University of Edinburgh

School of Mathematics

Bayesian Data Analysis, 2022/2023, Semester 2

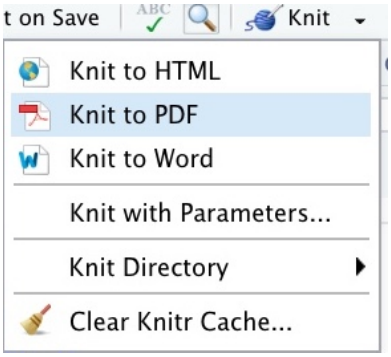
Assignment 1 Solutions

IMPORTANT INFORMATION ABOUT THE ASSIGNMENT

In this paragraph, we summarize the essential information about this assignment. The format and rules for this assignment are different from your other courses, so please pay attention.

1) Deadline: The deadline for submitting your solutions to this assignment is the 6 March 12:00 noon Edinburgh time.

2) Format: You will need to submit your work as 2 components: a PDF report, and your R Markdown (.Rmd) notebook. There will be two separate submission systems on Learn: Gradescope for the report in PDF format, and a Learn assignment for the code in Rmd format. You need to write your solutions into this R Markdown notebook (code in R chunks and explanations in Markdown chunks), and then select Knit/Knit to PDF in RStudio to create a PDF report.



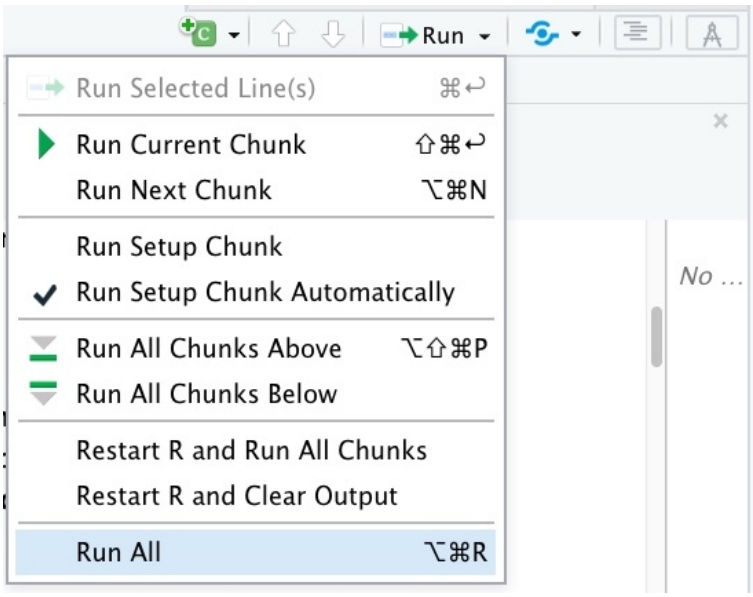
The compiled PDF needs to contain everything in this notebook, with your code sections clearly visible (not hidden), and the output of your code included. Reports without the code displayed in the PDF, or without the output of your code included in the PDF will be marked as 0, with the only feedback "Report did not meet submission requirements".

You need to upload this PDF in Gradescope submission system, and your Rmd file in the Learn assignment submission system. You will be required to tag every sub question on Gradescope.

Some key points that are different from other courses:

a) Your report needs to contain written explanation for each question that you solve, and some numbers or plots showing your results. Solutions without written explanation that clearly demonstrates that you understand what you are doing will be marked as 0 irrespectively whether the numerics are correct or not.

b) Your code has to be possible to run for all questions by the Run All in RStudio, and reproduce all of the numerics and plots in your report (up to some small randomness due to stochasticity of Monte Carlo simulations). The parts of the report that contain material that is not reproduced by the code will not be marked (i.e. the score will be 0), and the only feedback in this case will be that the results are not reproducible from the code.



c) Multiple Submissions are allowed BEFORE THE DEADLINE are allowed for both the report, and the code.
However, multiple submissions are NOT ALLOWED AFTER THE DEADLINE.
YOU WILL NOT BE ABLE TO MAKE ANY CHANGES TO YOUR SUBMISSION AFTER THE DEADLINE.
Nevertheless, if you did not submit anything before the deadline, then you can still submit your work after the deadline, but late penalties will apply. The timing of the late penalties will be determined by the time you have submitted BOTH the report, and the code (i.e. whichever was submitted later counts).

We illustrate these rules by some examples:

Alice has spent a lot of time and effort on her assignment for BDA. Unfortunately she has accidentally introduced a typo in her code in the first question, and it did not run using Run All in RStudio. - Alice will get 0 for the whole assignment, with the only feedback "Results are not reproducible from the code".

Bob has spent a lot of time and effort on his assignment for BDA. Unfortunately he forgot to submit his code. - Bob will get no personal reminder to submit his code. Bob will get 0 for the whole assignment, with the only feedback "Results are not reproducible from the code, as the code was not submitted."

Charles has spent a lot of time and effort on his assignment for BDA. He has submitted both his code and report in the correct formats. However, he did not include any explanations in the report. Charles will get 0 for the whole assignment, with the only feedback "Explanation is missing."

Denise has spent a lot of time and effort on her assignment for BDA. She has submitted her report in the correct format, but thought that she can include her code as a link in the report, and upload it online (such as Github, or Dropbox). - Denise will get 0 for the whole assignment, with the only feedback "Code was not uploaded on Learn."

3) Group work: This is an INDIVIDUAL ASSIGNMENT, like a 2 week exam for the course. Communication between students about the assignment questions is not permitted. Students who submit work that has not been done individually will be reported for Academic Misconduct, that can lead to serious consequences. Each problem will be marked by a single instructor, so we will be able to spot students who copy.

4) Piazza: During the periods of the assignments, the instructor will change Piazza to allow messaging the instructors only, i.e. students will not see each others messages and replies.

Only questions regarding clarification of the statement of the problems will be answered by the instructors. The instructors will not give you any information related to the solution of the problems, such questions will be simply answered as "This is not about the statement of the problem so we cannot answer your question."

THE INSTRUCTORS ARE NOT GOING TO DEBUG YOUR CODE, AND YOU ARE ASSESSED ON YOUR ABILITY TO RESOLVE ANY CODING OR TECHNICAL DIFFICULTIES THAT YOU ENCOUNTER ON YOUR OWN.

5) Office hours: There will be two office hours per week (Monday 14:00-15:00, and Wednesdays 15:00-16:00) during the 2 weeks for this assignment. The links are available on Learn / Course Information. I will be happy to discuss the course/workshop materials. However, I will only answer questions about the assignment that require clarifying the statement of the problems, and will not give you any information about the solutions. Students who ask for feedback on their assignment solutions during office hours will be removed from the meeting.

6) Late submissions and extensions: NO EXTENSIONS ARE ALLOWED FOR THIS ASSIGNMENT, AND THERE IS NO SUCH OPTION PROVIDED IN THE ESC SYSTEM. Students who have existing Learning Adjustments in Euclid will be allowed to have the same adjustments applied to this course as well, but they need to apply for this BEFORE THE DEADLINE on the website

https://www.ed.ac.uk/student-administration/extensions-special-circumstances

by clicking on "Access your learning adjustment". This will be approved automatically.

Students who submit their work late will have late submission penalties applied by the ESC team automatically (this means that even if you are 1 second late because of your internet connection was slow, the penalties will still apply). The penalties are 5% of the total mark deduced for every day of delay started (i.e. one minute of delay counts for 1 day). The course instructors do not have any role in setting these penalties, we will not be able to change them.

```
rm(list = ls(all = TRUE))
#Do not delete this!
#It clears all variables to ensure reproducibility
```

Problem 1

In this problem, we study a dataset about currency exchange rates. The exrates dataset of the stochvol package contains the daily average exchange rates of 24 currencies versus the EUR, from 2000-01-03 until 2012-04-04.

```
require(stochvol)
```

```
## Loading required package: stochvol
```

```
data("exrates")
#You may need to set the working directory first before loading the dataset
#setwd("location of Assignment 1")
#The first 6 rows of the dataframe
print.data.frame(exrates[1:6,])
```

```
##                AUD    CAD    CHF    CZK    DKK    GBP    HKD     IDR    JPY
## 2000/01/03 1.5346 1.4577 1.6043 36.063 7.4404 0.6246 7.8624 7133.32 102.75
## 2000/01/04 1.5677 1.4936 1.6053 36.270 7.4429 0.6296 8.0201 7265.16 105.88
## 2000/01/05 1.5773 1.5065 1.6060 36.337 7.4444 0.6324 8.0629 7437.97 107.34
## 2000/01/06 1.5828 1.5091 1.6068 36.243 7.4441 0.6302 8.0843 7495.52 108.72
## 2000/01/07 1.5738 1.5010 1.6079 36.027 7.4436 0.6262 8.0030 7398.88 108.09
## 2000/01/10 1.5587 1.4873 1.6089 35.988 7.4448 0.6249 7.9471 7320.49 107.26
##                 KRW    MXN    MYR    NOK    NZD    PHP    PLN    RON     RUB
## 2000/01/03 1140.02 9.6105 3.8422 8.0620 1.9331 40.424 4.1835 1.8273 27.7548
## 2000/01/04 1157.32 9.7453 3.9188 8.1500 1.9745 40.992 4.2423 1.8858 28.3594
## 2000/01/05 1176.08 9.8969 3.9393 8.2060 1.9956 41.637 4.2627 1.8979 28.3006
## 2000/01/06 1191.90 9.9751 3.9498 8.2030 2.0064 42.148 4.2593 1.9000 28.5111
## 2000/01/07 1169.52 9.8368 3.9100 8.1945 1.9942 41.493 4.1897 1.8822 28.2526
```

```
## 2000/01/10 1157.84 9.6449 3.8824 8.1900 1.9783 41.226 4.1567 1.8729 28.7609
##                SEK    SGD     THB      TRY    USD       date
## 2000/01/03 8.5520 1.6769 37.2793 0.546131 1.0090 2000-01-03
## 2000/01/04 8.6215 1.7047 38.2078 0.552354 1.0305 2000-01-04
## 2000/01/05 8.6415 1.7159 38.5375 0.555329 1.0368 2000-01-05
## 2000/01/06 8.6445 1.7291 39.0734 0.555674 1.0388 2000-01-06
## 2000/01/07 8.6450 1.7096 38.4299 0.554980 1.0284 2000-01-07
## 2000/01/10 8.6570 1.6975 38.0328 0.552469 1.0229 2000-01-10
```

```
cat(paste("Data from ", min(exrates$date)," until ",max(exrates$date)))
```

```
## Data from  2000-01-03  until  2012-04-04
```

As we can see, not all dates are included in the dataset. Some are missing, such as weekends, and public holidays.

In this problem, we are going to fit a various stochastic volatility models on this dataset (see e.g. https://www.jstor.org/stable/1392251).

a)[10 marks] Consider the following leveraged Stochastic Volatility (SV) model.

$$y_t = \beta_0 + \beta_1 y_{t-1} + \exp(h_t/2)\epsilon_t \quad \text{for} \quad 1 \le t \le T,$$
$$h_{t+1} = \mu + \phi(h_t - \mu) + \sigma\eta_t \quad \text{for} \quad 0 \le t \le T, \quad h_0 \sim N(\mu, \sigma^2/(1-\phi^2)),$$
$$(\epsilon_t, \eta_t) \sim N(0, \Sigma_\rho) \quad \text{for} \quad \Sigma_\rho = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}.$$

Here t is the time index, y_t are the observations (such as daily USD/EUR rate), h_t are the log-variance process, ϵ_t is the observation noise, and η_t is the log-variance process noise (which are correlated, but independent for different values of t). The hyperparameters are $\beta_0, \beta_1, \mu, \phi, \sigma, \rho$.

For stability, it is necessary to have $\phi \in (-1, 1)$, and by the definition of correlation matrices, we have $\rho \in [-1, 1]$.

Implement this model in JAGS or Stan on the first 3 months of USD/EUR data from the dataset, i.e. from dates 2000-01-03 until 2000-04-02.

Explain how did you choose priors for all parameters. Explain how did you take into account the days without observation in your model.

Fit the model, do convergence diagnostics, print out the summary of the results, and discuss them.

Make sure that the Effective Sample Size is at least 1000 for all 6 hyperparameters (you need to choose burn-in and number of steps appropriately for this).

Explanation:

First, we implement the model in JAGS. Here the unobserved components of y will be included as NA in the data, and JAGS will automatically create stochastic nodes for them.

We start by explaining our prior choices. We use Gamma$(10^{-4}, 10^{-4})$ prior for $\tau = 1/\sigma^2$, as this has mean 1 and variance 10^4, making it quite uninformative. We use uniform(-1,1) priors for parameters ρ and ϕ, as they are constrained in the interval $(-1, 1)$ according to the question statement. We also use uniform $(-1, 1)$ prior for β_1, since values larger than 1 would be unstable (i.e. y_t would grow exponentially in t), and we do not believe that this is expected of the major currencies we are modelling here. For β_0, we expect this parameter to be on the order of magnitude 1 for USD/EUR, so we set a $N(0, 1)$ prior. We also need to put a prior distribution on y_0, since this is part of the model for the first observation y_1. Based on the last few days of USD/EUR exchange rate in December 1999 available on https://www.federalreserve.gov/releases/h10/20000103, the rate is around 0.99 with low variability, so we select a prior $y_0 \sim N(0.99, 0.01^2)$.

In JAGS, multivariate nodes cannot be partially observed. Hence we do not use multivariate normal distributions in our implementation. Instead, we need to compute the conditional distribution of ϵ_t given η_t. Since these random variables are jointly normal with covariance matrix $\Sigma_\rho = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}$, based on the formula on https://statproofbook.github.io/P/mvn-cond, the conditional distribution of $\epsilon_t|\eta_t \sim N(\rho\eta_t, 1-\rho^2)$. This is used in our definition of ϵ_t in the model code.

```
library(rjags)
```

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

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

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

```
model_string_1a <- "
var h[n_days+1], y[n_days+1], eta[n_days+1],
  sigma, sigma2, tau, phi, rho, mu, beta0, beta1;

model{
  tau~dgamma(1e-4,1e-4);
  sigma<-1/sqrt(tau);
  phi~dunif(-1,1);
  rho~dunif(-1,1);
  mu~dnorm(-8,1);
  beta0~dnorm(0,1);
  beta1~dunif(-1,1);
  tau_h0<- (1-phi^2)*tau;
  h[1]~dnorm(mu, tau_h0);

  for(t in 1:(n_days+1))
  {
    eta[t]~dnorm(0,1);
  }
  for(t in 1:n_days)
  {
    h[t+1]=mu+phi*(h[t]-mu)+sigma*eta[t];
  }

  y[1]~dnorm(0.99,10000);
  for(t in 1:n_days)
  {
     y[t+1]~dnorm(beta0+beta1*y[t]+exp(h[t+1]/2)*rho*eta[t+1],1/((1-rho^2)*exp(h[t+1])));
  }
}"


n_observed=max(which(exrates$date<=as.Date("2000-04-02")));
n_days=as.numeric(as.Date("2000-04-02")-as.Date("2000-01-03"))+1
y=rep(NA,n_days+1);

#Data preparation - y is filled up with the observed exchange rates,
#and set to NA on dates without observations
for (it in 1:n_observed)
{
  date_it=exrates$date[it]-as.Date("2000-01-03")+1;
```

```
  y[date_it]=exrates$USD[it];
}


#data
data=list(y=y,n_days=n_days);


tau.init=10000;
phi.init=0;
rho.init=0;
beta0.init=0;
beta1.init=0.9;
mu.init=-8;


inits=list();
n.chains=8;
for(it in 1:n.chains)
{
  inits<-c(inits, list(list(tau=tau.init+runif(1,-1000,1000),
  phi=phi.init+runif(1,min=-0.2,max=0.2),
  rho=rho.init+runif(1,min=-0.2,max=0.2),mu=mu.init+rnorm(1),
  beta0=beta0.init+rnorm(1), beta1=beta1.init+runif(1,-0.1,0.1))));
}
#list of initial values
#(if not supplied, the function jags.model will generate initial values)

#passing the model to jags *format*
model=jags.model(textConnection(model_string_1a),n.chains=n.chains,data=data,
                 inits=inits)
```

```
## Compiling model graph
##    Declaring variables
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 65
##    Unobserved stochastic nodes: 126
##    Total graph size: 1297
##
## Initializing model
```

```
#Burnin
update(model,40000,progress.bar="none")

res.model=coda.samples(model,variable.names=
c("sigma","beta0","beta1","mu", "phi", "rho"), n.iter=60000,n.thin=10,progress.bar="none")
#Display some summary statistics
summary(res.model)
```

```
##
## Iterations = 41001:101000
## Thinning interval = 1
## Number of chains = 8
```

```
## Sample size per chain = 60000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean      SD  Naive SE Time-series SE
## beta0  0.05508 0.02941 4.245e-05       0.003071
## beta1  0.94350 0.02982 4.304e-05       0.003130
## mu    -9.84182 0.29268 4.224e-04       0.003875
## phi    0.12263 0.57670 8.324e-04       0.003440
## rho    0.10006 0.54457 7.860e-04       0.012361
## sigma  0.18271 0.18774 2.710e-04       0.001920
##
## 2. Quantiles for each variable:
##
##             2.5%       25%      50%      75%   97.5%
## beta0   0.004459   0.03240  0.05244  0.07683  0.1130
## beta1   0.884769   0.92144  0.94615  0.96654  0.9948
## mu    -10.271721 -10.00367 -9.86579 -9.72219 -9.2990
## phi    -0.906581  -0.36876  0.17227  0.64455  0.9678
## rho    -0.915391  -0.33621  0.14030  0.56175  0.9492
## sigma   0.007271   0.03702  0.11320  0.27735  0.6688
```

We can see that there is a relatively weak effect of leverage since the posterior mean of ρ is quite small. The posterior mean of the coefficient β_1 is close to 1, this indicates that there is a relatively strong correlation in exchange rates y_i between consecutive days, which is expected as these are major currencies.

```
effectiveSize(res.model)
```

```
##      beta0      beta1         mu        phi        rho      sigma
##   103.1913   102.5481  8395.2436 29917.4205  2302.1371 10397.9976
```

```
gelman.diag(res.model)
```

```
## Potential scale reduction factors:
##
##       Point est. Upper C.I.
## beta0       1.09       1.20
## beta1       1.09       1.20
## mu          1.00       1.00
## phi         1.00       1.00
## rho         1.01       1.01
## sigma       1.00       1.01
##
## Multivariate psrf
##
## 1.09
```

The effective sample sizes are quite low, and the Gelman-Rubin diagnostic values are high, indicating poor mixing.

Now we also implement the model in Stan. As explained on https://mc-stan.org/docs/stan-users-guide/stochastic-volatility-models.html, rescaling the parameter $h[t]$ by a factor of $\sigma/\sqrt{1-\phi^2}$ can improve mixing. For this reason, we use the parameters $h_{std}[t] = h[t]/\sigma/\sqrt{1-\phi^2}$ in the parameters section in Stan.

Taking into account the non-observed components in y is more difficult in Stan than in JAGS. We do this by storing the observed components in the data variable y_obs, and the non-observed components in the

parameter variable y_missing. These are then combined into a single vector y in the transformed parameters section. This vector y is used to define the model in the model section, in combination with the same approach as in JAGS, via the conditional distribution of ϵ_t given η_t.

```
library(rstan)
```

```
## Loading required package: StanHeaders
```

```
##
## rstan version 2.26.16 (Stan version 2.26.1)
```

```
## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
## For within-chain threading using `reduce_sum()` or `map_rect()` Stan functions,
## change `threads_per_chain` option:
## rstan_options(threads_per_chain = 1)
```

```
##
## Attaching package: 'rstan'
```

```
## The following object is masked from 'package:coda':
##
##     traceplot
```

```
options(mc.cores = parallel::detectCores())

#model in STAN language
model_string_1a <-
"data{
int<lower=0> n_days;
int<lower=0> n_observed;
int<lower=0> n_pred;
vector[n_observed] y_obs;
array[n_days] int obs_or_not;
}

parameters
{
  vector[n_days+n_pred-n_observed+1] y_missing; //y_missing[1] corresponds to y_0
  vector[n_days+n_pred+2] h_std;
  real<lower=-1,upper=1> rho;
  real<lower=-1,upper=1> phi;
  real<lower=0> sigma2;
  real beta0;
  real <lower=-1,upper=1> beta1;
  real mu;
}

transformed parameters
{
  real sigma=sqrt(sigma2);
  real sqrt1mphi2=sqrt(1-phi^2);

  vector[n_days+n_pred+1] h;
  h=mu+sigma/sqrt1mphi2*h_std;
```

```
  vector[n_days+n_pred+1] eta;

  eta[2:(n_days+n_pred+1)]=(h_std[3:(n_days+n_pred+2)]-phi*h_std[2:(n_days+n_pred+1)])/sqrt1mphi2;

  vector[n_days+n_pred+1] y;
  //y[1] corresponds to y_0 in the formula
  y[1]=y_missing[1];

  {

    int ind_obs=1;
    int ind_missing=2;

    for(i in 1:n_days)
    {
      if(obs_or_not[i]==1)
      {
        y[i+1]=y_obs[ind_obs];
        ind_obs=ind_obs+1;
      }
      else
      {
        y[i+1]=y_missing[ind_missing];
        ind_missing=ind_missing+1;
      }
    }
    for(i in (n_days+1):(n_days+n_pred))
    {
        y[i+1]=y_missing[ind_missing];
        ind_missing=ind_missing+1;
    }

  }
}



model{
  sigma2~inv_gamma(1e-4,1e-4);
  phi~uniform(-1,1);
  rho~uniform(-1,1);
  mu~normal(-8,1);
  beta0~normal(0,1);
  beta1~uniform(-1,1);

  h_std[1]~normal(0, 1);
  y[1]~normal(0.99,0.01);

  for(t in 1:(n_days+n_pred+1))
  {
     h_std[t+1]~normal(phi*h_std[t],sqrt1mphi2);
  }
```

```
  for(t in 1:(n_days+n_pred))
  {
     y[t+1]~normal(beta0+beta1*y[t]+exp(h[t+1]/2)*rho*eta[t+1],sqrt1mphi2*exp(h[t+1]/2));
  }
}

generated quantities
{
  vector[n_observed] y_obs_rep;

  {
    int ind_obs=1;

    for(t in 1:n_days)
    {
      if(obs_or_not[t]==1)
      {
      y_obs_rep[ind_obs]=normal_rng(beta0+beta1*y[t]+exp(h[t+1]/2)*rho*eta[t+1],sqrt1mphi2*exp(h[t+1]/2)
      ind_obs=ind_obs+1;
      }
    }

  }
}
"

#data

#finding dates up to 2000-04-02
n_observed=max(which(exrates$date<=as.Date("2000-04-02")))
n_days=as.numeric(as.Date("2000-04-02")-as.Date("2000-01-03"))+1
n_pred=3
y_obs=exrates$USD[1:n_observed]
obs_or_not=rep(0,n_days)
for (it in 1:n_observed)
{
  date_it=exrates$date[it]-as.Date("2000-01-03")+1
  obs_or_not[date_it]=1
}
#int<lower=0> n_days;
#int<lower=0> n_observed;
#vector[n_observed] y_obs;
#array[n_days] int obs_or_not;

data=list(n_days=n_days,n_observed=n_observed, y_obs=y_obs, obs_or_not=obs_or_not,n_pred=n_pred)

fname="model_1a.stan"
cat(model_string_1a,file=fname,append=FALSE)
# list with data and hyperparameters

#passing the model string to STAN
res1a<- stan(file = fname, data = data,
           # Below are optional arguments
```

```
            iter = 200000, warmup=40000,
            #iter is the number of iterations, including the burn-in
            #the burn-in period is set to iter/2 by default, it can be set to
            #something else using the warmup parameter
            chains = 8,cores = parallel::detectCores(),thin=10,refresh=0)
```

```
## Warning in validityMethod(object): The following variables have undefined
## values: eta[1]. Many subsequent functions will not work correctly.
```

```
## Warning: There were 7434 divergent transitions after warmup. See
## https://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.
```

```
## Warning: There were 3629 transitions after warmup that exceeded the maximum treedepth. Increase max_
## https://mc-stan.org/misc/warnings.html#maximum-treedepth-exceeded
```

```
## Warning: There were 8 chains where the estimated Bayesian Fraction of Missing Information was low. S
## https://mc-stan.org/misc/warnings.html#bfmi-low
```

```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

```
print(res1a,pars=c("mu", "rho", "phi", "beta0", "beta1", "sigma"))
```

```
## Inference for Stan model: anon_model.
## 8 chains, each with iter=2e+05; warmup=40000; thin=10;
## post-warmup draws per chain=16000, total post-warmup draws=128000.
##
##         mean se_mean   sd   2.5%   25%   50%   75% 97.5% n_eff Rhat
## mu     -9.31    0.01 0.72 -10.39 -9.84 -9.43 -8.87 -7.65 15532    1
## rho     0.13    0.00 0.40  -0.70 -0.12  0.15  0.41  0.87 11491    1
## phi     0.21    0.01 0.67  -0.95 -0.49  0.49  0.82  0.96  3406    1
## beta0   0.05    0.00 0.03   0.00  0.03  0.05  0.07  0.11 14057    1
## beta1   0.95    0.00 0.03   0.89  0.93  0.95  0.97  1.00 14025    1
## sigma   0.20    0.00 0.19   0.01  0.04  0.14  0.30  0.68 16064    1
##
## Samples were drawn using NUTS(diag_e) at Sun Mar 26 18:08:28 2023.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

The effective sample sizes are much better this time compared to JAGS. The summary statistics are similar to what we got from JAGS, with the posterior mean of β_1 close to 1 as before, indicating strong correlation between exchange rates at consecutive days.

b)[10 marks] In practice, one often encounters outliers in exchange rates. These can be sometimes modeled by assuming Student's t distribution in the observation errors (i.e. ϵ_t). The robust leveraged SV model can be expressed as

$$
\begin{aligned}
y_t &= \beta_0 + \beta_1 y_{t-1} + \exp(h_t/2)\epsilon_t \quad \text{for} \quad 1 \le t \le T, \\
h_{t+1} &= \mu + \phi(h_t - \mu) + \sigma\eta_t \quad \text{for} \quad 0 \le t \le T, \quad h_0 \sim N(\mu, \sigma^2/(1-\phi^2)), \\
\eta_t &\sim N(0,1) \\
\epsilon_t|\eta_t &\sim t_\nu(\rho\eta_t, 1).
\end{aligned}
$$

Here ν is the degrees of freedom parameter (unknown).

Implement this model in JAGS or Stan on the first 3 months of USD/EUR data from the dataset.

Explain how did you choose priors for all parameters. Explain how did you take into account the days without observation in your model.

Fit the model, do convergence diagnostics, print out the summary of the results, and discuss them.

Make sure that the Effective Sample Size is at least 1000 for all 6 hyperparameters (you need to choose burn-in and number of steps appropriately for this).

We implement this model in Stan. The only difference to the model in a) is that the conditional distribution $\epsilon_t|\eta_t \sim t_\nu(\rho\eta_t, 1)$, not $N(\rho\eta_t, 1-\rho^2)$ as previously. We use a Gamma(2,0.1) prior for the degrees of freedom parameter ν. As explained in https://www.tandfonline.com/doi/epdf/10.1198/jbes.2009.07145, this is a non-informative prior that has mean 20 and variance 200, covering a large range of possible values of ν. The un-observed days are taken into account in the same way as in part a).

```
library(rstan)
options(mc.cores = parallel::detectCores())

#model in STAN language
model_string_1b <-
"data{
int<lower=0> n_days;
int<lower=0> n_observed;
int<lower=0> n_pred;
vector[n_observed] y_obs;
array[n_days] int obs_or_not;
}

parameters
{
  vector[n_days+n_pred-n_observed+1] y_missing; //y_missing[1] corresponds to y_0
  vector[n_days+n_pred+2] h_std;
  real<lower=-1,upper=1> rho;
  real<lower=-1,upper=1> phi;
  real<lower=0> sigma2;
  real beta0;
  real <lower=-1,upper=1> beta1;
  real mu;
  real <lower=0> nu; #degrees of freedom
}

transformed parameters
{
  real sigma=sqrt(sigma2);
  real sqrt1mphi2=sqrt(1-phi^2);

  vector[n_days+n_pred+1] h;

  h=mu+sigma/sqrt1mphi2*h_std;

  vector[n_days+n_pred+1] eta;
  for(t in 1:(n_days+n_pred))
  {
     eta[t+1]=(h_std[t+2]-phi*h_std[t+1])/sqrt1mphi2;
  }
```

```
  vector[n_days+n_pred+1] y;
  //y[1] corresponds to y_0 in the formula
  y[1]=y_missing[1];

  {

    int ind_obs=1;
    int ind_missing=2;

    for(i in 1:n_days)
    {
      if(obs_or_not[i]==1)
      {
        y[i+1]=y_obs[ind_obs];
        ind_obs=ind_obs+1;
      }
      else
      {
        y[i+1]=y_missing[ind_missing];
        ind_missing=ind_missing+1;
      }
    }
    for(i in (n_days+1):(n_days+n_pred))
    {
        y[i+1]=y_missing[ind_missing];
        ind_missing=ind_missing+1;
    }

  }
}



model{
  sigma2~inv_gamma(1e-4,1e-4);
  phi~uniform(-1,1);
  rho~uniform(-1,1);
  mu~normal(-8,1);
  beta0~normal(0,1);
  beta1~uniform(-1,1);
  nu~gamma(2,0.1);

  h_std[1]~normal(0, 1);
  y[1]~normal(0.99,0.01);

  for(t in 1:(n_days+n_pred+1))
  {
     h_std[t+1]~normal(phi*h_std[t],sqrt1mphi2);
  }

  for(t in 1:(n_days+n_pred))
  {
     y[t+1]~student_t(nu,beta0+beta1*y[t]+exp(h[t+1]/2)*rho*eta[t+1],exp(h[t+1]/2));
```

```
  }

}

generated quantities
{
  vector[n_observed] y_obs_rep;

  {
    int ind_obs=1;

    for(t in 1:n_days)
    {
      if(obs_or_not[t]==1)
      {
      y_obs_rep[ind_obs]=student_t_rng(nu,beta0+beta1*y[t]+
      exp(h[t+1]/2)*rho*eta[t+1],exp(h[t+1]/2));
      ind_obs=ind_obs+1;
      }
    }

  }
}
"


#data

#finding dates up to 2000-04-02
n_observed=max(which(exrates$date<=as.Date("2000-04-02")));
n_days=as.numeric(as.Date("2000-04-02")-as.Date("2000-01-03"))+1
n_pred=3
y_obs=exrates$USD[1:n_observed];
obs_or_not=rep(0,n_days);
for (it in 1:n_observed)
{
  date_it=exrates$date[it]-as.Date("2000-01-03")+1;
  obs_or_not[date_it]=1;
}


data=list(n_days=n_days,n_pred=n_pred,n_observed=n_observed, y_obs=y_obs, obs_or_not=obs_or_not);

fname="model_1b.stan";
cat(model_string_1b,file=fname,append=FALSE);
# list with data and hyperparameters

#passing the model string to STAN
res1b<- stan(file = fname, data = data,
            # Below are optional arguments
            iter = 200000, warmup = 40000,
            #iter is the number of iterations, including the burn-in
            #the burn-in period is set to iter/2 by default, it can be set to
```

```
            #something else using the warmup parameter
            chains = 8,cores = parallel::detectCores(),thin=10,refresh=0);
```

```
## Warning in validityMethod(object): The following variables have undefined
## values: eta[1]. Many subsequent functions will not work correctly.
```

```
## Warning: There were 1217 divergent transitions after warmup. See
## https://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.
```

```
## Warning: There were 2617 transitions after warmup that exceeded the maximum treedepth. Increase max_
## https://mc-stan.org/misc/warnings.html#maximum-treedepth-exceeded
```

```
## Warning: There were 8 chains where the estimated Bayesian Fraction of Missing Information was low. S
## https://mc-stan.org/misc/warnings.html#bfmi-low
```

```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

```
print(res1b,pars=c("mu", "rho", "phi", "beta0", "beta1", "sigma", "nu"))
```

```
## Inference for Stan model: anon_model.
## 8 chains, each with iter=2e+05; warmup=40000; thin=10;
## post-warmup draws per chain=16000, total post-warmup draws=128000.
##
##         mean se_mean    sd   2.5%    25%    50%   75% 97.5%  n_eff Rhat
## mu    -10.10    0.00  0.39 -10.72 -10.32 -10.13 -9.94 -9.30  13361    1
## rho     0.11    0.00  0.45  -0.81  -0.21   0.13  0.45  0.91  58317    1
## phi     0.14    0.01  0.57  -0.90  -0.35   0.19  0.66  0.98   7360    1
## beta0   0.05    0.00  0.03   0.00   0.03   0.05  0.07  0.11   7929    1
## beta1   0.95    0.00  0.03   0.89   0.93   0.95  0.97  1.00   7942    1
## sigma   0.18    0.00  0.18   0.01   0.04   0.11  0.26  0.66  58153    1
## nu     22.47    0.04 13.89   5.39  12.32  19.26 29.14 57.71 107535    1
##
## Samples were drawn using NUTS(diag_e) at Sun Mar 26 19:52:17 2023.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

The posterior means of the parameter values are similar to what we had in part a). The Gelman-Rubin convergence diagnostics shows that we have reached convergence, and the ESS is above 1000 for all parameters. The posterior mean of ν is around 22, indicating that the noise distribution is not too far from a Gaussian.

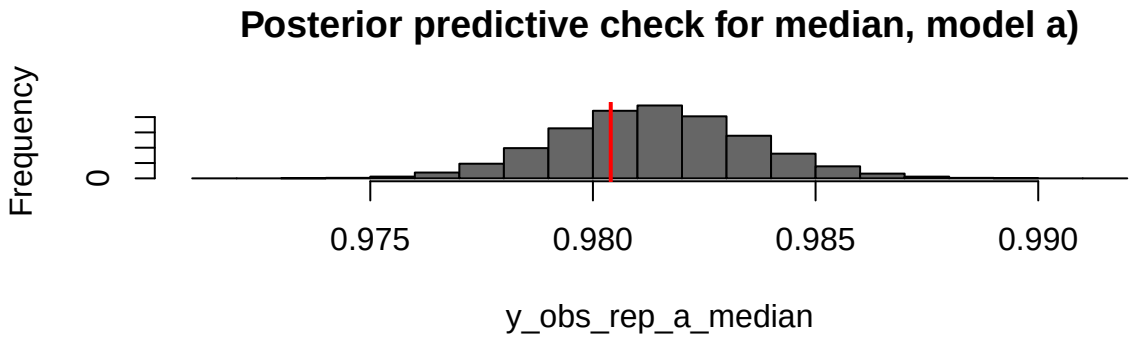
c)[10 marks]

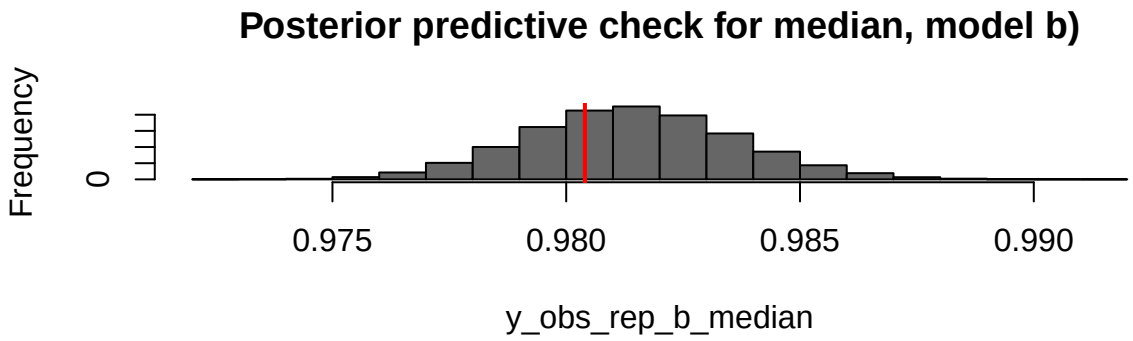
Perform posterior predictive checks on both models a) and b). Explain how did you choose the test functions.

We included the posterior replicates in the Stan model code in the generated quantities block. We try 3 test functions: the median, skewness, and the mean square difference, which we define as the mean of ((y_obs[i]-y_obs[i+1])^2) from $i = 1$ to n_obs-1 (n_obs is the number of days where we observed the exchange rate). This mean square function is related to the volatility of the exchange rate, so it is a relevant quantity to test our model on.

```
y_obs_rep_a=extract(res1a)$y_obs_rep;
y_obs_rep_b=extract(res1b)$y_obs_rep;



y_obs_rep_a_median=apply(y_obs_rep_a,1,median)
y_obs_rep_b_median=apply(y_obs_rep_b,1,median)
```

```
par(mfrow=c(2,1))
hist(y_obs_rep_a_median,col="gray40",
     main="Posterior predictive check for median, model a)", breaks=15)
abline(v=median(y_obs),col="red",lwd=2)

hist(y_obs_rep_b_median,col="gray40",
     main="Posterior predictive check for median, model b)", breaks=15)
abline(v=median(y_obs),col="red",lwd=2)
```

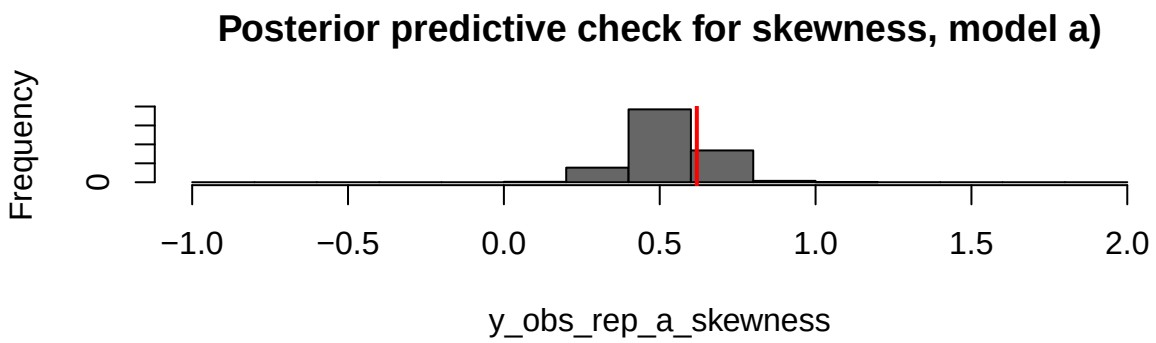


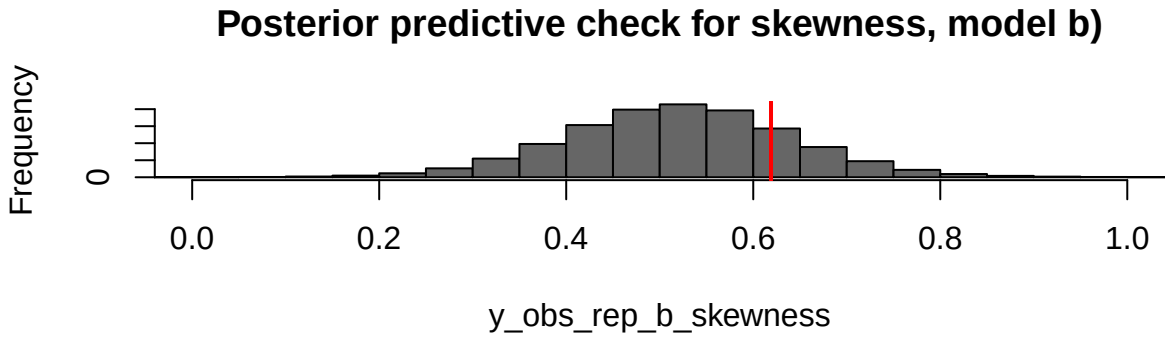



```
require(fBasics)
```

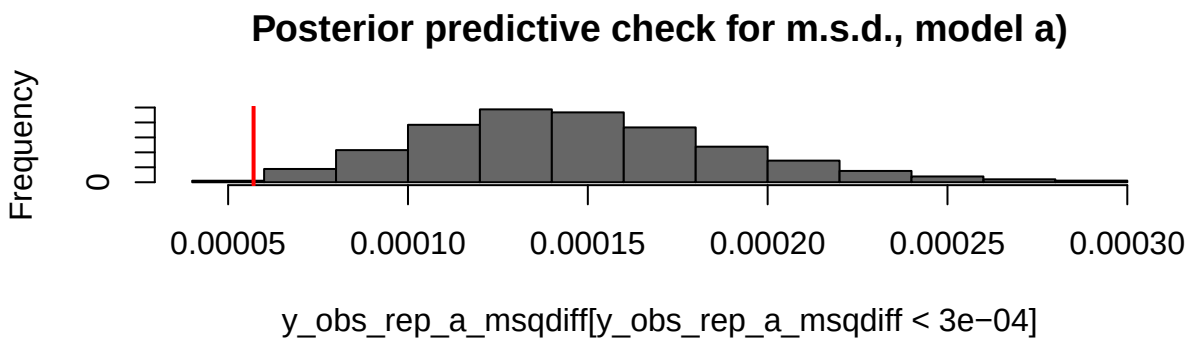
```
## Loading required package: fBasics
```

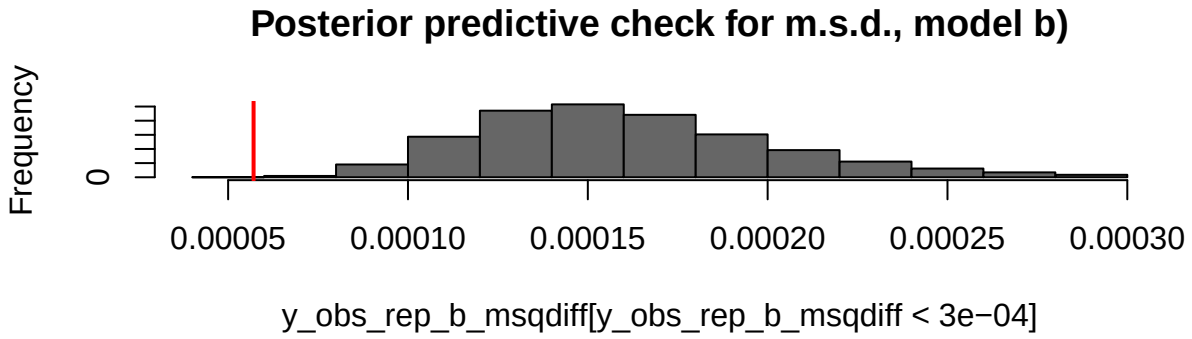
```
y_obs_rep_a_skewness=apply(y_obs_rep_a,1,skewness)
y_obs_rep_b_skewness=apply(y_obs_rep_b,1,skewness)

par(mfrow=c(2,1))
hist(y_obs_rep_a_skewness,col="gray40",
     main="Posterior predictive check for skewness, model a)", breaks=15)
abline(v=skewness(y_obs),col="red",lwd=2)

hist(y_obs_rep_b_skewness,col="gray40",
     main="Posterior predictive check for skewness, model b)", breaks=80,xlim=c(0,1))
abline(v=skewness(y_obs),col="red",lwd=2)
```






```
mean_square_diff<-function(v){l=length(v); return(mean((v[2:l]-v[1:(l-1)])^2)); }

y_obs_rep_a_msqdiff=apply(y_obs_rep_a,1,mean_square_diff)
y_obs_rep_b_msqdiff=apply(y_obs_rep_b,1,mean_square_diff)

par(mfrow=c(2,1))
hist(y_obs_rep_a_msqdiff[y_obs_rep_a_msqdiff<3e-4],col="gray40",
     main="Posterior predictive check for m.s.d., model a)", breaks=15)
abline(v=mean_square_diff(y_obs),col="red",lwd=2)

hist(y_obs_rep_b_msqdiff[y_obs_rep_b_msqdiff<3e-4],col="gray40",
     main="Posterior predictive check for m.s.d., model b)", breaks=15)
abline(v=mean_square_diff(y_obs),col="red",lwd=2)
```





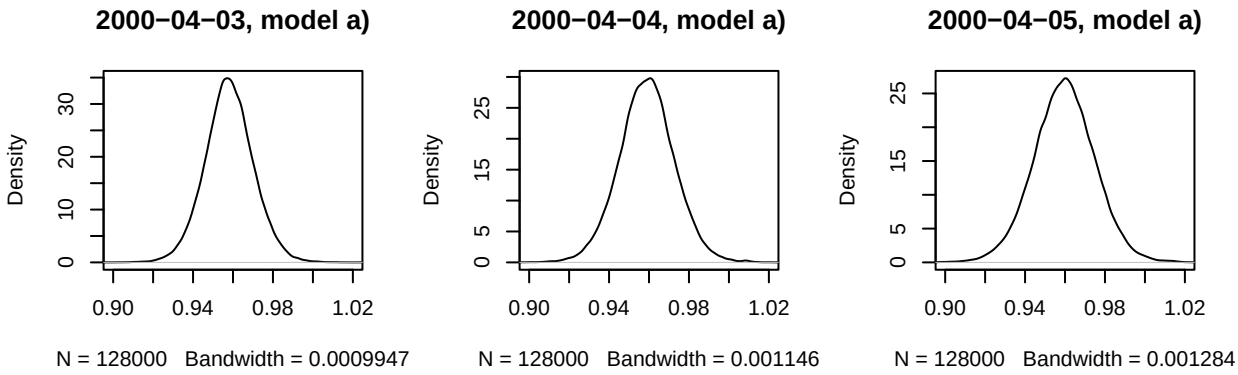
As we can see, both models a) and b) perform well for median and skewness. In terms of mean square difference, both seem to have significantly larger m.s.d. values in the posterior replicate samples than in the true data, especially for the second model b). This means that they tend to overestimate the volatility in y. Using more sophisticated models with higher order auto regressive terms and additional covariates containing relevant information might help to improve the fit on the data.

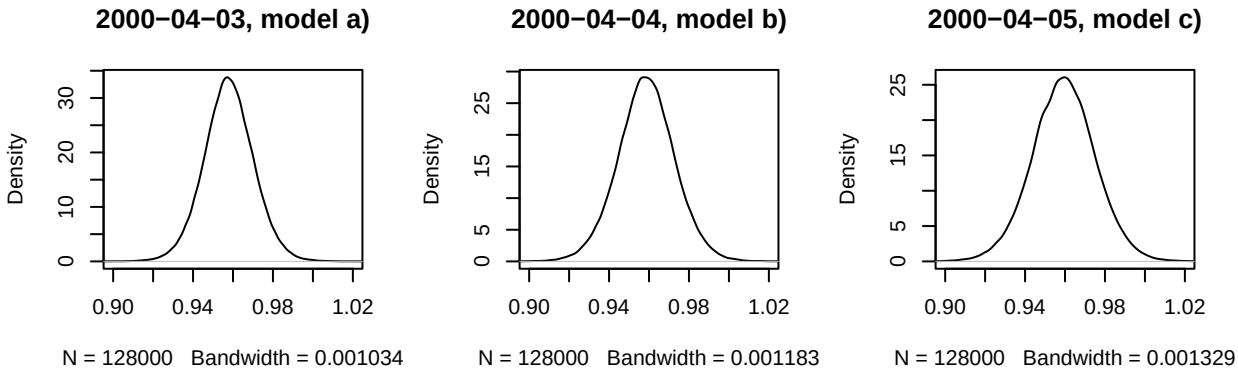
d)[10 marks]

Based on your models a) and b), plot the posterior predictive densities of the USD/EUR rate on the dates 2000-04-03, 2000-04-04 and 2000-04-05 (the next 3 days after the period considered). Compute the posterior means and 95% credible intervals. Discuss the results.

We included the subsequent 3 days in the model as unobserved parameters, so we can plot their density from the MCMC samples.

```
ypred_a=extract(res1a)$y[,(n_days+2):(n_days+4)];
ypred_b=extract(res1b)$y[,(n_days+2):(n_days+4)];
par(mfrow=c(2,3))
plot(density(ypred_a[,1]),xlim=c(0.9,1.02), main="2000-04-03, model a)")
plot(density(ypred_a[,2]),xlim=c(0.9,1.02), main="2000-04-04, model a)")
plot(density(ypred_a[,3]),xlim=c(0.9,1.02), main="2000-04-05, model a)")
plot(density(ypred_b[,1]),xlim=c(0.9,1.02), main="2000-04-03, model a)")
plot(density(ypred_b[,2]),xlim=c(0.9,1.02), main="2000-04-04, model b)")
plot(density(ypred_b[,3]),xlim=c(0.9,1.02), main="2000-04-05, model c)")
```






```
cat("2000-04-03, model a), mean: ", mean(ypred_a[,1]), ", std. deviation:", sd(ypred_a[,1]),
    ".\n 95% credible interval: [", quantile(ypred_a[,1],prob=c(0.025,0.975)),"]\n")
```

```
## 2000-04-03, model a), mean:  0.958079 , std. deviation: 0.01218852 .
##  95% credible interval: [ 0.9340785 0.9824931 ]
```

```
cat("2000-04-04, model a), mean: ", mean(ypred_a[,2]), ", std. deviation:", sd(ypred_a[,2]),
    ".\n 95% credible interval: [", quantile(ypred_a[,2],prob=c(0.025,0.975)),"]\n")
```

```
## 2000-04-04, model a), mean:  0.9589046 , std. deviation: 0.01398833 .
##  95% credible interval: [ 0.9313243 0.9866037 ]
```

```
cat("2000-04-05, model a), mean: ", mean(ypred_a[,3]), ", std. deviation:", sd(ypred_a[,3]),
    ".\n 95% credible interval: [", quantile(ypred_a[,3],prob=c(0.025,0.975)),"]\n")
```

```
## 2000-04-05, model a), mean:  0.9596348 , std. deviation: 0.0155117 .
##  95% credible interval: [ 0.9289029 0.9900925 ]
```

```
cat("2000-04-03, model b), mean: ", mean(ypred_b[,1]), ", std. deviation:", sd(ypred_b[,1]),
    ".\n 95% credible interval: [", quantile(ypred_b[,1],prob=c(0.025,0.975)),"]\n")
```

```
## 2000-04-03, model b), mean:  0.9578845 , std. deviation: 0.01260457 .
##  95% credible interval: [ 0.9331186 0.9828639 ]
```

```
cat("2000-04-04, model b), mean: ", mean(ypred_b[,2]), ", std. deviation:", sd(ypred_b[,2]),
    ".\n 95% credible interval: [", quantile(ypred_b[,2],prob=c(0.025,0.975)),"]\n")
```

```
## 2000-04-04, model b), mean:  0.9586398 , std. deviation: 0.01440782 .
##  95% credible interval: [ 0.9300562 0.9870911 ]
```

```
cat("2000-04-05, model b), mean: ", mean(ypred_b[,3]), ", std. deviation:", sd(ypred_b[,3]),
    ".\n 95% credible interval: [", quantile(ypred_b[,3],prob=c(0.025,0.975)),"]\n")
```

```
## 2000-04-05, model b), mean:  0.9593279 , std. deviation: 0.01600685 .
##  95% credible interval: [ 0.927595 0.9907107 ]
```

The results are quite similar for the two models. In both cases, the distributions become more spread out (i.e. higher standard deviation) when we try to predict further ahead in the future. This is to be expected as there are more and more uncertainty.

e)[10 marks]

In this question, we are going to look use a multivariate stochastic volatility model with leverage to study the USD/EUR and GBP/EUR exchange rates jointly. The model is described as follows,

$$\begin{aligned} \boldsymbol{y}_t &= \boldsymbol{\beta}_0 + \boldsymbol{\beta}_1 \boldsymbol{y}_{t-1} + \exp(h_t/2)\boldsymbol{\epsilon}_t \quad \text{for} \quad 1 \le t \le T, \\ \boldsymbol{h}_{t+1} &= \boldsymbol{\phi}(\boldsymbol{h}_t) + \boldsymbol{\eta}_t \quad \text{for} \quad 0 \le t \le T, \quad h_0 \sim N(\mu, I), \\ (\epsilon_t, \eta_t) &\sim N(0, \Sigma). \end{aligned}$$

Here I denotes the 2 x 2 identity matrix, $\boldsymbol{y}_t, \boldsymbol{\beta}_0, \boldsymbol{h}_t, \boldsymbol{\eta}_t, \boldsymbol{\epsilon}_t$ are 2 dimensional vectors, β_1 and ϕ are 2 x 2 matrices, Σ is a 4 x 4 covariance matrix. At each time step t, the two components of y_t will be used to model the USD/EUR and GBP/EUR exchange rates, respectively.

Implement this model in JAGS or Stan.

Discuss your choices for priors for every parameter.

Fit the model, do convergence diagnostics, print out the summary of the results, and discuss them.

In the model, $(\epsilon_t, \eta_t) \sim N(0, \Sigma)$, so these variables follow a multivariate normal distribution. Let $\Sigma = \begin{pmatrix} \Sigma_{(11)} & \Sigma_{(12)} \\ \Sigma_{(21)} & \Sigma_{(22)} \end{pmatrix}$, where $\Sigma_{(11)}, \Sigma_{(12)}, \Sigma_{(21)}, \Sigma_{(22)}$ are 2×2 matrices (this is the so-called block matrix form of Σ). Then it is easy to see that the marginal distribution of ϵ_t is $N(0, \Sigma_{(11)})$ and the marginal distribution of η_t is $N(0, \Sigma_{(22)})$. Based on the formula on https://statproofbook.github.io/P/mvn-cond, the conditional distribution of $\epsilon_t | \eta_t \sim N(\Sigma_{(12)}\Sigma_{(22)}^{-1}\eta_t, \Sigma_{(11)} - \Sigma_{(12)}\Sigma_{(22)}^{-1}\Sigma_{(21)})$. This is used in our definition of ϵ_t in the model code.

We can decompose $\Sigma = D\Omega D$, where D is a diagonal matrix with the square root of the eigenvalues, and Ω is a correlation matrix. We put a Cauchy prior with mean zero and scale 0.02 on the diagonal elements of D (we expect standard deviation to be not bigger than 0.02 based on typical daily fluctuations) and a Lewandowski-Kurowicka-Joe (LKJ) prior with parameter 1 on Ω (this corresponds to uniform distribution amongst correlation matrices), similarly to how it was done in Lecture 3.

The rest of the code is quite similar to the code in part a), except that we are using two dimensional matrices in terms of vectors for y, and use matrices or vectors of the required size for the other model parameters. Similarly to part a), the components of β_1 are chosen to have uniform (-1,1) priors (for stability), and we set N(0,1) priors for the components β_0 (as we expect them to be in this order of magnitude). For $(y_0)_1$ (USD), we choose $N(0.99, 0.01^2)$ prior as before, for $(y_0)_2$ (GBP), we select $N(1.61, 0.02^2)$ prior based on the GBP/EUR exchange rates for the last days of December 1999 available on https://www.poundsterlinglive.com/bank-of-england-spot/historical-spot-exchange-rates/gbp/GBP-to-EUR-1999. Here we do not standardize h, but use it as a parameter directly.

```
#model in STAN language
model_string_1e <-
"data{
int<lower=0> n_days;
int<lower=0> n_observed;
int<lower=0> n_pred;
int<lower=0> d; #number of different exchange rates considered, 2 in this question
matrix[d,n_observed] y_obs;
```

```
array[n_days] int obs_or_not;
}

parameters
{
  matrix[d,n_days+n_pred-n_observed+1] y_missing; //y_missing[1] corresponds to y_0
  matrix[d,n_days+n_pred+2] h; //h[1] corresponds to h_0 in the formula
  matrix[d,d] phi;
  vector[d] beta0;
  matrix<lower=-1,upper=1>[d,d] beta1;

  corr_matrix[2*d] Omega;        // correlation matrix of Sigma
  vector<lower=0>[2*d] Sqrt_Sigma_eig;
  // square root of eigenvalues of covariance matrix, with positivity constraint
}

transformed parameters
{
  matrix[2*d,2*d] Sigma=quad_form_diag(Omega, Sqrt_Sigma_eig);
  matrix[d,d] Sigma11=Sigma[1:d,1:d];
  matrix[d,d] Sigma12=Sigma[1:d,(d+1):(2*d)];
  matrix[d,d] Sigma21=Sigma[(d+1):(2*d),1:d];
  matrix[d,d] Sigma22=Sigma[(d+1):(2*d),(d+1):(2*d)];
  matrix[d,d] invSigma22;
  matrix[d,d] M1;
  matrix[d,d] M2;
  matrix[d,d] cholSigma22;
  matrix[d,d] cholM2;

  invSigma22=inverse(Sigma[(d+1):(2*d),(d+1):(2*d)]);
  M1=Sigma12 * invSigma22;
  M2=Sigma11-Sigma12 * invSigma22 * Sigma21;
  cholSigma22=cholesky_decompose(Sigma22);
  cholM2=cholesky_decompose(M2);

  matrix[d,n_days+n_pred+1] eta;
  eta[1:d,1]=rep_vector(1.0,d);
  eta[1:d,2:(n_days+n_pred+1)]=h[1:d,3:(n_days+n_pred+2)]-phi*h[1:d,2:(n_days+n_pred+1)];

  matrix[d,n_days+n_pred+1] y;
  //y[1] corresponds to y_0 in the formula
  y[1:d,1]=y_missing[1:d,1];
  {
    int ind_obs=1;
    int ind_missing=2;

    for(i in 1:n_days)
    {
      if(obs_or_not[i]==1)
      {
        y[1:d,i+1]=y_obs[1:d,ind_obs];
        ind_obs=ind_obs+1;
      }
```

```
      else
      {
        y[1:d,i+1]=y_missing[1:d,ind_missing];
        ind_missing=ind_missing+1;
      }
    }
    for(i in (n_days+1):(n_days+n_pred))
    {
        y[1:d,i+1]=y_missing[1:d,ind_missing];
        ind_missing=ind_missing+1;
    }
  }
}

model{
  Omega ~ lkj_corr(1);
  Sqrt_Sigma_eig ~ cauchy(0, 0.02);

  beta0~normal(0,1);
  h[1:d,1]~normal(0,1);

  for(i in 1:d)
  {
    for(j in 1:d)
    {
      phi[i,j]~uniform(-1,1);
      beta1[i,j]~uniform(-1,1);
    }
  }

  y[1,1]~normal(0.99,0.01);
  y[1,2]~normal(1.61,0.02);

  for(t in 1:(n_days+n_pred))
  {
      h[1:d,t+1]~multi_normal(phi*h[1:d,t],Sigma22);
  }
  for(t in 1:(n_days+n_pred))
  {
    y[1:d,t+1]~multi_normal_cholesky(beta0+beta1 * y[1:d,t]+exp(h[1:d,t+1]/2).* (M1 * eta[1:d,t+1]),
    diag_matrix(exp(h[1:d,t+1]/2))*cholM2);
  }

}

generated quantities
{
  matrix[d,n_observed] y_obs_rep;
  {
    int ind_obs=1;

    for(t in 1:n_days)
    {
```

```
      if(obs_or_not[t]==1)
      {
        y_obs_rep[1:d,ind_obs]= multi_normal_cholesky_rng(beta0+beta1 * y[1:d,t]+exp(h[1:d,t+1]/2) .* (
        diag_matrix(exp(h[1:d,t+1]/2))*cholM2);
        ind_obs=ind_obs+1;
      }
    }

  }
}
"

#data

#finding dates up to 2000-04-02
n_days=as.numeric(as.Date("2000-04-02")-as.Date("2000-01-03"))+1;
n_pred=3;
y_obs=t(cbind(exrates$USD[1:n_observed],exrates$GBP[1:n_observed]));
d=2;
obs_or_not=rep(0,n_days)
for (it in 1:n_observed)
{
  date_it=exrates$date[it]-as.Date("2000-01-03")+1
  obs_or_not[date_it]=1
}

data=list(n_days=n_days,n_observed=n_observed, y_obs=y_obs, obs_or_not=obs_or_not,n_pred=n_pred,d=d)

fname="model_1e.stan"
cat(model_string_1e,file=fname,append=FALSE)
# list with data and hyperparameters

#passing the model string to STAN
res1e<- stan(file = fname, data = data,
           # Below are optional arguments
           iter = 6000,
           #iter is the number of iterations, including the burn-in
           #the burn-in period is set to iter/2 by default, it can be set to
           chains = 8,cores = parallel::detectCores(),refresh=0);
```

```
## Warning: There were 2676 divergent transitions after warmup. See
## https://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
## to find out why this is a problem and how to eliminate them.

## Warning: There were 21324 transitions after warmup that exceeded the maximum treedepth. Increase max
## https://mc-stan.org/misc/warnings.html#maximum-treedepth-exceeded

## Warning: There were 8 chains where the estimated Bayesian Fraction of Missing Information was low. S
## https://mc-stan.org/misc/warnings.html#bfmi-low

## Warning: Examine the pairs() plot to diagnose sampling problems

## Warning: The largest R-hat is NA, indicating chains have not mixed.
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#r-hat
```

```
## Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians may be
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#bulk-ess
```

```
## Warning: Tail Effective Samples Size (ESS) is too low, indicating posterior variances and tail quanti
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#tail-ess
```

```
print(res1e,pars=c("phi", "beta0", "beta1", "Sigma"))
```

```
## Inference for Stan model: anon_model.
## 8 chains, each with iter=6000; warmup=3000; thin=1;
## post-warmup draws per chain=3000, total post-warmup draws=24000.
##
##             mean se_mean   sd  2.5%   25%   50%  75% 97.5% n_eff Rhat
## phi[1,1]    0.03    0.27 0.61 -0.94 -0.46 -0.03 0.63  0.91     5 3.03
## phi[1,2]   -0.03    0.28 0.60 -0.93 -0.48 -0.15 0.57  0.96     4 3.84
## phi[2,1]   -0.12    0.20 0.49 -0.94 -0.49 -0.19 0.17  0.92     6 2.16
## phi[2,2]   -0.04    0.12 0.43 -0.83 -0.37 -0.01 0.20  0.76    13 1.94
## beta0[1]    0.04    0.01 0.06 -0.08  0.00  0.05 0.08  0.16    19 1.33
## beta0[2]    0.07    0.01 0.03  0.02  0.05  0.07 0.10  0.12    11 1.50
## beta1[1,1]  0.94    0.01 0.04  0.86  0.91  0.94 0.96  0.99    27 1.33
## beta1[1,2]  0.03    0.02 0.12 -0.19 -0.06  0.03 0.11  0.26    25 1.40
## beta1[2,1]  0.00    0.00 0.02 -0.04 -0.02  0.00 0.01  0.03    33 1.26
## beta1[2,2]  0.89    0.01 0.06  0.77  0.85  0.89 0.94  0.99    21 1.45
## Sigma[1,1]  0.00    0.00 0.00  0.00  0.00  0.00 0.00  0.00    32 1.33
## Sigma[1,2]  0.00    0.00 0.00  0.00  0.00  0.00 0.00  0.00    29 1.36
## Sigma[1,3]  0.00    0.00 0.00  0.00  0.00  0.00 0.00  0.00    16 1.52
## Sigma[1,4]  0.00    0.00 0.00  0.00  0.00  0.00 0.00  0.00     9 1.59
## Sigma[2,1]  0.00    0.00 0.00  0.00  0.00  0.00 0.00  0.00    29 1.36
## Sigma[2,2]  0.00    0.00 0.00  0.00  0.00  0.00 0.00  0.00    27 1.33
## Sigma[2,3]  0.00    0.00 0.00  0.00  0.00  0.00 0.00  0.00    27 1.37
## Sigma[2,4]  0.00    0.00 0.00  0.00  0.00  0.00 0.00  0.00     9 1.69
## Sigma[3,1]  0.00    0.00 0.00  0.00  0.00  0.00 0.00  0.00    16 1.52
## Sigma[3,2]  0.00    0.00 0.00  0.00  0.00  0.00 0.00  0.00    27 1.37
## Sigma[3,3]  0.01    0.00 0.02  0.00  0.00  0.00 0.01  0.06    12 1.46
## Sigma[3,4]  0.00    0.00 0.01 -0.01  0.00  0.00 0.00  0.01    56 1.16
## Sigma[4,1]  0.00    0.00 0.00  0.00  0.00  0.00 0.00  0.00     9 1.59
## Sigma[4,2]  0.00    0.00 0.00  0.00  0.00  0.00 0.00  0.00     9 1.69
## Sigma[4,3]  0.00    0.00 0.01 -0.01  0.00  0.00 0.00  0.01    56 1.16
## Sigma[4,4]  0.04    0.03 0.07  0.00  0.01  0.01 0.04  0.27     6 2.87
##
## Samples were drawn using NUTS(diag_e) at Sun Mar 26 21:27:00 2023.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

Due to the complexity of this model, the chain is mixing quite slowly, and convergence requires a large sample size and long burn-in period. The posterior mean of $(\beta_1)_{11}$ and $(\beta_1)_{22}$ are close to 1 as before. The non-diagonal terms have non-zero means, indicating interactions between these two exchange rates.

Problem 2 - NBA data

In this problem, we are going to construct a predictive model for NBA games.

We start by loading the dataset.

```
games<-read.csv("games.csv")
teams<-read.csv("teams.csv")
```

games.csv contains the information about games such as GAME_DATE, SEASON, HOME_TEAM_ID, VISITOR_TEAM_ID, PTS_home (final score for home team) and PTS_away (final score for away team).

teams.csv contains the names of each team, i.e. the names corresponding to each team ID.

We are going to fit some Bayesian linear regression models on the scores of each team.

You can use either INLA, JAGS or Stan.

a)[10 marks]

The dataset contains data from 20 seasons, but we are going to focus on only one, the 2021 season.
Please only keep games where SEASON is 2021 in the dataset, and remove all other seasons. Please order the games according to the date of occurrence (they are not ordered like that in the dataset).

The scores are going to be assumed to follow a linear Gaussian model,

$$S_g^H \sim N(\mu_g^H, \sigma^2), \quad S_g^A \sim N(\mu_g^A, \sigma^2).$$

Here S_g^H denotes the final score of the home team in game g, and S_g^A denotes the final score of the away team in game g.

Note that the true scores can only take non-negative integer values, so the Gaussian distribution is not perfect, but it can still be used nevertheless.

The means for the scores are going to be modeled as a combination of three terms: attacking strength, defending ability, and whether the team is playing at home, or away. For each team, we denote their attacking strength parameter by a_{team}, their defending strength parameter by d_{team}, and the effect of playing at home as h. This quantifies the effect of playing at home on the expected number of goals scored. Our model is the following (μ_g^H is for the goals scored by the home team, and is μ_g^A is for the away team):

$\mu_g^H = \beta_0 + a_{home.team} + d_{away.team} + h$

$\mu_g^A = \beta_0 + a_{away.team} + d_{home.team}$

Implement this model. Select your own prior distributions for the parameters, and discuss the reason for using those priors.

Obtain the summary statistics for the posterior distribution of the model parameters.

Evaluate the root mean square error (RMSE) of your posterior means versus the true scores.

Interpret the results.

We implement the model in INLA, using a similar approach as Question 3 in Workshop 2. We set the mean and the standard deviation for the prior for the intercept based on the mean and standard deviation of the scores in 2020. We select mean 0 and standard deviation 10 for the regression coefficients. We choose Gamma(0.1,0.1) prior for the precision τ of the Gaussian model, this is a non-informative prior with a spike near zero and most of its mass in the [0,3] region.

```
library(INLA)
```

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

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

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

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

```
## This is INLA_23.02.04 built 2023-02-04 09:22:22 UTC.
##  - See www.r-inla.org/contact-us for how to get help.
##  - To enable PARDISO sparse library; see inla.pardiso()
```

```
mean_season_2020=mean(c(games[games$SEASON==2020,]$PTS_away,games[games$SEASON==2020,]$home));
sd_season_2020=sd(c(games[games$SEASON==2020,]$PTS_away,games[games$SEASON==2020,]$PTS_home));
games=games[games$SEASON==2021,]
games=games[order(games$GAME_DATE_EST),]
HOME_TEAM_NAME<-function(HOME_TEAM_ID){teams$NICKNAME[which(teams$TEAM_ID==HOME_TEAM_ID)]}
AWAY_TEAM_NAME<-function(AWAY_TEAM_ID){teams$NICKNAME[which(teams$TEAM_ID==AWAY_TEAM_ID)]}
games$HOME_TEAM<-apply(as.array(games$HOME_TEAM_ID),1,HOME_TEAM_NAME);
games$AWAY_TEAM<-apply(as.array(games$VISITOR_TEAM_ID),1,AWAY_TEAM_NAME);

y=c(games$PTS_home, games$PTS_away)
G=nrow(games)
HT_char=as.character(games$HOME_TEAM)
AT_char=as.character(games$AWAY_TEAM)
attack=as.factor(c(HT_char,AT_char))
defense=as.factor(c(AT_char,HT_char))
```

```r
playing.at.home=c(rep(1,G),rep(0,G))

data=data.frame(y,attack,defense,playing.at.home)
prior.beta <- list(mean.intercept = 110, prec.intercept = 1e-2,
                   mean = 0, prec = 1e-2)
prec.prior <- list(prec=list(prior = "loggamma", param = c(0.1, 0.1)))

m1=inla(formula=y~1+attack+defense+playing.at.home, data=data, family="gaussian",
        control.compute = list(dic = T,cpo=T,config=T),
        control.family=list(hyper=prec.prior),control.fixed=prior.beta)
summary(m1)
```

```
##
## Call:
##    c("inla.core(formula = formula, family = family, contrasts = contrasts,
##    ", " data = data, quantiles = quantiles, E = E, offset = offset, ", "
##    scale = scale, weights = weights, Ntrials = Ntrials, strata = strata,
##    ", " lp.scale = lp.scale, link.covariates = link.covariates, verbose =
##    verbose, ", " lincomb = lincomb, selection = selection, control.compute
##    = control.compute, ", " control.predictor = control.predictor,
##    control.family = control.family, ", " control.inla = control.inla,
##    control.fixed = control.fixed, ", " control.mode = control.mode,
##    control.expert = control.expert, ", " control.hazard = control.hazard,
##    control.lincomb = control.lincomb, ", " control.update =
##    control.update, control.lp.scale = control.lp.scale, ", "
##    control.pardiso = control.pardiso, only.hyperparam = only.hyperparam,
##    ", " inla.call = inla.call, inla.arg = inla.arg, num.threads =
##    num.threads, ", " blas.num.threads = blas.num.threads, keep = keep,
##    working.directory = working.directory, ", " silent = silent, inla.mode
##    = inla.mode, safe = FALSE, debug = debug, ", " .parent.frame =
##    .parent.frame)")
## Time used:
##     Pre = 4.84, Running = 7.93, Post = 0.758, Total = 13.5
## Fixed effects:
##                          mean    sd 0.025quant 0.5quant 0.975quant    mode kld
## (Intercept)           105.013 1.424    102.219  105.013    107.805 105.014   0
## attackBucks             5.093 1.532      2.089    5.093      8.099   5.093   0
## attackBulls             2.168 1.568     -0.908    2.168      5.244   2.167   0
## attackCavaliers        -2.109 1.578     -5.204   -2.110      0.986  -2.110   0
## attackCeltics           1.373 1.487     -1.544    1.373      4.291   1.372   0
## attackClippers         -1.212 1.588     -4.327   -1.213      1.904  -1.213   0
## attackGrizzlies         5.342 1.533      2.336    5.342      8.350   5.342   0
## attackHawks             4.285 1.556      1.234    4.285      7.338   4.285   0
## attackHeat              0.202 1.508     -2.754    0.201      3.159   0.201   0
## attackHornets           5.547 1.588      2.433    5.546      8.661   5.546   0
## attackJazz              3.568 1.570      0.490    3.568      6.648   3.568   0
## attackKings             1.431 1.599     -1.704    1.431      4.567   1.430   0
## attackKnicks           -2.364 1.593     -5.488   -2.364      0.762  -2.365   0
## attackLakers            2.441 1.588     -0.673    2.441      5.556   2.440   0
## attackMagic            -4.903 1.593     -8.027   -4.904     -1.778  -4.904   0
## attackMavericks        -0.671 1.520     -3.651   -0.671      2.310  -0.672   0
## attackNets              4.246 1.568      1.172    4.246      7.321   4.246   0
## attackNuggets           3.433 1.570      0.355    3.433      6.513   3.433   0
## attackPacers            2.464 1.593     -0.660    2.463      5.588   2.463   0
```

```
## attackPelicans        -0.106 1.560     -3.165   -0.107      2.954  -0.107   0
## attackPistons         -3.876 1.594     -7.001   -3.876     -0.750  -3.877   0
## attackRaptors          0.473 1.562     -2.590    0.473      3.538   0.473   0
## attackRockets          0.613 1.596     -2.518    0.613      3.745   0.612   0
## attackSpurs            3.651 1.586      0.540    3.651      6.763   3.650   0
## attackSuns             4.779 1.540      1.760    4.779      7.800   4.779   0
## attackThunder         -5.499 1.598     -8.632   -5.499     -2.364  -5.500   0
## attackTimberwolves     6.430 1.565      3.362    6.430      9.500   6.430   0
## attackTrail Blazers   -3.379 1.598     -6.513   -3.379     -0.243  -3.379   0
## attackWarriors         2.701 1.500     -0.241    2.701      5.644   2.700   0
## attackWizards         -0.188 1.592     -3.309   -0.188      2.934  -0.189   0
## defenseBucks           3.630 1.532      0.625    3.630      6.635   3.630   0
## defenseBulls           3.957 1.568      0.881    3.957      7.033   3.957   0
## defenseCavaliers      -1.702 1.578     -4.797   -1.702      1.393  -1.703   0
## defenseCeltics        -3.385 1.487     -6.302   -3.385     -0.468  -3.386   0
## defenseClippers        1.095 1.588     -2.020    1.095      4.210   1.095   0
## defenseGrizzlies       1.487 1.533     -1.520    1.487      4.494   1.487   0
## defenseHawks           4.479 1.556      1.427    4.479      7.531   4.479   0
## defenseHeat           -2.636 1.507     -5.592   -2.636      0.320  -2.637   0
## defenseHornets         7.989 1.588      4.876    7.989     11.103   7.989   0
## defenseJazz            0.338 1.570     -2.741    0.338      3.417   0.338   0
## defenseKings           7.754 1.598      4.619    7.754     10.889   7.754   0
## defenseKnicks         -0.824 1.593     -3.948   -0.824      2.301  -0.824   0
## defenseLakers          7.794 1.588      4.680    7.794     10.909   7.794   0
## defenseMagic           4.199 1.593      1.075    4.199      7.323   4.199   0
## defenseMavericks      -3.440 1.520     -6.421   -3.440     -0.460  -3.440   0
## defenseNets            4.887 1.568      1.813    4.887      7.962   4.887   0
## defenseNuggets         3.474 1.570      0.395    3.473      6.552   3.473   0
## defensePacers          7.557 1.593      4.432    7.557     10.681   7.557   0
## defensePelicans        2.446 1.560     -0.613    2.446      5.505   2.446   0
## defensePistons         5.098 1.593      1.973    5.098      8.223   5.098   0
## defenseRaptors         0.393 1.562     -2.670    0.393      3.457   0.393   0
## defenseRockets        10.610 1.596      7.479   10.610     13.741  10.610   0
## defenseSpurs           5.221 1.586      2.109    5.221      8.332   5.221   0
## defenseSuns            0.216 1.540     -2.803    0.216      3.236   0.216   0
## defenseThunder         4.059 1.598      0.925    4.059      7.193   4.059   0
## defenseTimberwolves    5.637 1.565      2.568    5.636      8.705   5.636   0
## defenseTrail Blazers   7.366 1.598      4.232    7.366     10.501   7.366   0
## defenseWarriors       -1.794 1.500     -4.736   -1.794      1.148  -1.794   0
## defenseWizards         5.086 1.592      1.965    5.086      8.208   5.086   0
## playing.at.home        2.084 0.444      1.214    2.084      2.955   2.084   0
##
## Model hyperparameters:
##                                       mean   sd 0.025quant 0.5quant
## Precision for the Gaussian observations 0.007 0.00      0.007    0.007
##                                       0.975quant  mode
## Precision for the Gaussian observations      0.008 0.007
##
## Deviance Information Criterion (DIC) ...............: 21613.87
## Deviance Information Criterion (DIC, saturated) ....: 2842.06
## Effective number of parameters .....................: 59.63
##
## Marginal log-Likelihood:  -11000.08
## CPO, PIT is computed
```

```
## Posterior summaries for the linear predictor and the fitted values are computed
## (Posterior marginals needs also 'control.compute=list(return.marginals.predictor=TRUE)')
```

The model fit indicates that the intercept has mean 105, meaning that the typical number of goals per team is close to this, which is realistic. The playing.at.home coefficient has mean 2, indicating that there is a home advantage of around 2 goals. Timberwolves has the best attack (largest value), while Mavericks has the best defense (smallest value).

```
library(Metrics)
rmse(y, m1$summary.fitted.values$mean)
```

```
## [1] 11.58483
```

The RMSE of this model is 11.58484.

We also compute the NLSCPO to compare models (this was not required in the question).

```
m1.nlscpo=-sum(log(m1$cpo$cpo))
cat("NLSCPO of model 1:",m1.nlscpo,"\n")
```

```
## NLSCPO of model 1: 10807.38
```

b)[10 marks] In part a), the model assumed that the home effect is the same for each team. In this part, we consider a team-specific home effect $h_{home.team}$,

$\mu_g^H = \beta_0 + a_{home.team} + d_{away.team} + h_{home.team}$

$\mu_g^A = \beta_0 + a_{away.team} + d_{home.team}$

Implement this model. Select your own prior distributions for the parameters, and discuss the reason for using those priors.

Obtain the summary statistics for the posterior distribution of the model parameters.

Evaluate the root mean square error (RMSE) of your posterior means versus the true scores.

Interpret the results.

This team specific home effect can be achieved by using the playing.at.home:attack term in place of playing.at.home in the INLA formula. We use the same priors as in part a).

```
prior.beta <- list(mean.intercept = 110, prec.intercept = 1e-2,
                   mean = 0, prec = 1e-2)
prec.prior <- list(prec=list(prior = "loggamma", param = c(0.1, 0.1)))

m2=inla(formula=y~1+attack+defense+playing.at.home:attack, data=data,
        family="gaussian",control.compute = list(dic = T,cpo=T,config=T),control.family=list(hyper=prec
summary(m2)
```

```
##
## Call:
##    c("inla.core(formula = formula, family = family, contrasts = contrasts,
##    ", " data = data, quantiles = quantiles, E = E, offset = offset, ", "
##    scale = scale, weights = weights, Ntrials = Ntrials, strata = strata,
##    ", " lp.scale = lp.scale, link.covariates = link.covariates, verbose =
##    verbose, ", " lincomb = lincomb, selection = selection, control.compute
##    = control.compute, ", " control.predictor = control.predictor,
##    control.family = control.family, ", " control.inla = control.inla,
##    control.fixed = control.fixed, ", " control.mode = control.mode,
##    control.expert = control.expert, ", " control.hazard = control.hazard,
##    control.lincomb = control.lincomb, ", " control.update =
##    control.update, control.lp.scale = control.lp.scale, ", "
```

```
##    control.pardiso = control.pardiso, only.hyperparam = only.hyperparam,
##    ", " inla.call = inla.call, inla.arg = inla.arg, num.threads =
##    num.threads, ", " blas.num.threads = blas.num.threads, keep = keep,
##    working.directory = working.directory, ", " silent = silent, inla.mode
##    = inla.mode, safe = FALSE, debug = debug, ", " .parent.frame =
##    .parent.frame)")
## Time used:
##     Pre = 4.82, Running = 4.04, Post = 1.03, Total = 9.89
## Fixed effects:
##                                    mean    sd 0.025quant 0.5quant
## (Intercept)                     104.728 1.574    101.641  104.729
## attackBucks                       6.006 2.010      2.064    6.006
## attackBulls                       1.004 2.074     -3.063    1.003
## attackCavaliers                  -1.001 2.073     -5.065   -1.001
## attackCeltics                     3.534 1.955     -0.299    3.534
## attackClippers                   -0.728 2.104     -4.854   -0.728
## attackGrizzlies                   4.883 2.013      0.936    4.883
## attackHawks                       2.694 2.045     -1.316    2.694
## attackHeat                        0.060 1.989     -3.839    0.060
## attackHornets                     8.722 2.087      4.630    8.722
## attackJazz                        1.137 2.064     -2.911    1.136
## attackKings                       2.458 2.105     -1.669    2.458
## attackKnicks                     -0.726 2.116     -4.875   -0.726
## attackLakers                      3.865 2.090     -0.233    3.865
## attackMagic                      -3.682 2.101     -7.801   -3.682
## attackMavericks                   0.372 1.981     -3.513    0.371
## attackNets                        7.612 2.072      3.548    7.612
## attackNuggets                     2.737 2.038     -1.260    2.737
## attackPacers                      1.302 2.102     -2.819    1.302
## attackPelicans                   -1.009 2.037     -5.003   -1.009
## attackPistons                    -3.141 2.101     -7.260   -3.141
## attackRaptors                     0.534 2.048     -3.482    0.534
## attackRockets                    -0.476 2.103     -4.600   -0.476
## attackSpurs                       4.115 2.089      0.019    4.115
## attackSuns                        4.749 2.026      0.777    4.749
## attackThunder                    -4.866 2.119     -9.022   -4.867
## attackTimberwolves                9.615 2.050      5.596    9.615
## attackTrail Blazers              -4.503 2.104     -8.629   -4.503
## attackWarriors                    1.775 1.980     -2.107    1.775
## attackWizards                    -1.605 2.101     -5.725   -1.605
## defenseBucks                      3.671 1.529      0.672    3.671
## defenseBulls                      3.945 1.565      0.876    3.945
## defenseCavaliers                 -1.668 1.575     -4.757   -1.669
## defenseCeltics                   -3.338 1.484     -6.248   -3.338
## defenseClippers                   1.096 1.585     -2.012    1.096
## defenseGrizzlies                  1.508 1.530     -1.492    1.508
## defenseHawks                      4.424 1.553      1.379    4.424
## defenseHeat                      -2.695 1.504     -5.646   -2.695
## defenseHornets                    8.003 1.585      4.895    8.003
## defenseJazz                       0.405 1.567     -2.667    0.405
## defenseKings                      7.716 1.595      4.588    7.716
## defenseKnicks                    -0.780 1.590     -3.898   -0.780
## defenseLakers                     7.838 1.584      4.731    7.838
## defenseMagic                      4.208 1.590      1.090    4.208
```

```
## defenseMavericks                     -3.388 1.517     -6.363   -3.388
## defenseNets                           4.818 1.565      1.749    4.817
## defenseNuggets                        3.465 1.567      0.392    3.465
## defensePacers                         7.663 1.589      4.546    7.663
## defensePelicans                       2.496 1.557     -0.557    2.496
## defensePistons                        5.167 1.590      2.049    5.167
## defenseRaptors                        0.443 1.559     -2.615    0.443
## defenseRockets                       10.599 1.593      7.475   10.599
## defenseSpurs                          5.357 1.583      2.252    5.357
## defenseSuns                           0.281 1.536     -2.732    0.281
## defenseThunder                        4.104 1.595      0.976    4.104
## defenseTimberwolves                   5.689 1.561      2.627    5.689
## defenseTrail Blazers                  7.336 1.595      4.208    7.336
## defenseWarriors                      -1.740 1.497     -4.675   -1.740
## defenseWizards                        5.143 1.588      2.029    5.143
## attack76ers:playing.at.home           2.896 2.035     -1.096    2.896
## attackBucks:playing.at.home           0.744 2.271     -3.710    0.744
## attackBulls:playing.at.home           4.932 2.361      0.300    4.932
## attackCavaliers:playing.at.home       0.339 2.386     -4.341    0.339
## attackCeltics:playing.at.home        -1.757 2.161     -5.994   -1.757
## attackClippers:playing.at.home        1.624 2.398     -3.081    1.624
## attackGrizzlies:playing.at.home       3.523 2.260     -0.910    3.523
## attackHawks:playing.at.home           5.882 2.338      1.296    5.882
## attackHeat:playing.at.home            2.882 2.199     -1.430    2.882
## attackHornets:playing.at.home        -3.911 2.412     -8.641   -3.911
## attackJazz:playing.at.home            7.531 2.350      2.922    7.531
## attackKings:playing.at.home           0.506 2.425     -4.249    0.506
## attackKnicks:playing.at.home         -0.660 2.424     -5.413   -0.660
## attackLakers:playing.at.home         -0.292 2.398     -4.996   -0.292
## attackMagic:playing.at.home           0.122 2.424     -4.633    0.122
## attackMavericks:playing.at.home       0.442 2.220     -3.912    0.442
## attackNets:playing.at.home           -4.169 2.361     -8.800   -4.169
## attackNuggets:playing.at.home         4.092 2.353     -0.524    4.092
## attackPacers:playing.at.home          4.969 2.424      0.215    4.969
## attackPelicans:playing.at.home        4.474 2.327     -0.091    4.474
## attackPistons:playing.at.home         1.113 2.424     -3.641    1.113
## attackRaptors:playing.at.home         2.494 2.338     -2.092    2.494
## attackRockets:playing.at.home         4.804 2.424      0.049    4.804
## attackSpurs:playing.at.home           1.666 2.399     -3.040    1.666
## attackSuns:playing.at.home            2.653 2.270     -1.800    2.653
## attackThunder:playing.at.home         1.332 2.424     -3.423    1.332
## attackTimberwolves:playing.at.home   -3.922 2.338     -8.507   -3.922
## attackTrail Blazers:playing.at.home   4.874 2.424      0.119    4.874
## attackWarriors:playing.at.home        4.411 2.170      0.155    4.411
## attackWizards:playing.at.home         5.488 2.424      0.733    5.488
##                                    0.975quant    mode kld
## (Intercept)                           107.814 104.729   0
## attackBucks                             9.949   6.005   0
## attackBulls                             5.072   1.003   0
## attackCavaliers                         3.065  -1.002   0
## attackCeltics                           7.368   3.534   0
## attackClippers                          3.400  -0.729   0
## attackGrizzlies                         8.831   4.882   0
## attackHawks                             6.705   2.693   0
```

```
## attackHeat                          3.961   0.060   0
## attackHornets                      12.815   8.721   0
## attackJazz                          5.186   1.136   0
## attackKings                         6.586   2.457   0
## attackKnicks                        3.424  -0.727   0
## attackLakers                        7.965   3.865   0
## attackMagic                         0.439  -3.683   0
## attackMavericks                     4.258   0.371   0
## attackNets                         11.677   7.612   0
## attackNuggets                       6.736   2.736   0
## attackPacers                        5.424   1.301   0
## attackPelicans                      2.988  -1.009   0
## attackPistons                       0.980  -3.141   0
## attackRaptors                       4.552   0.534   0
## attackRockets                       3.650  -0.477   0
## attackSpurs                         8.212   4.114   0
## attackSuns                          8.722   4.748   0
## attackThunder                      -0.709  -4.867   0
## attackTimberwolves                 13.636   9.615   0
## attackTrail Blazers                -0.375  -4.504   0
## attackWarriors                      5.658   1.774   0
## attackWizards                       2.516  -1.606   0
## defenseBucks                        6.671   3.671   0
## defenseBulls                        7.014   3.945   0
## defenseCavaliers                    1.420  -1.669   0
## defenseCeltics                     -0.427  -3.338   0
## defenseClippers                     4.204   1.096   0
## defenseGrizzlies                    4.509   1.508   0
## defenseHawks                        7.470   4.424   0
## defenseHeat                         0.255  -2.696   0
## defenseHornets                     11.111   8.003   0
## defenseJazz                         3.478   0.405   0
## defenseKings                       10.845   7.716   0
## defenseKnicks                       2.339  -0.780   0
## defenseLakers                      10.945   7.838   0
## defenseMagic                        7.326   4.208   0
## defenseMavericks                   -0.414  -3.388   0
## defenseNets                         7.886   4.817   0
## defenseNuggets                      6.538   3.465   0
## defensePacers                      10.781   7.663   0
## defensePelicans                     5.549   2.496   0
## defensePistons                      8.285   5.167   0
## defenseRaptors                      3.502   0.443   0
## defenseRockets                     13.724  10.599   0
## defenseSpurs                        8.462   5.357   0
## defenseSuns                         3.295   0.281   0
## defenseThunder                      7.231   4.104   0
## defenseTimberwolves                 8.752   5.689   0
## defenseTrail Blazers               10.464   7.336   0
## defenseWarriors                     1.196  -1.740   0
## defenseWizards                      8.258   5.143   0
## attack76ers:playing.at.home         6.889   2.895   0
## attackBucks:playing.at.home         5.198   0.744   0
## attackBulls:playing.at.home         9.562   4.932   0
```

```
## attackCavaliers:playing.at.home          5.019   0.339   0
## attackCeltics:playing.at.home            2.481  -1.757   0
## attackClippers:playing.at.home           6.328   1.624   0
## attackGrizzlies:playing.at.home          7.955   3.523   0
## attackHawks:playing.at.home             10.467   5.882   0
## attackHeat:playing.at.home               7.194   2.882   0
## attackHornets:playing.at.home            0.821  -3.911   0
## attackJazz:playing.at.home              12.139   7.531   0
## attackKings:playing.at.home              5.262   0.506   0
## attackKnicks:playing.at.home             4.094  -0.660   0
## attackLakers:playing.at.home             4.412  -0.293   0
## attackMagic:playing.at.home              4.876   0.122   0
## attackMavericks:playing.at.home          4.795   0.442   0
## attackNets:playing.at.home               0.463  -4.169   0
## attackNuggets:playing.at.home            8.707   4.092   0
## attackPacers:playing.at.home             9.723   4.969   0
## attackPelicans:playing.at.home           9.038   4.474   0
## attackPistons:playing.at.home            5.867   1.113   0
## attackRaptors:playing.at.home            7.079   2.494   0
## attackRockets:playing.at.home            9.558   4.804   0
## attackSpurs:playing.at.home              6.371   1.666   0
## attackSuns:playing.at.home               7.106   2.653   0
## attackThunder:playing.at.home            6.087   1.332   0
## attackTimberwolves:playing.at.home       0.664  -3.923   0
## attackTrail Blazers:playing.at.home      9.629   4.874   0
## attackWarriors:playing.at.home           8.667   4.411   0
## attackWizards:playing.at.home           10.242   5.488   0
##
## Model hyperparameters:
##                                          mean   sd 0.025quant 0.5quant
## Precision for the Gaussian observations 0.007 0.00      0.007    0.007
##                                         0.975quant  mode
## Precision for the Gaussian observations      0.008 0.007
##
## Deviance Information Criterion (DIC) ...............: 21616.92
## Deviance Information Criterion (DIC, saturated) ....: 2860.92
## Effective number of parameters .....................: 86.37
##
## Marginal log-Likelihood:  -11017.64
## CPO, PIT is computed
## Posterior summaries for the linear predictor and the fitted values are computed
## (Posterior marginals needs also 'control.compute=list(return.marginals.predictor=TRUE)')
```

The results indicate that there are significant differences in the home effect between teams.

```
rmse(y, m2$summary.fitted.values$mean)
```

```
## [1] 11.47807
```

The RMSE has slightly improved compared to the model in part a).

This was not required, but we also compute the NLSCPO.

```
m2.nlscpo=-sum(log(m2$cpo$cpo))
cat("NLSCPO of model 2:",m2.nlscpo,"\n")
```

```
## NLSCPO of model 2: 10809.2
```

The NLSCPO increased compared to the first model, indicating worse predictive performance. This could be due to the fact that we have much more parameters than in the first model, which can lead to overfitting.

c)[10 marks] Propose an improved linear model using the information in the dataset before the game (you cannot use any information in the same row as the game, as this is only available after the game). Hint: you can try incorporating running averages of some covariates specific to each team, by doing some pre-processing.

Implement your model. Select your own prior distributions for the parameters, and discuss the reason for using those priors.

Obtain the summary statistics for the posterior distribution of the model parameters.

Evaluate the root mean square error (RMSE) of your posterior means versus the true scores.

Interpret the results.

We compute running averages of the home or away scores by each team that happened during the last 200 games among all teams (or less during the first 200 games in the season). We include this information about the home or away team in the dataframe that is passed to INLA, and include it in the formula with scaling. We use the same priors as in a) and b).

```
nteams=nrow(teams);
team_at_home_index=list()
team_at_away_index=list()
for(it in 1:nteams)
{
  team_at_home_index[[it]]=which(games$HOME_TEAM==teams$NICKNAME[it]);
  team_at_away_index[[it]]=which(games$AWAY_TEAM==teams$NICKNAME[it]);
}

games$AV_PTS_home=rep(110, G);
games$AV_PTS_away=rep(110, G);

for(i in 1:G)
{
  HT=which(teams$NICKNAME==games$HOME_TEAM[i]);
  AT=which(teams$NICKNAME==games$AWAY_TEAM[i]);
  if(sum(team_at_home_index[[HT]]<i)>0)
  {
    games$AV_PTS_home[i]=mean(
    games$PTS_home[team_at_home_index[[HT]][team_at_home_index[[HT]]<i
                   & team_at_home_index[[HT]]>(i-200)]]);
  }

  if(sum(team_at_away_index[[HT]]<i)>0)
  {
    games$AV_PTS_away[i]=mean(
    games$PTS_away[team_at_away_index[[HT]][team_at_away_index[[HT]]<i
                   & team_at_away_index[[HT]]>(i-200)]]);
  }
}

#AV_PTS=c(games$AV_PTS_home-games$AV_PTS_away, games$AV_PTS_away-games$AV_PTS_home);
AV_PTS=c(games$AV_PTS_home, games$AV_PTS_away);
#AV_PTS2=c(games$AV_PTS_away, games$AV_PTS_home);
data=data.frame(y,attack,defense,playing.at.home,AV_PTS)
```

```
m3=inla(formula=y~1+attack+defense+playing.at.home+scale(AV_PTS),
        data=data, family="gaussian",control.compute = list(dic = T,cpo=T,config=T),
        control.family=list(hyper=prec.prior),control.fixed=prior.beta)
summary(m3)
```

```
##
## Call:
##    c("inla.core(formula = formula, family = family, contrasts = contrasts,
##    ", " data = data, quantiles = quantiles, E = E, offset = offset, ", "
##    scale = scale, weights = weights, Ntrials = Ntrials, strata = strata,
##    ", " lp.scale = lp.scale, link.covariates = link.covariates, verbose =
##    verbose, ", " lincomb = lincomb, selection = selection, control.compute
##    = control.compute, ", " control.predictor = control.predictor,
##    control.family = control.family, ", " control.inla = control.inla,
##    control.fixed = control.fixed, ", " control.mode = control.mode,
##    control.expert = control.expert, ", " control.hazard = control.hazard,
##    control.lincomb = control.lincomb, ", " control.update =
##    control.update, control.lp.scale = control.lp.scale, ", "
##    control.pardiso = control.pardiso, only.hyperparam = only.hyperparam,
##    ", " inla.call = inla.call, inla.arg = inla.arg, num.threads =
##    num.threads, ", " blas.num.threads = blas.num.threads, keep = keep,
##    working.directory = working.directory, ", " silent = silent, inla.mode
##    = inla.mode, safe = FALSE, debug = debug, ", " .parent.frame =
##    .parent.frame)")
## Time used:
##     Pre = 4.43, Running = 2.87, Post = 0.764, Total = 8.07
## Fixed effects:
##                         mean    sd 0.025quant 0.5quant 0.975quant    mode kld
## (Intercept)          105.020 1.419    102.236  105.020    107.802 105.020   0
## attackBucks            4.738 1.527      1.744    4.737      7.733   4.737   0
## attackBulls            1.688 1.564     -1.378    1.688      4.755   1.687   0
## attackCavaliers       -1.663 1.573     -4.748   -1.663      1.423  -1.664   0
## attackCeltics          1.609 1.482     -1.296    1.609      4.515   1.608   0
## attackClippers        -0.906 1.582     -4.008   -0.906      2.198  -0.906   0
## attackGrizzlies        4.911 1.528      1.914    4.911      7.910   4.911   0
## attackHawks            3.870 1.551      0.829    3.870      6.913   3.870   0
## attackHeat             0.271 1.501     -2.672    0.271      3.216   0.271   0
## attackHornets          5.611 1.581      2.512    5.611      8.712   5.611   0
## attackJazz             2.990 1.566     -0.081    2.990      6.063   2.990   0
## attackKings            1.586 1.592     -1.535    1.586      4.709   1.585   0
## attackKnicks          -2.019 1.588     -5.132   -2.020      1.095  -2.020   0
## attackLakers           2.682 1.581     -0.419    2.682      5.784   2.681   0
## attackMagic           -4.189 1.591     -7.309   -4.190     -1.068  -4.190   0
## attackMavericks       -0.333 1.514     -3.302   -0.333      2.638  -0.333   0
## attackNets             4.439 1.561      1.377    4.438      7.501   4.438   0
## attackNuggets          3.259 1.563      0.194    3.259      6.325   3.258   0
## attackPacers           2.352 1.586     -0.758    2.351      5.463   2.351   0
## attackPelicans        -0.035 1.553     -3.081   -0.036      3.011  -0.036   0
## attackPistons         -3.406 1.589     -6.521   -3.406     -0.289  -3.406   0
## attackRaptors          0.364 1.555     -2.686    0.364      3.416   0.364   0
## attackRockets          0.665 1.589     -2.451    0.665      3.783   0.665   0
## attackSpurs            3.351 1.580      0.252    3.351      6.451   3.350   0
## attackSuns             4.359 1.535      1.349    4.359      7.370   4.358   0
## attackThunder         -4.625 1.599     -7.760   -4.626     -1.489  -4.626   0
```

```
## attackTimberwolves     6.144 1.559      3.087    6.144      9.202   6.143   0
## attackTrail Blazers   -3.360 1.591     -6.481   -3.361     -0.239  -3.361   0
## attackWarriors         2.342 1.495     -0.589    2.342      5.275   2.342   0
## attackWizards         -0.169 1.585     -3.277   -0.169      2.939  -0.170   0
## defenseBucks           3.229 1.527      0.234    3.229      6.224   3.229   0
## defenseBulls           4.334 1.563      1.269    4.334      7.399   4.334   0
## defenseCavaliers      -1.232 1.573     -4.318   -1.232      1.854  -1.232   0
## defenseCeltics        -3.559 1.481     -6.464   -3.559     -0.654  -3.559   0
## defenseClippers        1.369 1.582     -1.734    1.369      4.471   1.369   0
## defenseGrizzlies       1.293 1.527     -1.701    1.293      4.288   1.293   0
## defenseHawks           4.582 1.549      1.544    4.582      7.621   4.582   0
## defenseHeat           -2.423 1.501     -5.367   -2.423      0.522  -2.423   0
## defenseHornets         7.487 1.583      4.382    7.487     10.592   7.487   0
## defenseJazz            0.536 1.563     -2.531    0.535      3.602   0.535   0
## defenseKings           7.989 1.592      4.867    7.989     11.111   7.989   0
## defenseKnicks         -0.355 1.588     -3.470   -0.355      2.761  -0.355   0
## defenseLakers          7.833 1.581      4.732    7.833     10.933   7.833   0
## defenseMagic           4.993 1.592      1.871    4.993      8.116   4.993   0
## defenseMavericks      -3.187 1.514     -6.156   -3.187     -0.218  -3.187   0
## defenseNets            4.269 1.564      1.200    4.269      7.337   4.269   0
## defenseNuggets         3.688 1.563      0.622    3.688      6.754   3.688   0
## defensePacers          8.096 1.589      4.980    8.096     11.212   8.096   0
## defensePelicans        2.679 1.553     -0.368    2.679      5.725   2.679   0
## defensePistons         5.870 1.592      2.748    5.870      8.993   5.870   0
## defenseRaptors         0.526 1.555     -2.524    0.526      3.577   0.526   0
## defenseRockets        11.106 1.592      7.984   11.106     14.227  11.106   0
## defenseSpurs           5.292 1.579      2.195    5.292      8.390   5.292   0
## defenseSuns           -0.061 1.534     -3.069   -0.061      2.947  -0.061   0
## defenseThunder         4.829 1.597      1.697    4.829      7.961   4.829   0
## defenseTimberwolves    5.003 1.562      1.939    5.003      8.066   5.002   0
## defenseTrail Blazers   8.129 1.597      4.997    8.129     11.261   8.129   0
## defenseWarriors       -1.745 1.494     -4.674   -1.745      1.184  -1.745   0
## defenseWizards         5.554 1.587      2.442    5.554      8.666   5.554   0
## playing.at.home        1.735 0.446      0.861    1.735      2.609   1.735   0
## scale(AV_PTS)          1.364 0.241      0.890    1.364      1.837   1.364   0
##
## Model hyperparameters:
##                                        mean   sd 0.025quant 0.5quant
## Precision for the Gaussian observations 0.007 0.00      0.007    0.007
##                                        0.975quant  mode
## Precision for the Gaussian observations      0.008 0.007
##
## Deviance Information Criterion (DIC) ...............: 21583.64
## Deviance Information Criterion (DIC, saturated) ....: 2840.14
## Effective number of parameters .....................: 60.68
##
## Marginal log-Likelihood:  -10988.00
## CPO, PIT is computed
## Posterior summaries for the linear predictor and the fitted values are computed
## (Posterior marginals needs also 'control.compute=list(return.marginals.predictor=TRUE)')
```

The posterior mean for the scale(AV_PTS) is 1.36, indicating that teams that have a relatively good recent performance tend to score higher in the current game too.

```
rmse(y, m3$summary.fitted.values$mean)
```

```
## [1] 11.51747
```

The RMSE is lower than for model a), showing that including this new covariate in the model is useful.

This was not required, but we computed the NLSCPO as well.

```
m3.nlscpo=-sum(log(m3$cpo$cpo))
cat("NLSCPO of model 3:",m3.nlscpo,"\n")
```

```
## NLSCPO of model 3: 10792.27
```

The NLSCPO is better than for models in questions a) and b), indicating that this new covariate helps with improving the predictive performance.

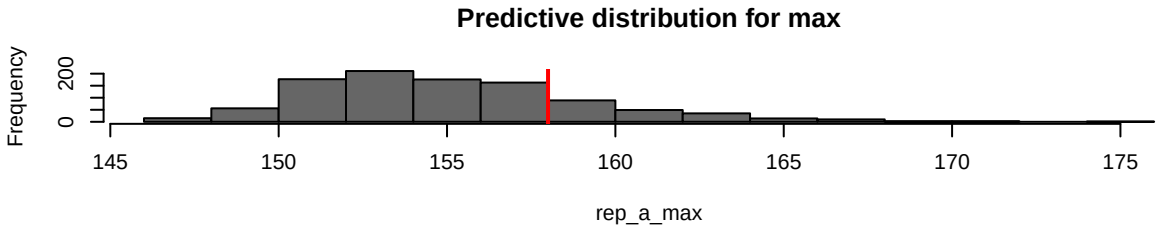
d)[10 marks] Perform posterior predictive checks on all 3 models a), b), and c). Explain how did you choose the test functions.

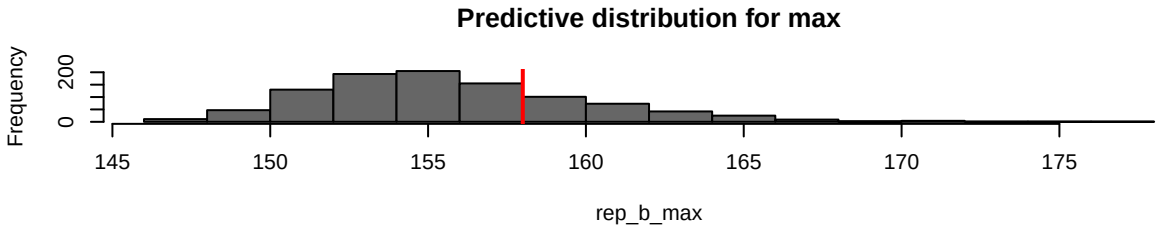
Discuss the results.

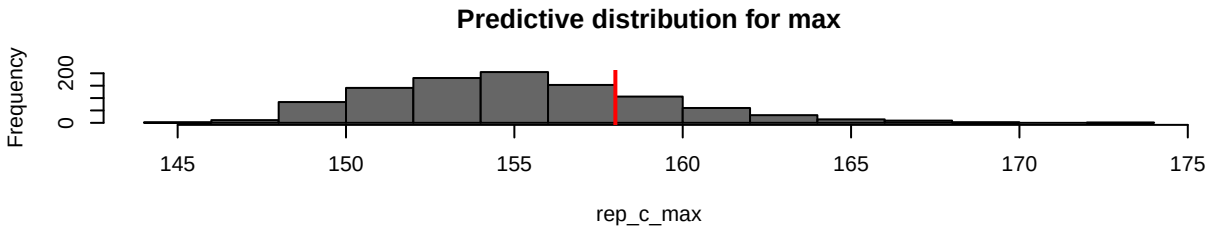
We obtain the samples from the linear predictors of the 3 using inla.posterior.sample. To get the replicate samples, we also add the noise terms, as it was done in Lecture 2 and Workshop 2. As test functions, we choose the maximum amongst all scores, the minimum amongst all scores, and the mean absolute differences between home and away team scores in each game.

```
nbsamp=1000;
samp<-inla.posterior.sample(nbsamp, m1);
predictor.samples=inla.posterior.sample.eval(function(...) {Predictor},
samp)
sigma.samples=1/sqrt(inla.posterior.sample.eval(function(...) {theta},
  samp))
rep_a=matrix(0,nrow=2*G,ncol=nbsamp);
for(it in 1:(2*G)){
rep_a[it,]=predictor.samples[it,]+rnorm(nbsamp, mean=0,sd=sigma.samples)
}

samp<-inla.posterior.sample(nbsamp, m2);
predictor.samples=inla.posterior.sample.eval(function(...) {Predictor},
samp)
sigma.samples=1/sqrt(inla.posterior.sample.eval(function(...) {theta},
  samp))
rep_b=matrix(0,nrow=2*G,ncol=nbsamp);
for(it in 1:(2*G)){
rep_b[it,]=predictor.samples[it,]+rnorm(nbsamp, mean=0,sd=sigma.samples)
}

samp<-inla.posterior.sample(nbsamp, m3);
predictor.samples=inla.posterior.sample.eval(function(...) {Predictor},
samp)
sigma.samples=1/sqrt(inla.posterior.sample.eval(function(...) {theta},
  samp))
rep_c=matrix(0,nrow=2*G,ncol=nbsamp);
for(it in 1:(2*G)){
rep_c[it,]=predictor.samples[it,]+rnorm(nbsamp, mean=0,sd=sigma.samples)
}
```

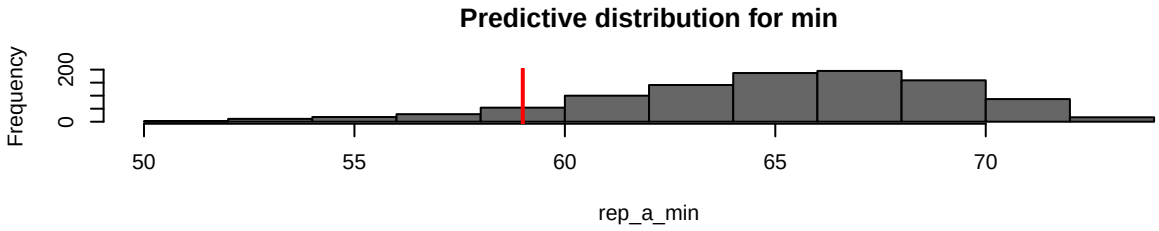
```
rep_a_max=apply(rep_a,2,max)
rep_b_max=apply(rep_b,2,max)
rep_c_max=apply(rep_c,2,max)

par(mfrow=c(3,1))
hist(rep_a_max,col="gray40",main="Predictive distribution for max", breaks=15)
abline(v=max(y),col="red",lwd=2)

hist(rep_b_max,col="gray40",main="Predictive distribution for max", breaks=15)
abline(v=max(y),col="red",lwd=2)

hist(rep_c_max,col="gray40",main="Predictive distribution for max", breaks=15)
abline(v=max(y),col="red",lwd=2)
```

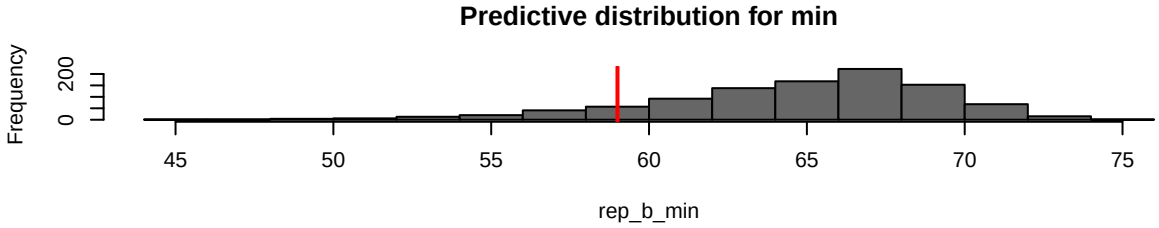


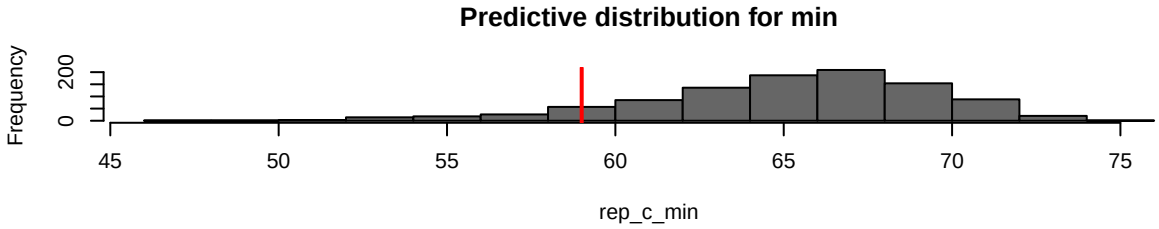




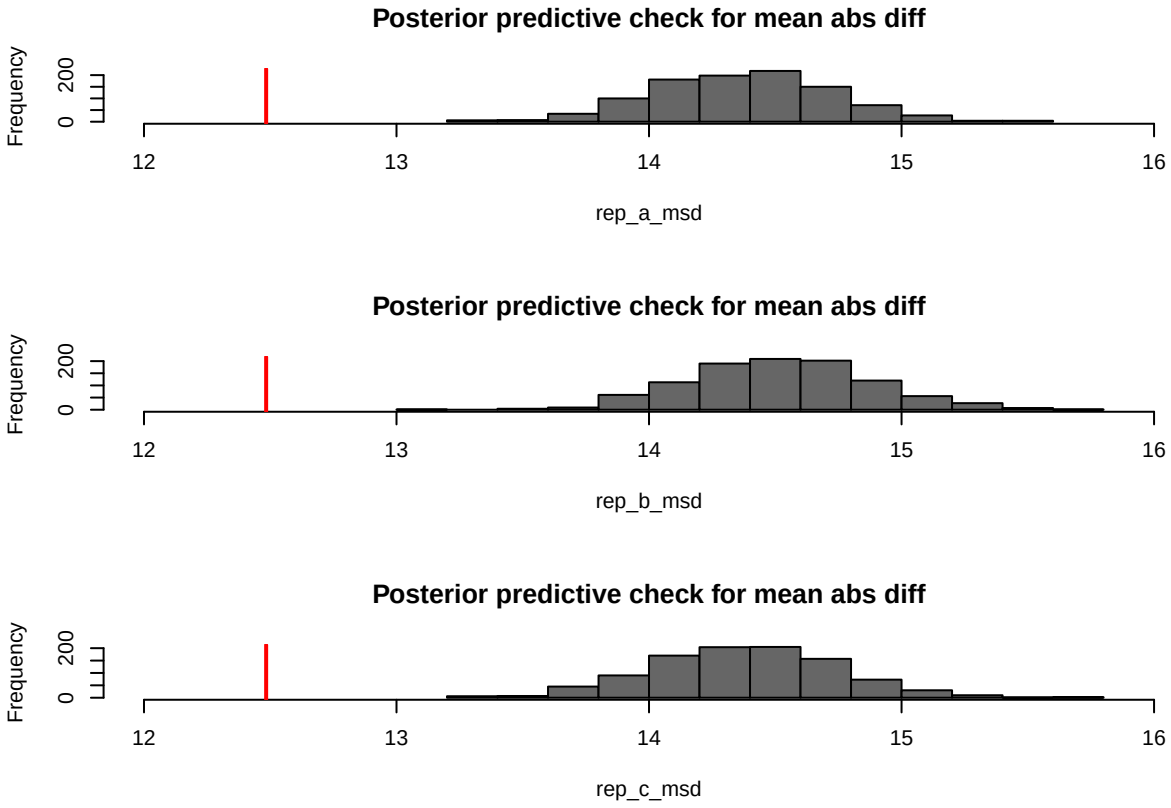

```
rep_a_min=apply(rep_a,2,min)
rep_b_min=apply(rep_b,2,min)
rep_c_min=apply(rep_c,2,min)

par(mfrow=c(3,1))
hist(rep_a_min,col="gray40",main="Predictive distribution for min", breaks=15)
abline(v=min(y),col="red",lwd=2)

hist(rep_b_min,col="gray40",main="Predictive distribution for min", breaks=15)
abline(v=min(y),col="red",lwd=2)

hist(rep_c_min,col="gray40",main="Predictive distribution for min", breaks=15)
abline(v=min(y),col="red",lwd=2)
```








```r
mean_abs_diff<-function(v){l=length(v)/2; return(mean(abs(v[1:l]-v[(l+1):(2*l)]))); }

rep_a_msd=apply(rep_a,2,mean_abs_diff)
rep_b_msd=apply(rep_b,2,mean_abs_diff)
rep_c_msd=apply(rep_c,2,mean_abs_diff)

par(mfrow=c(3,1))
hist(rep_a_msd,col="gray40",main="Posterior predictive check for mean abs diff", breaks=15,xlim=c(12,16)
abline(v=mean_abs_diff(y),col="red",lwd=2)

hist(rep_b_msd,col="gray40",main="Posterior predictive check for mean abs diff", breaks=15,xlim=c(12,16)
abline(v=mean_abs_diff(y),col="red",lwd=2)

hist(rep_c_msd,col="gray40",main="Posterior predictive check for mean abs diff", breaks=15,xlim=c(12,16)
abline(v=mean_abs_diff(y),col="red",lwd=2)
```



The first two test functions (min and max) do not seem to detect any issues with the model fit. However, the third test function (mean absolute difference between home/away team scores) seems to indicate that our model tends to have a larger score difference than in reality. This could be due to the independent noise assumption between home/away team scores, which might not be a good model of reality given the competitive nature of basketball.

e)[10 marks] In the previous questions, we were assuming a model of the form.

$$S_g^H \sim N(\mu_g^H, \sigma^2), \quad S_g^A \sim N(\mu_g^A, \sigma^2).$$

It is natural to model these two results jointly with a multivariate normal,

$$(S_g^H, S_g^A) \sim N\left(\begin{pmatrix}\mu_g^H \\ \mu_g^A\end{pmatrix}, \Sigma\right),$$

where Σ is a 2 times 2 covariance matrix.

Implement such a model. The definition of μ_g^H and μ_g^A can be either one of a), b), or c), you just need to implement one of them.

Explain how did you choose the prior on Σ [Hint: you can use a Wishart prior, or express this a product of diagonal and correlation matrices and put priors on those terms].

Obtain the summary statistics for the posterior distribution of the model parameters.

Evaluate the root mean square error (RMSE) of your posterior means versus the true scores.

Interpret the results.

We implement this model in INLA using the iid2d random effect, see https://inla.r-inla-download.org/r-inla.org/doc/latent/iid.pdf. This random effect allows for having a correlations between pairs of random

variables, i.e. there are 2*G random variables in total in G pairs, and these G pairs are independent, identically distributed (i.i.d.).

As explained in the INLA documentation, this model has to be parameterised in terms of indices $1:2*G$, where the pairs of variables are $(1, G+1)$, $(2, G+2)$, etc. This is exactly in the right order in our dataset, since the scores of the home team is in the first G rows, and the score of the away teams are in the last G rows. Note that the Gaussian likelihood also has an independent noise term, which is not needed as we model the noise via the random effect. To take this into account, the precision of the Gaussian likelihood is set to a very large value (10^6) via the term control.family = list(hyper = list(prec=list(initial=log(1e6), fixed=TRUE))). This means that the standard deviation is 0.001, so this term is negligibly small, and the noise will be instead fitted by the random effect terms.

For the regression coefficients, we use the same priors as before in part a). For the random effect, we use the default Wishart 2d prior, with parameters (4,1,1,0).

```
id=1:(2*G);
data=data.frame(y,attack,defense,playing.at.home,AV_PTS,id);

m4=inla(formula=y~1+attack+defense+playing.at.home+scale(AV_PTS)
        +f(id, model="iid2d",n=(2*G)), data=data, family="gaussian",
        control.family = list(hyper = list(prec=list(initial=log(1e6), fixed=TRUE))),
        control.compute=list(cpo=T, dic=T,config=T), control.fixed=prior.beta)
summary(m4)
```

```
##
## Call:
##    c("inla.core(formula = formula, family = family, contrasts = contrasts,
##    ", " data = data, quantiles = quantiles, E = E, offset = offset, ", "
##    scale = scale, weights = weights, Ntrials = Ntrials, strata = strata,
##    ", " lp.scale = lp.scale, link.covariates = link.covariates, verbose =
##    verbose, ", " lincomb = lincomb, selection = selection, control.compute
##    = control.compute, ", " control.predictor = control.predictor,
##    control.family = control.family, ", " control.inla = control.inla,
##    control.fixed = control.fixed, ", " control.mode = control.mode,
##    control.expert = control.expert, ", " control.hazard = control.hazard,
##    control.lincomb = control.lincomb, ", " control.update =
##    control.update, control.lp.scale = control.lp.scale, ", "
##    control.pardiso = control.pardiso, only.hyperparam = only.hyperparam,
##    ", " inla.call = inla.call, inla.arg = inla.arg, num.threads =
##    num.threads, ", " blas.num.threads = blas.num.threads, keep = keep,
##    working.directory = working.directory, ", " silent = silent, inla.mode
##    = inla.mode, safe = FALSE, debug = debug, ", " .parent.frame =
##    .parent.frame)")
## Time used:
##     Pre = 5.13, Running = 4.18, Post = 0.904, Total = 10.2
## Fixed effects:
##                          mean    sd 0.025quant 0.5quant 0.975quant    mode kld
## (Intercept)           104.839 1.530    101.841  104.839    107.837 104.839   0
## attackBucks             4.755 1.527      1.762    4.755      7.747   4.755   0
## attackBulls             1.778 1.565     -1.289    1.778      4.846   1.778   0
## attackCavaliers        -1.678 1.575     -4.764   -1.678      1.409  -1.678   0
## attackCeltics           1.567 1.482     -1.338    1.567      4.472   1.567   0
## attackClippers         -0.921 1.580     -4.019   -0.921      2.176  -0.921   0
## attackGrizzlies         4.987 1.527      1.994    4.987      7.979   4.987   0
## attackHawks             3.968 1.551      0.927    3.968      7.008   3.968   0
## attackHeat              0.322 1.508     -2.633    0.322      3.277   0.322   0
```

```
## attackHornets          5.527 1.582      2.426    5.527      8.628   5.527   0
## attackJazz             3.131 1.565      0.064    3.131      6.198   3.131   0
## attackKings            1.537 1.590     -1.580    1.537      4.653   1.537   0
## attackKnicks          -2.073 1.589     -5.186   -2.073      1.041  -2.073   0
## attackLakers           2.622 1.580     -0.474    2.622      5.719   2.622   0
## attackMagic           -4.272 1.593     -7.393   -4.272     -1.150  -4.272   0
## attackMavericks       -0.345 1.512     -3.309   -0.345      2.619  -0.345   0
## attackNets             4.331 1.563      1.267    4.331      7.395   4.331   0
## attackNuggets          3.280 1.562      0.218    3.280      6.341   3.280   0
## attackPacers           2.390 1.586     -0.719    2.390      5.499   2.390   0
## attackPelicans         0.007 1.552     -3.034    0.007      3.048   0.007   0
## attackPistons         -3.433 1.591     -6.552   -3.433     -0.314  -3.433   0
## attackRaptors          0.425 1.566     -2.644    0.425      3.494   0.425   0
## attackRockets          0.663 1.588     -2.449    0.663      3.776   0.663   0
## attackSpurs            3.359 1.579      0.264    3.359      6.453   3.359   0
## attackSuns             4.419 1.533      1.414    4.419      7.423   4.419   0
## attackThunder         -4.704 1.597     -7.834   -4.704     -1.574  -4.704   0
## attackTimberwolves     6.105 1.557      3.053    6.105      9.158   6.105   0
## attackTrail Blazers   -3.319 1.590     -6.435   -3.319     -0.203  -3.319   0
## attackWarriors         2.453 1.493     -0.473    2.453      5.378   2.453   0
## attackWizards         -0.123 1.585     -3.229   -0.123      2.983  -0.123   0
## defenseBucks           3.414 1.527      0.422    3.414      6.407   3.414   0
## defenseBulls           4.462 1.564      1.397    4.462      7.528   4.462   0
## defenseCavaliers      -1.074 1.575     -4.160   -1.074      2.012  -1.074   0
## defenseCeltics        -3.379 1.482     -6.283   -3.379     -0.475  -3.379   0
## defenseClippers        1.515 1.581     -1.583    1.515      4.613   1.515   0
## defenseGrizzlies       1.521 1.525     -1.468    1.521      4.510   1.521   0
## defenseHawks           4.745 1.549      1.709    4.745      7.781   4.745   0
## defenseHeat           -2.263 1.509     -5.220   -2.263      0.694  -2.263   0
## defenseHornets         7.725 1.584      4.620    7.725     10.830   7.725   0
## defenseJazz            0.686 1.562     -2.375    0.686      3.747   0.686   0
## defenseKings           8.140 1.591      5.022    8.140     11.257   8.140   0
## defenseKnicks         -0.285 1.590     -3.402   -0.285      2.831  -0.285   0
## defenseLakers          8.019 1.579      4.924    8.019     11.115   8.019   0
## defenseMagic           5.063 1.593      1.940    5.063      8.186   5.063   0
## defenseMavericks      -3.000 1.512     -5.962   -3.000     -0.037  -3.000   0
## defenseNets            4.450 1.567      1.379    4.450      7.521   4.450   0
## defenseNuggets         3.761 1.561      0.701    3.761      6.821   3.761   0
## defensePacers          8.200 1.589      5.085    8.200     11.314   8.200   0
## defensePelicans        2.830 1.552     -0.211    2.830      5.872   2.830   0
## defensePistons         6.014 1.595      2.888    6.014      9.139   6.014   0
## defenseRaptors         0.679 1.566     -2.390    0.679      3.748   0.679   0
## defenseRockets        11.189 1.590      8.072   11.189     14.306  11.189   0
## defenseSpurs           5.411 1.578      2.318    5.411      8.504   5.411   0
## defenseSuns            0.135 1.532     -2.868    0.135      3.138   0.135   0
## defenseThunder         4.937 1.595      1.810    4.937      8.064   4.937   0
## defenseTimberwolves    5.307 1.560      2.250    5.307      8.364   5.307   0
## defenseTrail Blazers   8.248 1.595      5.121    8.248     11.374   8.248   0
## defenseWarriors       -1.523 1.492     -4.447   -1.523      1.401  -1.523   0
## defenseWizards         5.658 1.587      2.547    5.658      8.768   5.658   0
## playing.at.home        1.747 0.384      0.993    1.747      2.500   1.747   0
## scale(AV_PTS)          1.225 0.242      0.751    1.225      1.699   1.225   0
##
## Random effects:
```

```
##   Name     Model
##     id IID2D model
##
## Model hyperparameters:
##                                  mean    sd 0.025quant 0.5quant 0.975quant  mode
## Precision for id (component 1) 0.008 0.000      0.007    0.008      0.008 0.008
## Precision for id (component 2) 0.007 0.000      0.007    0.007      0.008 0.007
## Rho1:2 for id                  0.265 0.016      0.232    0.265      0.297 0.265
##
## Deviance Information Criterion (DIC) ...............: -27717.87
## Deviance Information Criterion (DIC, saturated) ....: 5556.00
## Effective number of parameters .....................: 2778.00
##
## Marginal log-Likelihood:  -10977.47
## CPO, PIT is computed
## Posterior summaries for the linear predictor and the fitted values are computed
## (Posterior marginals needs also 'control.compute=list(return.marginals.predictor=TRUE)')
```

The results show that the correlation between the two components (rho1:2) has posterior mean 0.237, indicating that indeed there is a positive correlation between the noise terms between home/away score. This seems reasonable given the competitive nature of the game.

Now we are going to compute the RMSE. Note that we cannot use the previous method for this based on the fitted values,

```
rmse(y, m4$summary.fitted.values$mean)
```

```
## [1] 8.803469e-08
```

This is essentially zero, because the fitted values already include the random effect terms, which we use for modelling the noise.

The way for evaluating the predicted values for the scores from our model is using the model matrix together with the posterior mean of the fixed effect regression coefficients.

```
predicted.y=as.numeric(m4$model.matrix%*%m4$summary.fixed$mean);
rmse(y,predicted.y)
```

```
## [1] 11.51853
```

As we can see, the RMSE is similar to our previous models.

This was not required, but we also evaluated the NLSCPO.

```
m4.nlscpo=-sum(log(m4$cpo$cpo))
cat("NLSCPO of model 4:",m4.nlscpo,"\n")
```

```
## NLSCPO of model 4: -11692.87
```

There is a very significant improvement over earlier models, indicating that this model will likely have better predictive performance.