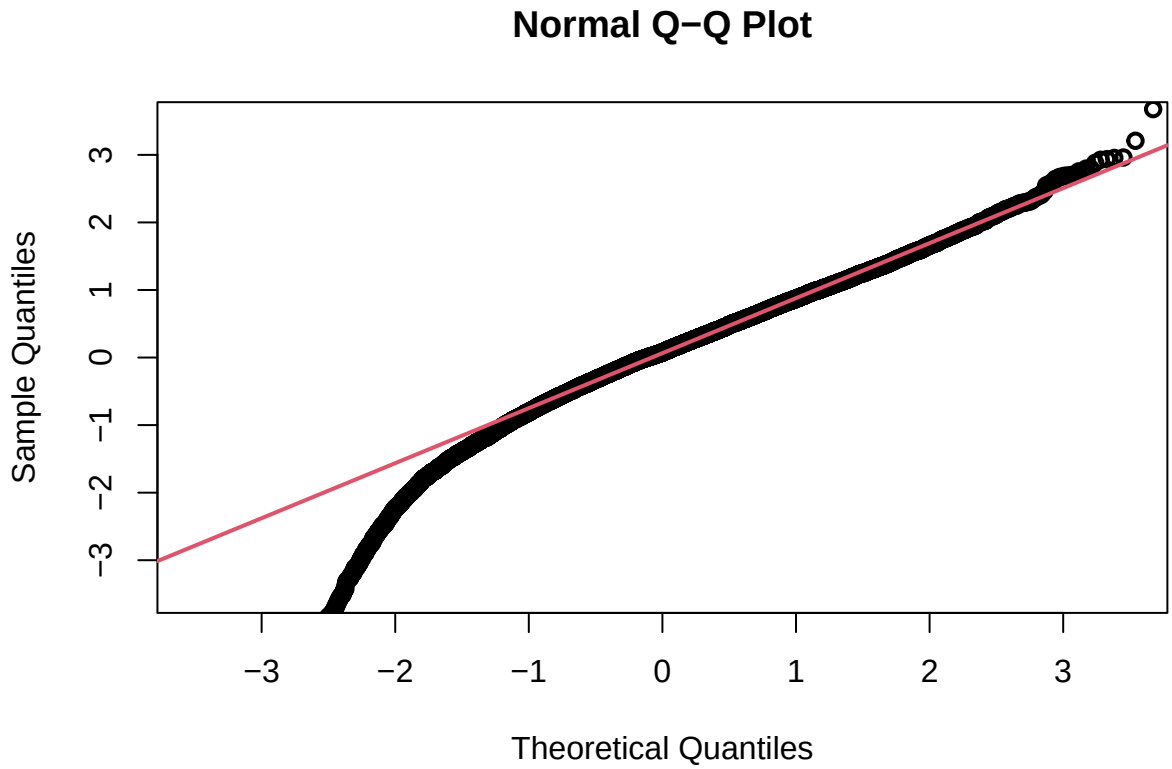
```
#We do have some elements in H[i,i] that are 1, due to those categorical variables (horses) only appear
#Studentization is not appropriate in such cases (as it would mean division by 0), so we change H[i,i] 
Hdiag[Hdiag==1]=0

#Previously, we have computed the whole H matrix using the formula
#H=x%*%solve(t(x)%*%x,t(x))
#This can be quite slow in this example, and also use a lot of memory when the number of data points is
#Since we only use the diagonal terms, it suffices to only compute those,
#meaning that the memory usage and computational time can be significantly improved.
#The diagonal terms H[i,i] can be computed by the scalar product of the ith row of x and the ith column
#This is computed for each i and the results are stored in a vector using the apply(x*t(A),MARGIN=1,sum
#Here MARGIN=1 tells the apply function to evalue the sum function along the rows (MARGIN=2 would corre

#studentised residuals
#n is the number of rows in the dataset, i.e. the number of observations
#studentised residuals
studentisedred=matrix(0,nrow=n,ncol=nbsamp)

#create a matrix of size n * nbsamp, repeating y in each column
y=horse$y
ymx=as.matrix(y)%*%matrix(1,nrow=1,ncol=nbsamp);

studentisedred=ymx-fittedvalues;

for(l in 1:nbsamp){
  studentisedred[,l]=studentisedred[,l]/sigma[l];
}

for(i in 1:n){
  studentisedred[i,]=studentisedred[i,]/sqrt(1-Hdiag[i]);
}


#posterior mean of studentised residuals
studentisedredm=numeric(n)
for(i in 1:n){
  studentisedredm[i]=mean(studentisedred[i,])
}



#Plot of posterior mean studentised residual versus observation number.
par(mfrow=c(1,1))
plot(seq_along(studentisedredm),studentisedredm,xlab="Index",ylab="Bayesian studentised residual")#,yli
```




```
#Compute posterior mean fitted values
fittedvaluesm=numeric(n)
for(i in 1:n){
fittedvaluesm[i]=mean(fittedvalues[i,])
}

plot(fittedvaluesm,studentisedredm,xlab="Fitted value (posterior mean)",ylab="Bayesian Studentised resi
```



```
#QQ-plot
qqnorm(studentisedredm,xlim=c(-3.5,3.5),ylim=c(-3.5,3.5),lwd=2)
qqline(studentisedredm,col=2,lwd=2)
```



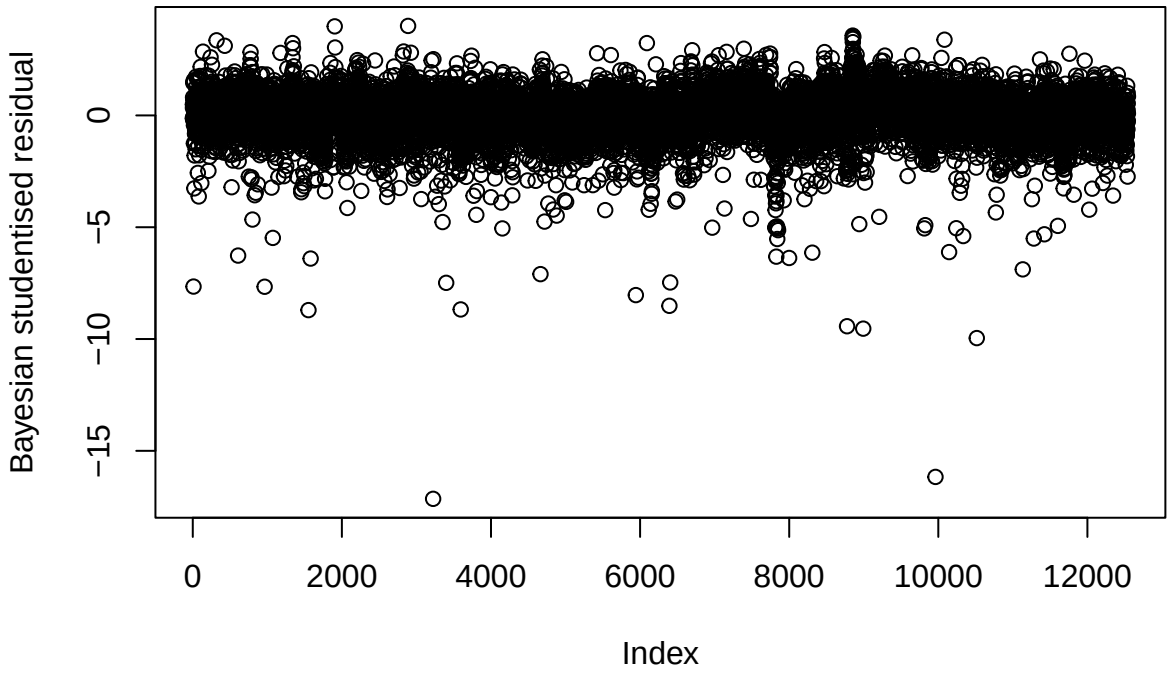
There is a strange behavior near indices 7500-8000 among the studentized residuals (they are significantly more negative than usual). This could warrant some further study to see if some particular factors affected those races. However, outside of this range there is no significant structure here, i.e. the posterior means of the residuals does not have a clear dependence on the index or the fitted value. There are some clear outliers.

On the QQ-plot, the fit is reasonably good for small deviations, but becomes rather poor in the tails. So a robust regression model might be more appropriate here.

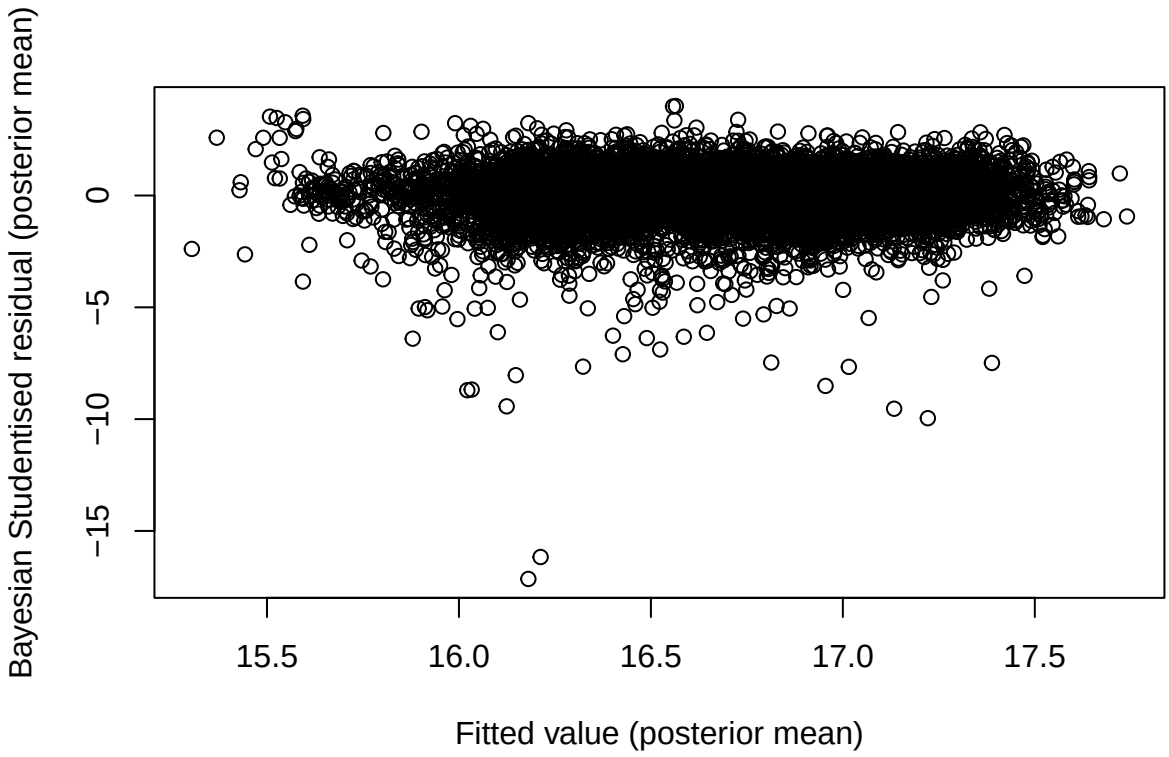
Now we redo these tests for the improved model from part c).

```
nbsamp=1000
samp <- inla.posterior.sample(nbsamp, m.I2)


sigma=1/sqrt(inla.posterior.sample.eval(function(...) {theta},
  samp))

#In this model the link function is the identity, so fitted values are the same as the linear predictor
#(E(y_i|x,theta)=mu_i=eta_i)

fittedvalues=inla.posterior.sample.eval(function(...) {Predictor},
samp)

#studentised residuals

studentisedred=ymx-fittedvalues;
```

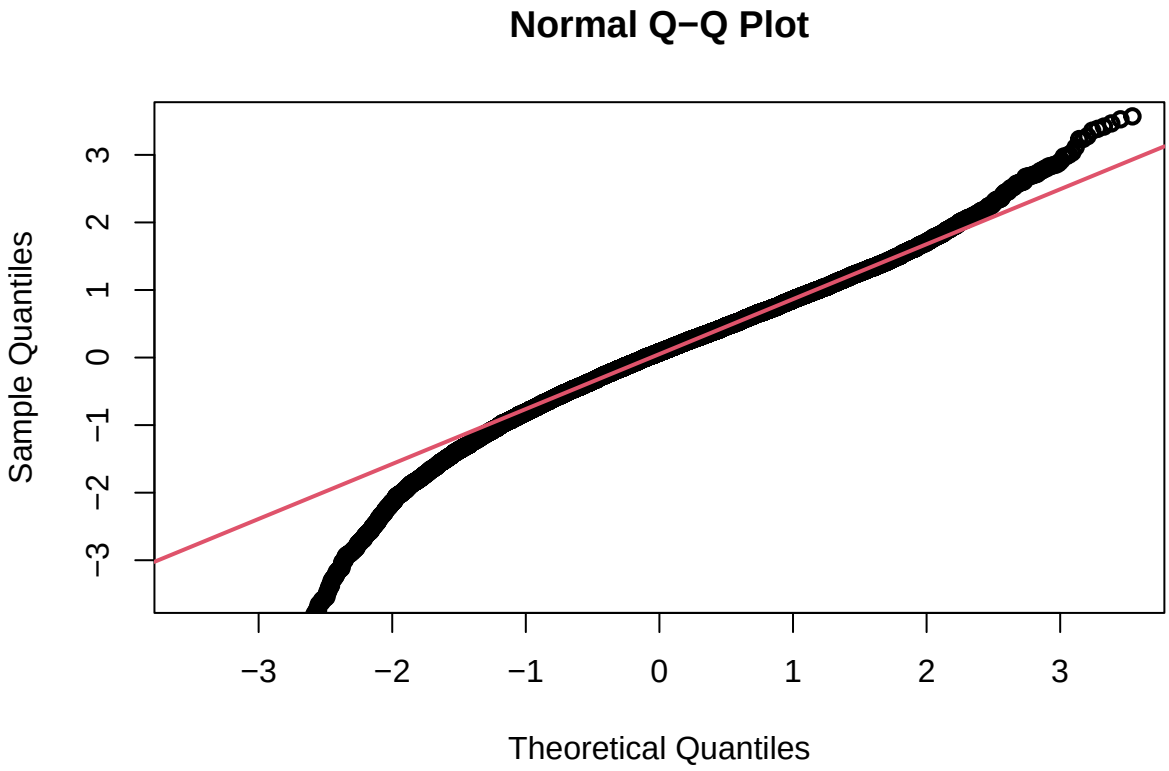
```
for(l in 1:nbsamp){
  studentisedred[,l]=studentisedred[,l]/sigma[l];
}

for(i in 1:n){
  studentisedred[i,]=studentisedred[i,]/sqrt(1-Hdiag[i]);
}


#posterior mean of studentised residuals
studentisedredm=numeric(n)
for(i in 1:n){
  studentisedredm[i]=mean(studentisedred[i,])
}



#Plot of posterior mean studentised residual versus observation number.
par(mfrow=c(1,1))
plot(seq_along(studentisedredm),studentisedredm,xlab="Index",ylab="Bayesian studentised residual")#,yli
```




```
#Compute posterior mean fitted values
fittedvaluesm=numeric(n)
for(i in 1:n){
fittedvaluesm[i]=mean(fittedvalues[i,])
}

plot(fittedvaluesm,studentisedredm,xlab="Fitted value (posterior mean)",ylab="Bayesian Studentised resi
```




```
#QQ-plot
qqnorm(studentisedredm,xlim=c(-3.5,3.5),ylim=c(-3.5,3.5),lwd=2)
qqline(studentisedredm,col=2,lwd=2)
```



As we can see, the results are quite similar to what we have seen for the model in part b).

There is still a strange behavior near indices 7500-8000 among the studentized residuals. However, outside of this range there is no significant structure here, i.e. the posterior means of the residuals does not have a clear dependence on the index or the fitted value. There are some clear outliers.

On the QQ-plot, the fit is reasonably good for small deviations, but becomes rather poor in the tails. So a robust regression model might be more appropriate here.

e)[10 marks] In this question, we are going to use the model you have constructed in part c) to predict a new race, i.e. calculate the posterior probabilities of each participating horse winning that race. First, we load the dataset containing information about the future race.

```
race_to_predict <- read.csv(file = 'race_to_predict.csv')
race_to_predict
```

```
##   race_id       date venue race_no config surface distance going horse_ratings
## 1    1000 1998-09-18    ST       2    B+2       0     1400  GOOD         40-15
##    prize race_class sec_time1 sec_time2 sec_time3 sec_time4 sec_time5 sec_time6
## 1 485000          5        NA        NA        NA        NA        NA        NA
##   sec_time7 time1 time2 time3 time4 time5 time6 time7 place_combination1
## 1        NA    NA    NA    NA    NA    NA    NA    NA                  5
##   place_combination2 place_combination3 place_combination4 place_dividend1
## 1                  7                  8                 NA           27.5
##   place_dividend2 place_dividend3 place_dividend4 win_combination1
## 1              43              57              NA                5
##   win_dividend1 win_combination2 win_dividend2
## 1            86               NA            NA
```

```
runs_to_predict <- read.csv(file = 'runs_to_predict.csv')
runs_to_predict
```

```
##   race_id horse_no horse_id result won lengths_behind horse_age horse_country
## 1    1000        1     3940     NA  NA             NA         3            NZ
## 2    1000        2      474     NA  NA             NA         3            NZ
##   horse_type horse_rating horse_gear declared_weight actual_weight draw
## 1    Gelding           60         --            1148           133    7
## 2    Gelding           60         --            1039           122    4
##   position_sec1 position_sec2 position_sec3 position_sec4 position_sec5
## 1             9             6             6            14            NA
## 2             4             4             4             4            NA
##   position_sec6 behind_sec1 behind_sec2 behind_sec3 behind_sec4 behind_sec5
## 1            NA        2.75        2.75           3       11.00          NA
## 2            NA        1.00        1.75           2        2.25          NA
##   behind_sec6 time1 time2 time3 time4 time5 time6 finish_time win_odds
## 1          NA    NA    NA    NA    NA    NA    NA          NA     55.0
## 2          NA    NA    NA    NA    NA    NA    NA          NA      4.6
##   place_odds trainer_id jockey_id
## 1       17.0         38       138
## 2        1.7         47        31
##  [ reached 'max' / getOption("max.print") -- omitted 12 rows ]
```

Based on your model from part c), compute the posterior probabilities of each of these 14 horses winning the race. [Hint: you will need to sample from the posterior predictive distribution.]

Explanation: We included the information from the new race into the original dataset, except for the mean speed, which was replaced by NA. We applied the appropriate scaling on the covariates. We re-ran the INLA model on this dataset, and obtained samples from the linear predictors for the mean speeds of the horses in the new race using inla.posterior.sample. We added some Gaussian noise to these based on samples from

the sigma2 to obtain samples from the posterior predictive. Based on these samples, we have computed the posterior probabilities of each horse winning the race (i.e. the horse with the highest mean speed wins the race).

```
runs <- read.csv(file = 'runs.csv')
races <- read.csv(file = 'races.csv')

runs2=rbind(runs,runs_to_predict)
races2=rbind(races,race_to_predict)

runs2$distance=races2$distance[runs2$race_id+1]
runs2$speed=runs2$distance/runs2$finish_time
runs2$venue=races2$venue[runs2$race_id+1]
runs2$surface=races2$surface[runs2$race_id+1]
runs2$going=races2$going[runs2$race_id+1]
runs2$race_class=races2$race_class[runs2$race_id+1]
runs2$config=races2$config[runs2$race_id+1]
runs2$prize=races2$prize[runs2$race_id+1]

n=nrow(runs)
npred=nrow(runs_to_predict)
runs2[(n+1):(n+npred),"speed"]=NA

prior.prec <- list(prec=list(prior = "loggamma", param = c(0.1, 0.1)))

prior.beta <- list(mean.intercept = 16.5, prec.intercept = 0.25,
                   mean = 0, prec = 10)

mean.horse_rating=mean(runs$horse_rating[1:n])
sd.horse_rating=sd(runs$horse_rating[1:n])
mean.age=mean(runs$age[1:n])
```

```
## Warning in mean.default(runs$age[1:n]): argument is not numeric or logical:
## returning NA
```

```
sd.age=sd(runs$age[1:n])
mean.log.prize=mean(runs$log.prize[1:n])
```

```
## Warning in mean.default(runs$log.prize[1:n]): argument is not numeric or
## logical: returning NA
```

```
sd.log.prize=sd(runs$log.prize[1:n])


horse2=data.frame(y=runs2$speed, horse_id=as.factor(runs2$horse_id), distance=as.factor(runs2$distance)
                  rating=(runs2$horse_rating-mean.horse_rating)/sd.horse_rating, age=(runs2$horse_age-mea
                  race_class=as.factor(runs2$race_class), config=as.factor(runs2$config),log.prize=(log(

m.I3=inla(y~horse_id+distance+rating+age+surface+going+race_class+config+log.prize,data=horse2,
          family="Gaussian",control.predictor = list(compute = TRUE),
          control.compute = list(cpo=TRUE,dic=TRUE,config = TRUE),
          control.family=list(hyper=prior.prec),control.fixed=prior.beta)
#options(max.print=100)
#summary(m.I3)

nbsamp=2000
```

```
m.I3.samp <- inla.posterior.sample(nbsamp, m.I3,selection = list(Predictor=(n+1):(n+npred)))
sigma.samples=1/sqrt(inla.posterior.sample.eval(function(...) {theta},
m.I3.samp))
predictor.samples=inla.posterior.sample.eval(function(...) {Predictor},
m.I3.samp)

post.pred.samples=matrix(0,nrow=npred,ncol=nbsamp)
for(it in 1:npred){
post.pred.samples[it,]=predictor.samples[it,]+rnorm(nbsamp, mean=0,sd=sigma.samples)
}

winners=apply(post.pred.samples,MARGIN=2, which.max)
probs=rep(0,npred)
for(it in 1:npred){
  probs[it]=mean(winners==it)
}

winning.probs=data.frame(horse_id=runs_to_predict$horse_id,winning.probability=probs)
winning.probs
```

```
##    horse_id winning.probability
## 1      3940              0.0020
## 2       474              0.0815
## 3      3647              0.0695
## 4       144              0.0220
## 5      3712              0.0855
## 6      3734              0.1815
## 7      1988              0.0915
## 8      3247              0.0110
## 9      4320              0.0950
## 10     1077              0.0525
## 11     3916              0.1605
## 12      768              0.0805
## 13     3164              0.0580
## 14      498              0.0090
```

As we can see, according to the posterior distribution, horse with id 3734 is most likely to win the race (with probability 0.165).