University of Edinburgh

School of Mathematics

Bayesian Data Analysis, 2023/2024, Semester 2

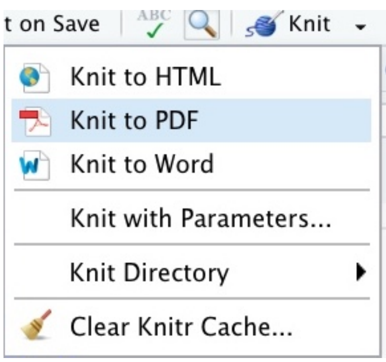
Assignment 1

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 1 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 (this can be in a zip file if you include additional images). 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.



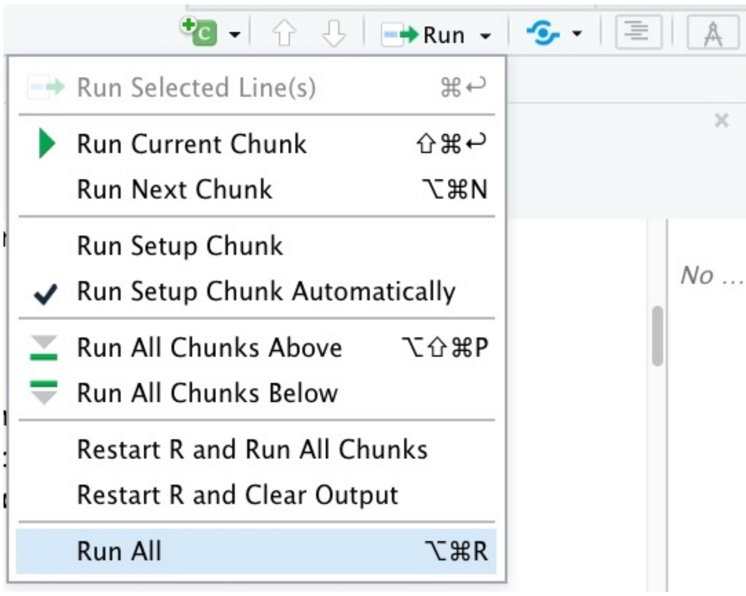
The compiled PDF needs to contain everything in this notebook, with your code sections clearly visible (not hidden), and the output of your code included. Reports without the code displayed in the PDF, or without the output of your code included in the PDF will be marked as 0, with the only feedback "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 part of the assignments that do not run, 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. He will get one reminder to submit his code. If he does not do it, 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."

3) Group work: This is an INDIVIDUAL ASSIGNMENT. You can talk to your classmates to clarify questions, but you have to do your work individually and cannot copy parts from other students. Students who submit work that has not been done individually will be reported for Academic Misconduct, which can lead to severe consequences. Each question will be marked by a single instructor, and submissions will be compared by advanced software tools, so we will be able to spot students who copy.

4) Piazza: During 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 one office hour per week (Wednesdays 16:00-17:00) during the 2 weeks for this assignment. This is in JCMB 5413. 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.

6) Late submissions and extensions: UP TO A MAXIMUM OF 3 CALENDAR DAYS EXTENSION IS ALLOWED FOR THIS ASSIGNMENT IN THE ESC SYSTEM. You need to apply before the deadline.

If you submit your solutions on Learn before the deadline, the system will not allow you to update it even if you have received an extension. There is only 1 submission allowed after the deadline.

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.

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

Ping-pong, or table tennis, is a popular sport around the world. In this assignment, we will apply Bayesian modelling to the movement of a ping-pong ball.

As explained in the paper "Optimal State Estimation of Spinning Ping-Pong Ball Using Continuous Motion Model" by Zhao et al., the physical equations describing the movement of a spinning ball in air without wind can be described as

$$\frac{dV}{dt} = k_d \|V\| V + k_m \omega \times V + g,$$

where V is the velocity of the ball (3-dimensional vector), ω is the angular velocity (3-dimensional vector), g is the local gravity acceleration (3-dimensional vector), $\times$ refers to the cross product [https://en.wikipedia.org/wiki/Cross_product]. The constants k_d and k_m are expressed as

$$k_d := -\frac{1}{2m} C_D \rho A$$
$$k_m := \frac{1}{2m} C_m \rho A r.$$

Figure 1: Ma Long. Made in China.

The meaning and values of the parameters here are shown in the following table.

TABLE I

VALUES OF THE CONSTANT PARAMETERS

Symbol	Quantity	Value
m	mass	$0.0027kg$
g	acceleration of gravity	$9.827\, m/s^2$
r	radius	$0.02\, m$
ρ	air density	$1.29kg/m^3$
A	cross sectional area	$0.001256m^2$
C_D	coefficient of air resistance	0.456
C_m	coefficient of Magnus	1.0

We observe positions and velocities at times $T_1, T_2, \ldots, T_n$, and define $\Delta_k = T_{k+1} - T_k$. The simplest way to discretize this ODE is as follows(Euler-Mayurama discretization of the original ODE, see equation (2) of "Optimal State Estimation of Spinning Ping-Pong Ball Using Continuous Motion Model"),

$$\begin{bmatrix} x(k+1) \\ y(k+1) \\ z(k+1) \\ v_x(k+1) \\ v_y(k+1) \\ v_z(k+1) \end{bmatrix} = \begin{bmatrix} x(k) \\ y(k) \\ z(k) \\ v_x(k) \\ v_y(k) \\ v_z(k) \end{bmatrix} + \begin{bmatrix} v_x(k) \\ v_y(k) \\ v_z(k) \\ k_d\|V(k)\|v_x(k) + k_m(\omega_y v_z(k) - \omega_z v_y(k)) \\ k_d\|V(k)\|v_y(k) + k_m(\omega_z v_x(k) - \omega_x v_z(k)) \\ k_d\|V(k)\|v_z(k) + k_m(\omega_x v_y(k) - \omega_y v_x(k)) - g \end{bmatrix} \cdot \Delta_k,$$

where $\|V(k)\| = \sqrt{v_x(k)^2 + v_y(k)^2 + v_z(k)^2}$ is the magnitude of velocity at time T_k.

In this question, we are going to assume a similar model, but with additional Gaussian model noise, and also assume that the position and velocity observations are also subject to observation

noise. Hence, our model equations are as follows,

$$\begin{bmatrix} x(k+1) \\ y(k+1) \\ z(k+1) \\ v_x(k+1) \\ v_y(k+1) \\ v_z(k+1) \end{bmatrix} \sim N \left[\begin{bmatrix} x(k) \\ y(k) \\ z(k) \\ v_x(k) \\ v_y(k) \\ v_z(k) \end{bmatrix} + \begin{bmatrix} v_x(k) \\ v_y(k) \\ v_z(k) \\ k_d \|V(k)\| v_x(k) + k_m(\omega_y v_z(k) - \omega_z v_y(k)) \\ k_d \|V(k)\| v_y(k) + k_m(\omega_z v_x(k) - \omega_x v_z(k)) \\ k_d \|V(k)\| v_z(k) + k_m(\omega_x v_y(k) - \omega_y v_x(k)) - g \end{bmatrix} \cdot \Delta_k, \Sigma \right],$$

where $\Sigma = Diag[\tau_{pos}^{-1}, \tau_{pos}^{-1}, \tau_{pos}^{-1}, \tau_{vel}^{-1}, \tau_{vel}^{-1}, \tau_{vel}^{-1}]$ **is a diagonal covariance matrix depending on parameters** τ_{pos} **and** τ_{vel}**.**

The observation model is the following,

$$\begin{bmatrix} x^o(k) \\ y^o(k) \\ z^o(k) \\ v_x^o(k) \\ v_y^o(k) \\ v_z^o(k) \end{bmatrix} \sim N \left[\begin{bmatrix} x(k) \\ y(k) \\ z(k) \\ v_x(k) \\ v_y(k) \\ v_z(k) \end{bmatrix}, \Sigma^o \right],$$

where $\Sigma^o = Diag[(\tau_{pos}^o)^{-1}, (\tau_{pos}^o)^{-1}, (\tau_{pos}^o)^{-1}, (\tau_{vel}^o)^{-1}, (\tau_{vel}^o)^{-1}, (\tau_{vel}^o)^{-1}]$

Q1) [10 marks]

a) [5 marks] Draw a DAG representation of this model (this can be done on a tablet, or draw it on a piece of paper and then take a picture or scan). Images can be included in R Markdown, as explained in [https://www.earthdatascience.org/courses/earth-analytics/document-your-science/add-images-to-rmarkdown-report/].

b) [5 marks] For the initial values, choose priors of the form$x(1) \sim N(0,1), y(1) \sim N(0,1), z(1) \sim N(0,1)$, $v_x(1) \sim N(0,25), v_y(1) \sim N(0,25), v_z(1) \sim N(0,25)$.**Choose your own priors for** τ_{pos}**,** τ_{vel}**,** τ_{pos}^o**,** τ_{vel}^o **,** ω_x**,** ω_y**,** ω_z**. Explain your choices.**

If you use informative priors, please cite the source of the information you used precisely (i.e. web link, or precise page number in a paper. Saying Google search said " " will not suffice).

- ω_x, ω_y, ω_z: For angular velocities of the three dimensions, since we have little information about the launching details and the angular velocity and direction of the ball, we choose uninformative normal distributions for modelling, which is a proper consideration under this randomized situation, and set the means as 0. In addition, since the measured ball spins among players on national team level which used maximum strength to hit a defined target is $113.9s^{-1}$ [1], we set the standard deviations as reasonably high values, 100, i.e. $\omega_x \sim N(0, 100^2)$, $\omega_y \sim N(0, 100^2)$, and $\omega_z \sim N(0, 100^2)$, to ensure the dataset can have a greater impact on the estimation than the priors.

- τ_{pos}, τ_{vel}, τ_{pos}^o, τ_{vel}^o: we set gamma priors to the precisions. However, given that the maximum speed of AIMY's precise launch is $15.4m \cdot s^{-1}$ [1], and the commonsense that the outside environmental force will not have a considerable impact on the velocity relatively, we believe that under the experimental settings, the velocity fluctuation should generally not drop out of ± 5 interval. Therefore, we choose 1 and 0.1 for the shape and scale parameters, respectively (making the mean precision $\mu_\tau = \alpha\beta = 0.1$, in a reasonable scale and mass not too concentrated), i.e. $\tau_{pos} \sim \gamma(1, 0.1)$, $\tau_{vel} \sim \gamma(1, 0.1)$, $\tau_{pos}^o \sim \gamma(1, 0.1)$, $\tau_{vel}^o \sim \gamma(1, 0.1)$.

References

[1] Dittrich, A., Schneider, J., Guist, S., Gurtler, N., Ott, H., Steinbrenner, T., ... & Buchler, D. (2023, May). AIMY: An open-source table tennis ball launcher for versatile and high-fidelity trajectory generation. In 2023 IEEE International Conference on Robotics and Automation (ICRA) (pp. 3058-3064). IEEE. (paragraph 4, page 1)

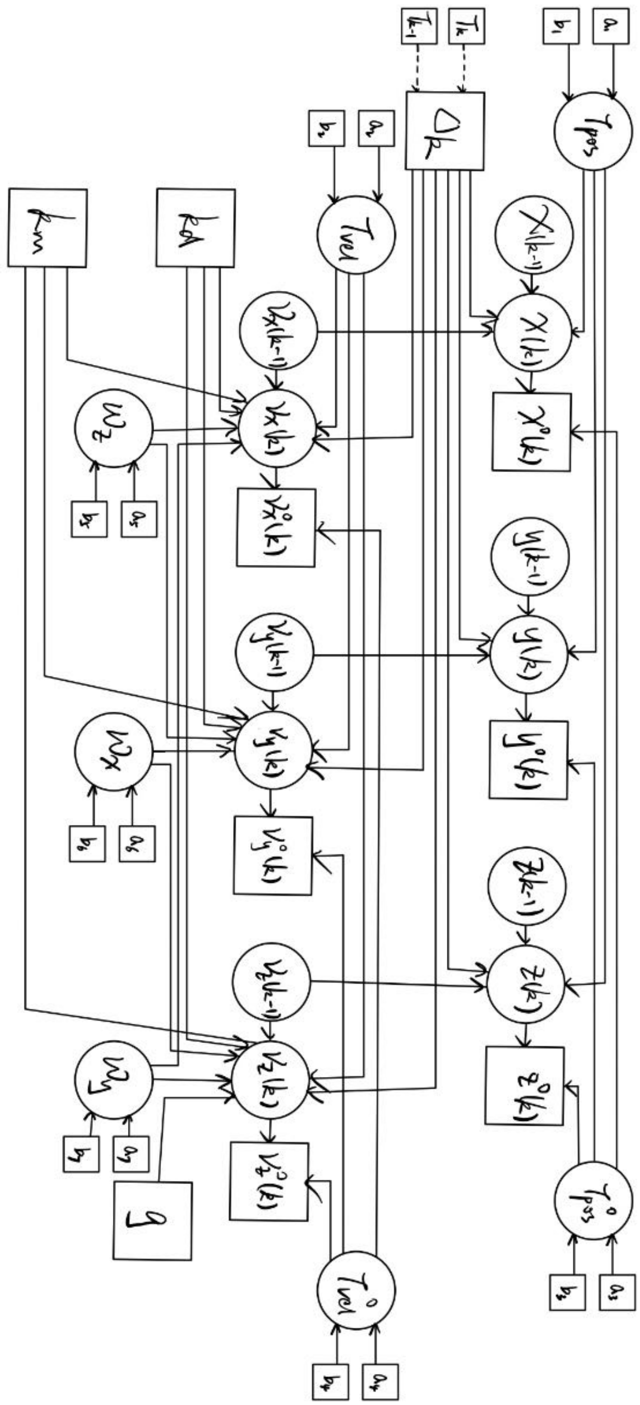


Figure 2: A DAG representation of the above model.

Q2) [10 marks] In this question, we are going to fit the model of Q1) on a real dataset from [Table Tennis Ball Trajectories with Spin - Edmond (mpg.de)]. In this dataset, there are many recorded trajectories of ping-pong balls shot out by a table tennis launcher robot. We will only use one trajectory here.

First, we load the dataset, and show a 3D plot of the trajectory.

```r
# If you do not have BiocManager and rhdf5 packages installed, you need to install these first.
# install.packages("BiocManager")
# BiocManager::install("rhdf5")
library(rhdf5)
```

```
## Warning: package 'rhdf5' was built under R version 4.3.2
```

```r
# This command lists all the information in this dataset.
# Please do not include it in the knitted PDF, as it takes 20+ pages
# h5ls("MN5008_grid_data_equal_speeds.hdf5",)
n <- 60;

xyz.obs <- h5read("MN5008_grid_data_equal_speeds.hdf5", "/originals/405/positions")[, 2: (n + 1)];
# Read positions of simulation number 405
xo <- xyz.obs[1, ];
yo <- xyz.obs[2, ];
zo <- xyz.obs[3, ];

vxvyvz.obs <- h5read("MN5008_grid_data_equal_speeds.hdf5", "/originals/405/velocities")[, 2: (n + 1)];
#Read velocities of simulation number 405
vxo <- vxvyvz.obs[1, ];
vyo <- vxvyvz.obs[2, ];
vzo <- vxvyvz.obs[3, ];

T <- h5read("MN5008_grid_data_equal_speeds.hdf5", "/originals/405/time_stamps")[2: (n + 1)];
#Read time points of observations

library(rgl)
```

```
## Warning: package 'rgl' was built under R version 4.3.2
```

```r
rgl_init <- function(new.device=FALSE, bg="white", width=640) {
  if(new.device | rgl.cur() == 0) {
    open3d()
    par3d(windowRect=50 + c(0, 0, width, width))
    bg3d(color=bg)
  }
  clear3d(type=c("shapes", "bboxdeco"))
  view3d(theta=15, phi=20, zoom=0.7)
}

rgl_init()
spheres3d(xo, yo, zo, r=0.05, color="yellow")  # Scatter plot
bbox3d(color="#333377")
```

Implement the model explained in Q1) in JAGS or STAN, with the data here referring to the observations $x^o, y^o, z^o, v_x^o, v_y^o, v_z^o$.

Please treat k_m and k_d as fixed constants that can be computed based on the equations in Q1).

Evaluate the Gelman-Rubin diagnostics for model parameters τ_{pos}, τ_{vel}, τ_{pos}^o, τ_{vel}^o , ω_x, ω_y, ω_z, as

well as their effective sample sizes. Choose the burn-in period, number of chains, and number of iterations such that the effective sample size is at least 1000 in all of these parameters.

Include the summary statistics from these parameters. Discuss the results.

Plot the posterior density of the angular velocity parameters (ω_x, ω_y, ω_z). Discuss the results.

```
library(rjags)
```

```
## Warning: package 'rjags' was built under R version 4.3.2

## Loading required package: coda

## Warning: package 'coda' was built under R version 4.3.2

## Linked to JAGS 4.3.1

## Loaded modules: basemod,bugs
```

```
# jags model string
model_string <- "
model {
  tau.p ~ dgamma(a.tau, b.tau)
  tau.po ~ dgamma(a.tau, b.tau)
  tau.v ~ dgamma(a.tau, b.tau)
  tau.vo ~ dgamma(a.tau, b.tau)

  x[1] ~ dnorm(mup1, taup1)
  y[1] ~ dnorm(mup1, taup1)
  z[1] ~ dnorm(mup1, taup1)
  vx[1] ~ dnorm(muv1, tauv1)
  vy[1] ~ dnorm(muv1, tauv1)
  vz[1] ~ dnorm(muv1, tauv1)

  wx ~ dnorm(muw, tauw)
  wy ~ dnorm(muw, tauw)
  wz ~ dnorm(muw, tauw)

  for (i in 1: (n - 1)) {
    delta[i] = t[i + 1] - t[i]
    V[i] = sqrt(vx[i] ** 2 + vy[i] ** 2 + vz[i] ** 2)
    x[i + 1] ~ dnorm(x[i] + delta[i] * vx[i], tau.p)
    y[i + 1] ~ dnorm(y[i] + delta[i] * vy[i], tau.p)
    z[i + 1] ~ dnorm(z[i] + delta[i] * vz[i], tau.p)
    vx[i + 1] ~ dnorm(vx[i] +
                    delta[i] * (kd * V[i] * vx[i] + km * (wy * vz[i] - wz * vy[i])), tau.v)
    vy[i + 1] ~ dnorm(vy[i] +
                    delta[i] * (kd * V[i] * vy[i] + km * (wz * vx[i] - wx * vz[i])), tau.v)
    vz[i + 1] ~ dnorm(vz[i] +
                    delta[i] * (kd * V[i] * vz[i] + km * (wx * vy[i] - wy * vx[i]) - g), tau.v)
  }

  for (i in 1: n) {
    x.o[i] ~ dnorm(x[i], tau.po)
    y.o[i] ~ dnorm(y[i], tau.po)
    z.o[i] ~ dnorm(z[i], tau.po)
    vx.o[i] ~ dnorm(vx[i], tau.vo)
    vy.o[i] ~ dnorm(vy[i], tau.vo)
```

```
    vz.o[i] ~ dnorm(vz[i], tau.vo)

    xrep.o[i] ~ dnorm(x[i], tau.po)
    yrep.o[i] ~ dnorm(y[i], tau.po)
    zrep.o[i] ~ dnorm(z[i], tau.po)
    vxrep.o[i] ~ dnorm(vx[i], tau.vo)
    vyrep.o[i] ~ dnorm(vy[i], tau.vo)
    vzrep.o[i] ~ dnorm(vz[i], tau.vo)
  }

}
"

# defining constants, hyperparameters, and model running parameters
m <- 0.0027; g <- 9.827; r <- 0.02; rho <- 1.29
A <- 0.001256; CD <- 0.456; Cm <- 1.0
kd <- - CD * rho * A / (2 * m)
km <- Cm * rho * A * r / (2 * m)

mup1 <- 0; taup1 <- 1; muv1 <- 0; tauv1 <- 0.04
at <- 1; bt <- 0.1; muw <- 0; tauw <- 1e-4

num.chains <- 3

# defining the data to be fed in
data <- list(x.o=xo, y.o=yo, z.o=zo, vx.o=vxo, vy.o=vyo, vz.o=vzo, t=T, n=n,
             a.tau=at, b.tau=bt, muw=muw, tauw=tauw,
             mup1=mup1, taup1=taup1, muv1=muv1, tauv1=tauv1,
             kd=kd, km=km, g=g
             )

# running the jags model
model <- jags.model(textConnection(model_string), data=data, n.chains=num.chains)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 360
##    Unobserved stochastic nodes: 727
##    Total graph size: 3343
##
## Initializing model
```

```
# burn-in phase
update(model, n.iter=1000, progress.bar="none")
# sampling process
res <- coda.samples(model, n.iter=30000,
                    variable.names=c("tau.p", "tau.v", "tau.po", "tau.vo", "wx", "wy", "wz"),
                    progress.bar="none")
```

```
# some summary statistics
summary(res)
```

```
##
```

```
## Iterations = 2001:32000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 30000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##            Mean      SD Naive SE Time-series SE
## tau.p  424.294 57.4961 0.191654       0.304183
## tau.po 480.657 62.1391 0.207130       0.323215
## tau.v   86.577 24.4491 0.081497       0.389772
## tau.vo   4.891  0.5453 0.001818       0.003084
## wx      -2.694 94.8911 0.316304       0.371783
## wy      15.928 67.4821 0.224940       0.362351
## wz      -1.452 71.6296 0.238765       0.363988
##
## 2. Quantiles for each variable:
##
##            2.5%     25%     50%     75%   97.5%
## tau.p   319.987 384.150 421.442 461.094 545.322
## tau.po  367.556 437.459 477.528 520.558 611.235
## tau.v    46.616  69.037  83.837 101.279 141.613
## tau.vo    3.886   4.513   4.870   5.246   6.018
## wx     -188.298 -66.750  -2.995  61.331 183.434
## wy     -116.442 -29.134  15.924  61.037 149.564
## wz     -142.157 -49.577  -1.089  46.888 139.181
```

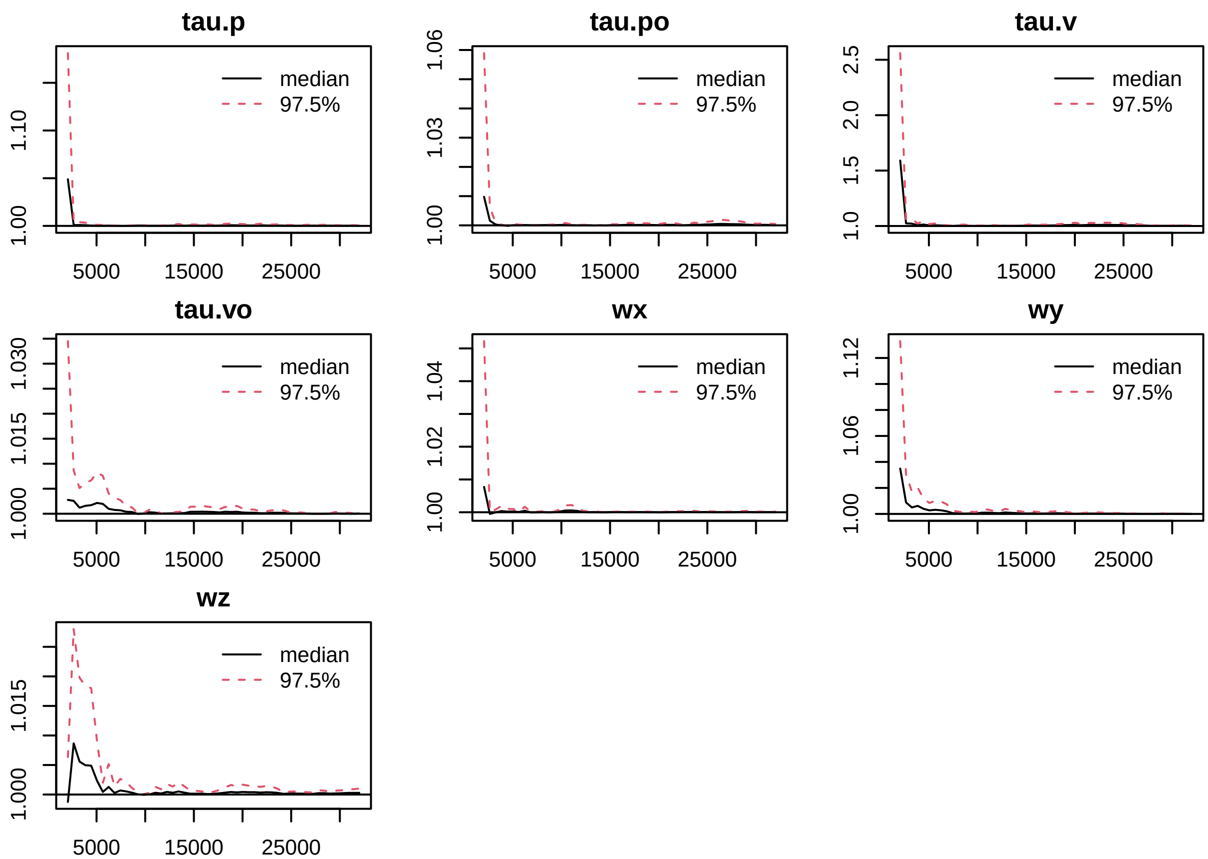
```r
gelman.diag(res)
```

```
## Potential scale reduction factors:
##
##        Point est. Upper C.I.
## tau.p           1          1
## tau.po          1          1
## tau.v           1          1
## tau.vo          1          1
## wx              1          1
## wy              1          1
## wz              1          1
##
## Multivariate psrf
##
## 1
```

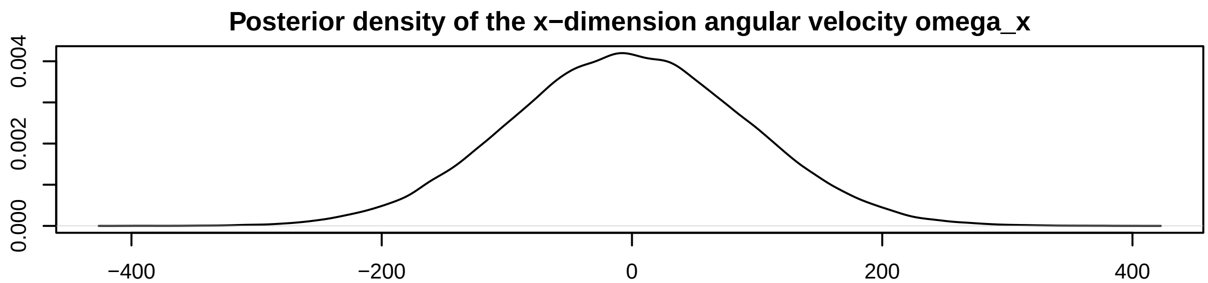
```r
effectiveSize(res[[1]])
```

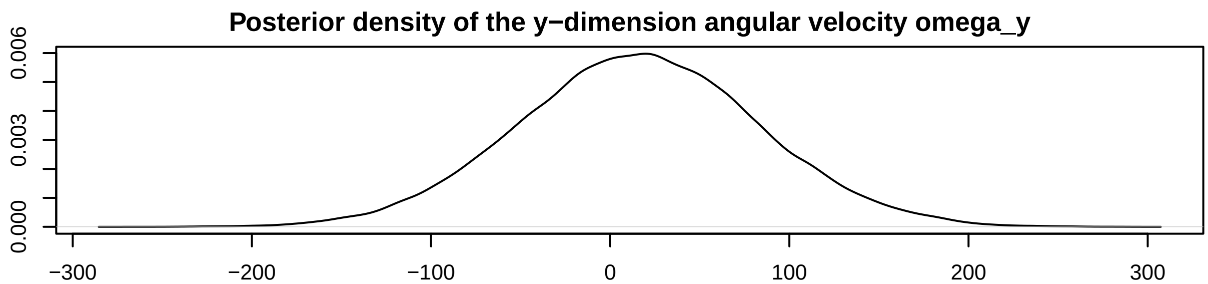
```
##     tau.p    tau.po     tau.v    tau.vo        wx        wy        wz
## 12246.875 12051.004  1368.506  9780.023 22243.229 11183.635 12485.137
```

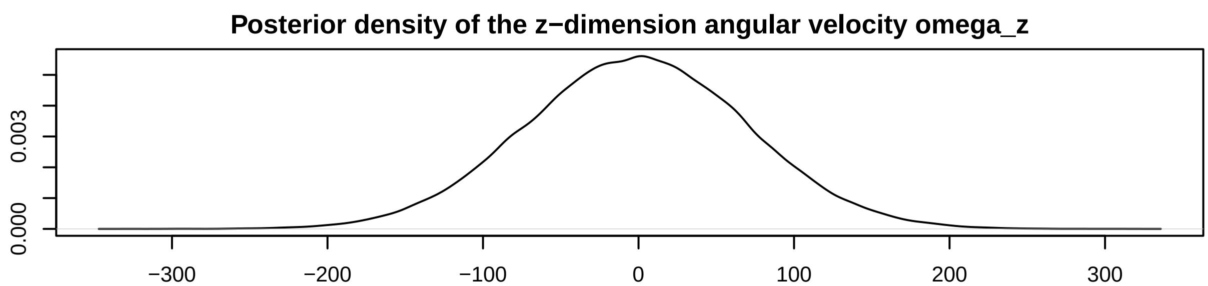
```r
# Gelman-Rubin plots
par(mfrow=c(3, 3))
par(mar=c(2, 2, 2, 2))
gelman.plot(res)
```




```r
# retrieving samples of omegas and plot
res.mat <- as.matrix(res)
par(mfrow=c(3, 1))
par(mar=c(2, 2, 2, 2))
plot(density(res.mat[, 5]), main="Posterior density of the x-dimension angular velocity omega_x")
plot(density(res.mat[, 6]), main="Posterior density of the y-dimension angular velocity omega_y")
plot(density(res.mat[, 7]), main="Posterior density of the z-dimension angular velocity omega_z")
```







Explanation: (Write your explanation here)

Note that all the result numbers recorded in the explanation are from a single "Run All" of the notebook. Since the knitting process will rerun the whole notebook again, the values in the explanation may differ from the output displayed (up to some small randomness).

We choose chain number 3, burn-in step 1000, and sample iteration 30000 to ensure good mixing of the sample results (`plot(res)` can be the checking code, which is omitted here since undemanded). From the summary statistics we can see that the posterior mean estimates of τ_{pos} (423.7460)**,** τ_{vel} (85.9849)**,** τ^o_{pos}(480.1612)**, and** τ^o_{vel} (4.8880), i.e. σ_{pos} (0.04858)**,** σ_{vel} (0.1078)**,** σ^o_{pos}(0.04564)**, and** σ^o_{vel} (0.4523), implies a larger uncertainty in velocities than positions, given their similar scales (as we can see from the data).

Combining the posterior distribution plots of the ωs, we can see that the posterior means of ω_x (-3.8221) and ω_z (-0.8516) implies a relatively slower spin regarding the x and z coordinates than the y coordinate, comparing to ω_y (15.1637), i.e. the spin direction of the ball most likely incline more to the y coordinate. However, the standard deviations and coefficients of variation of the angular velocities are much larger, implying greater uncertainties, which is reasonable since we have only one trajectory data and a very uncertain priori.

We also implemented 3 chains to evaluate the Gelman-Rubin diagnostics, with which of all monitored parameters equal to 1, implying the within-chain variability and the between-chain variability are the same, i.e. there is no or ignorable variation between the 3 chains. The Gelman-Rubin plots also demonstrate good convergence.

The entire set of parameters are chosen so that the effective sizes of the monitored variables are all greater than 1000, with most of them even greater than 10000. This demonstrates a low correlation between samples, ensuring the effectiveness of our sampling. Note that here we choose to calculate the within-chain effective size, which is more meaningful regardless of the number of chains.

Q3)[10 marks] Perform posterior predictive checks on the model in Q2). Explain your choices of test functions, and interpret the results.

- Posterior predictive checks

```
# sampling process of the replicate data
res.rep <- coda.samples(model, n.iter=30000,
                        variable.names=c("xrep.o", "yrep.o", "zrep.o",
                                         "vxrep.o", "vyrep.o", "vzrep.o"),
                        progress.bar="none")

res.rep <- as.matrix(res.rep)

# retrieving the corresponding samples
vxres.rep <- res.rep[, 1: n]
vyres.rep <- res.rep[, 1: n + n]
vzres.rep <- res.rep[, 1: n + 2 * n]
xres.rep <- res.rep[, 1: n + 3 * n]
yres.rep <- res.rep[, 1: n + 4 * n]
zres.rep <- res.rep[, 1: n + 5 * n]

require(fBasics)
```

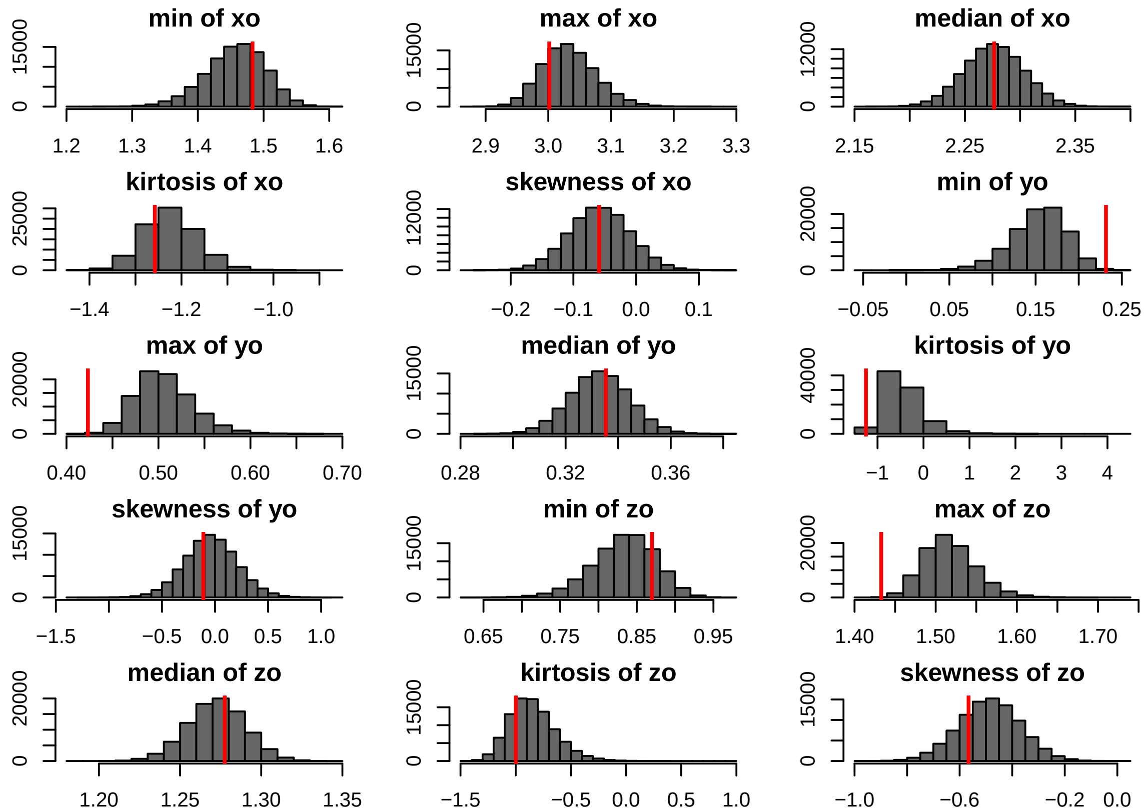
```
## Loading required package: fBasics
```

```
## Warning: package 'fBasics' was built under R version 4.3.2
```

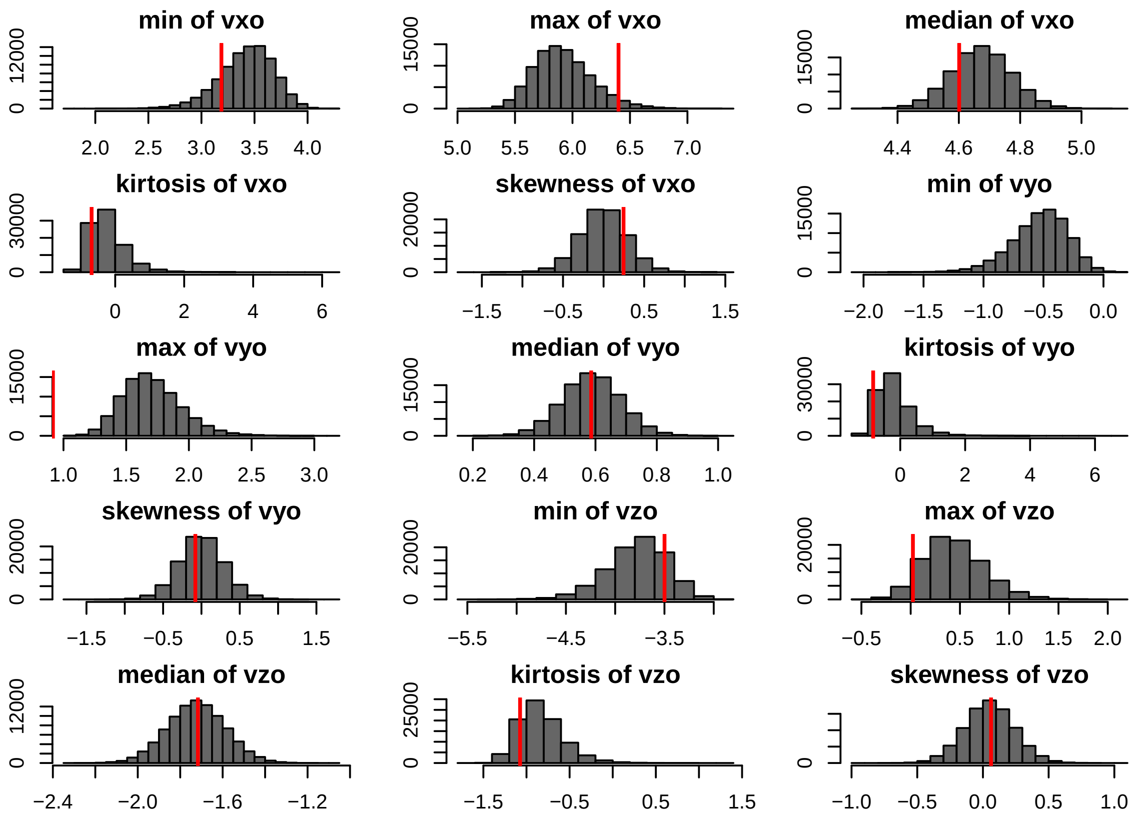
```
# function for replicate data checks, calculating the five statistics and plot
repanalysis <- function(res.ori, res.rep, str) {
  # res.ori: the original data from the real world
  # res,rep: the replicate data from the model

  # calculating the five statistics of the replicate data
  rep.min <- apply(res.rep, 1, min)
  rep.max <- apply(res.rep, 1, max)
  rep.med <- apply(res.rep, 1, median)
  rep.kurt <- apply(res.rep, 1, kurtosis)
  rep.skew <- apply(res.rep, 1, skewness)

  # plotting the original data statistics and replicate data statistics
  hist(rep.min, col="gray40", main=paste("min of", str))
  abline(v=min(res.ori), col="red", lwd=2)
  hist(rep.max, col="gray40", main=paste("max of", str))
  abline(v=max(res.ori), col="red", lwd=2)
  hist(rep.med, col="gray40",main=paste("median of", str))
  abline(v=median(res.ori), col="red", lwd=2)
  hist(rep.kurt, col="gray40", main=paste("kirtosis of", str))
  abline(v=kurtosis(res.ori), col="red", lwd=2)
  hist(rep.skew, col="gray40", main=paste("skewness of", str))
  abline(v=skewness(res.ori), col="red", lwd=2)
}
```

```
# posterior predictive checks for xo, yo, zo
par(mfrow=c(5, 3))
par(mar=c(2, 2, 2, 2))
repanalysis(xo, xres.rep, "xo")
repanalysis(yo, yres.rep, "yo")
```

```r
repanalysis(zo, zres.rep, "zo")
```




```r
# posterior predictive checks for vxo, vyo, vzo
par(mfrow=c(5, 3))
par(mar=c(2, 2, 2, 2))
repanalysis(vxo, vxres.rep, "vxo")
repanalysis(vyo, vyres.rep, "vyo")
repanalysis(vzo, vzres.rep, "vzo")
```



- DIC

  ```
  # DIC check
  dic.samples(model, n.iter=10000)
  ```

  ```
  ## Mean deviance:  -466
  ## penalty 112.2
  ## Penalized deviance: -353.9
  ```

Discussion 1:

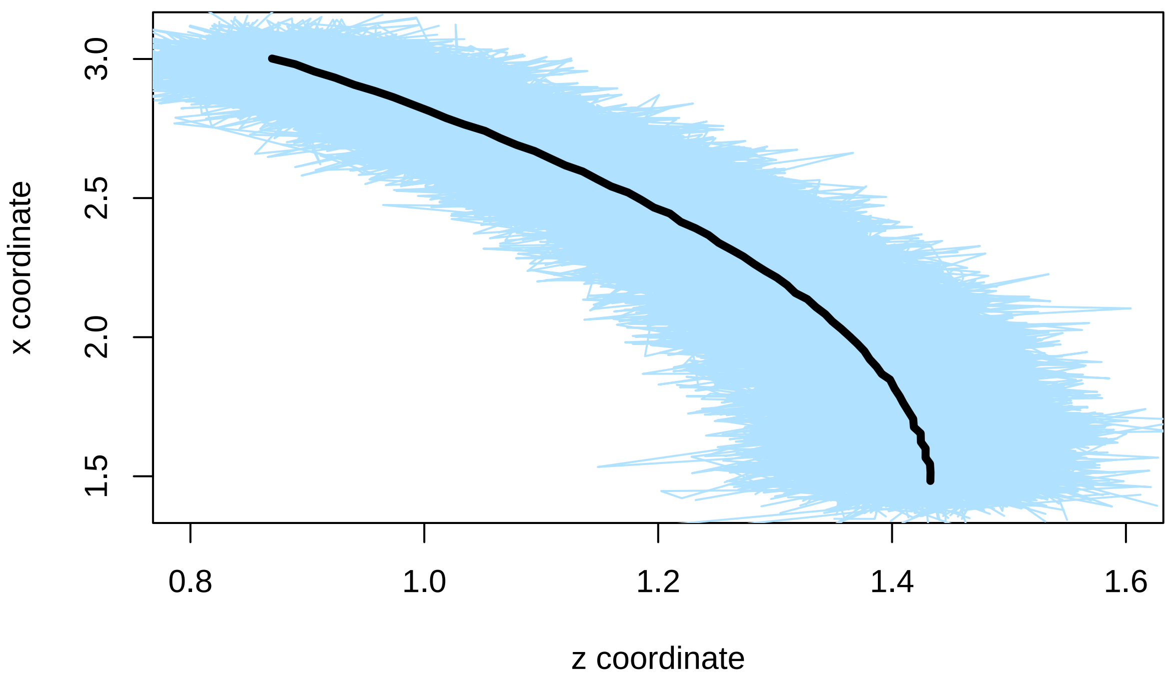
First, we do posterior predictive checks on the observed data, which means reproducing the data based on our model and comparing the relative statistics of the replicated and original data, thus checking the reasonability of our model. Here, we choose minimum, maximum, median, kurtosis, and skewness as the comparison statistics. From the plots, we can see that the median, skewness, and kurtosis of thier posterior distributions generally meet those of the real-world data. Regarding minimum and maximum, the velocities and positions of the x coordinate fit well, but not for the y and z coordinates. This implies that we might get a better model if we separate the priors of the noise parameters of the three coordinates since the variations are likely to be different.

Apart from the demanded check, we also do a DIC (Deviance Information Criterion) check on the model. The result (-354.5) is a reasonably small value, which likely indicates a fairly good model, but we still need to compare it with other model's DIC to ensure this.

Do a plot of the x coordinate of the trajectory against the z coordinate, and include at least 100 of the posterior replicates on the same plot (see Line 351 in the code of Lecture 3 for a similar plot). Discuss the results.

```
# plot as desired
plot(zres.rep[1, ], xres.rep[1, ], col="lightskyblue1",
     xlab="z coordinate", ylab="x coordinate",
```

```
     xlim=c(0.8, 1.6), ylim=c(1.4, 3.1))
for(i in 2: 1000){
  lines(zres.rep[i, ], xres.rep[i, ], col="lightskyblue1")
}
lines(zo, xo, col="black", lwd=4)
```



Discussion 2:

The plot of the x coordinate against the z coordinate is the trajectory of the table tennis ball in the x-z plane. The first 1000 posterior replicates with noises (the light blue lines) altogether form an area with a shape similar to the real-world data (the dark line), indicating that the model has correctly done the expected modelling.

Q4)[10 marks] In this question, we are will use the model to predict the trajectory for the next 6 time steps, and compare it to the observed values. First, we load the data including the next 6 steps (note that these observations cannot be used for prediction, only for testing).

```
n.pr <- 66;
xyz.obs <- h5read("MN5008_grid_data_equal_speeds.hdf5",
                  "/originals/405/positions")[, 2: (n.pr + 1)]
#Read positions of simulation number 405
xo <- xyz.obs[1, ]
yo <- xyz.obs[2, ]
zo <- xyz.obs[3, ]

vxvyvz.obs <- h5read("MN5008_grid_data_equal_speeds.hdf5",
                     "/originals/405/velocities")[, 2: (n.pr + 1)]
```

```r
#Read velocities of simulation number 405
vxo <- vxvyvz.obs[1, ]
vyo <- vxvyvz.obs[2, ]
vzo <- vxvyvz.obs[3, ]

T <- h5read("MN5008_grid_data_equal_speeds.hdf5", "/originals/405/time_stamps")[2: (n.pr + 1)]
```

Explain how you can implement a posterior predictive model for the position and velocity variables at the next 6 time steps, i.e. T[61], T[62],. . . ,T[66]. Do not pass along the position and velocity observations at these time points to the model in the data (you can replace them with NA if using JAGS). Implement this in JAGS or Stan. Compute the posterior predictive mean of all position and velocity components at these new time steps, and compare them with their observed values. Compute the Euclidean distance between the observed position and the posterior predictive mean of the position variables at the next 6 time steps, and do the same for the velocities. Discuss the predictive accuracy of this Bayesian model.

```r
# constructing training data
xo.pr <- c(xo[1: n], rep(NA, 6))
yo.pr <- c(yo[1: n], rep(NA, 6))
zo.pr <- c(zo[1: n], rep(NA, 6))
vxo.pr <- c(vxo[1: n], rep(NA, 6))
vyo.pr <- c(vyo[1: n], rep(NA, 6))
vzo.pr <- c(vzo[1: n], rep(NA, 6))
```

```r
# making predictions by running the model fed with training data
data.pred <- list(x.o=xo.pr, y.o=yo.pr, z.o=zo.pr,
                  vx.o=vxo.pr, vy.o=vyo.pr, vz.o=vzo.pr, t=T, n=n.pr,
                  a.tau=at, b.tau=bt, muw=muw, tauw=tauw,
                  mup1=mup1, taup1=taup1, muv1=muv1, tauv1=tauv1,
                  kd=kd, km=km, g=g
                  )
num.chains <- 1

model.pr <- jags.model(textConnection(model_string), data=data.pred, n.chains=num.chains)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 360
##    Unobserved stochastic nodes: 835
##    Total graph size: 3679
##
## Initializing model
```

```r
# burn-in phase
update(model.pr, n.iter=1000)
# sampling process
pred.varlist <- c("x.o[61:66]", "y.o[61:66]", "z.o[61:66]",
               "vx.o[61:66]", "vy.o[61:66]", "vz.o[61:66]")
res.pr <- coda.samples(model.pr, variable.names=pred.varlist, n.iter=30000)
```

```r
# posterior mean predictions
res.mean <- apply(res.pr[[1]], 2, mean)

pred.res <- as.data.frame(t(matrix(res.mean, ncol=6)))
```

```r
colnames(pred.res) <- as.character((n + 1): n.pr)
rownames(pred.res) <- c("Velocities of x coordinate",
                        "Velocities of y coordinate",
                        "Velocities of z coordinate",
                        "Positions of x coordinate",
                        "Positions of y coordinate",
                        "Positions of z coordinate"
                        )
print(pred.res)
```

```
##                                   61         62         63         64
## Velocities of x coordinate  4.1565737  4.1429534  4.1270536  4.1059362
## Velocities of y coordinate  0.5041079  0.5032765  0.4973083  0.4970954
## Velocities of z coordinate -3.2880343 -3.3315918 -3.3737071 -3.4106231
## Positions of x coordinate   3.0268768  3.0505385  3.0727382  3.0959686
## Positions of y coordinate   0.4265279  0.4296616  0.4318648  0.4345088
## Positions of z coordinate   0.8526514  0.8333740  0.8159819  0.7975440
##                                   65         66
## Velocities of x coordinate  4.0880729  4.0683215
## Velocities of y coordinate  0.4959655  0.4897301
## Velocities of z coordinate -3.4546182 -3.5049169
## Positions of x coordinate   3.1189383  3.1444942
## Positions of y coordinate   0.4375442  0.4406248
## Positions of z coordinate   0.7786104  0.7558802
```

```r
# Euclidean distance
pred.dist.res <- as.data.frame(
  rbind(sqrt((pred.res[4, ] - xo[61: 66]) ** 2 +
             (pred.res[5, ] - yo[61: 66]) ** 2 +
             (pred.res[6, ] - zo[61: 66]) ** 2),
        sqrt((pred.res[1, ] - vxo[61: 66]) ** 2 +
             (pred.res[2, ] - vyo[61: 66]) ** 2 +
             (pred.res[3, ] - vzo[61: 66]) ** 2)
        ))

colnames(pred.dist.res) <- as.character((n + 1): n.pr)
rownames(pred.dist.res) <- c("Euclidean distances of positions",
                             "Euclidean distances of velocities")

print(pred.dist.res)
```

```
##                                            61         62          63
## Euclidean distances of positions  0.002000525 0.00492279 0.001546866
## Euclidean distances of velocities 0.484821974 0.91225768 0.679190181
##                                            64          65          66
## Euclidean distances of positions  0.002223358 0.001824005 0.006357024
## Euclidean distances of velocities 0.347903803 0.331355581 1.086762253
```

Explanation: (Write your explanation here)

We first replace the new data at the last 6 time steps with `NA` for prediction, then feed the data again into the same model (`n=60` also needs to be replaced with `n=66`). Then we do the same steps of model creation, burn-in, sampling, and mean calculation to get the posterior mean of the velocity and position data at the last 6 time steps, which is also the expected prediction result.

The prediction results and their Euclidean distances with real-world data are displayed as data frames. The distances of the prediction of positions are smaller than those of velocities, implying more precise predictions in positions than in velocities, given their similar numerical scales, which is the same as the conclusion we obtained from Q2's summary statistics.

Q5)[10 marks] In this question, we will try to improve the model by using a different numerical discretization of the ODE

$$\frac{dX}{dt} = V, \quad \frac{dV}{dt} = k_d \|V\| V + k_m \omega \times V + g.$$

In Q1), we have used the Euler-Mayurama scheme, and added some model and observation noise.

In this question, you are expected to choose a different scheme (see e.g. Lecture 4 for some examples, or you could use the one based on the analytical solution of the ODE described in the paper "Optimal State Estimation of Spinning Ping-Pong Ball Using Continuous Motion Model"). You should still allow for model and observation noises. You can also consider different covariance matrices, such as ones that allow correlation between the model noise for x and v_x, and similarly, between the observation noise of x and v_x, etc. (such models would have additional parameters related to the amount of correlation). Describe the motivation for your scheme. Implement the new model similarly to Q2) (ESS should be at least 1000 for all model parameters), do the posterior predictive checks from Q3), and compare its predictive performance on the next 6 datapoints as in Q4). Discuss the results.

```
# jags model string with Leapfrog
model_string.lf <- "
model {
  tau.p ~ dgamma(a.tau, b.tau)
  tau.po ~ dgamma(a.tau, b.tau)
  tau.v ~ dgamma(a.tau, b.tau)
  tau.vo ~ dgamma(a.tau, b.tau)

  x[1] ~ dnorm(mup1, taup1)
  y[1] ~ dnorm(mup1, taup1)
  z[1] ~ dnorm(mup1, taup1)
  vx[1] ~ dnorm(muv1, tauv1)
  vy[1] ~ dnorm(muv1, tauv1)
  vz[1] ~ dnorm(muv1, tauv1)

  wx ~ dnorm(muw, tauw)
  wy ~ dnorm(muw, tauw)
  wz ~ dnorm(muw, tauw)

  for (i in 1: (n - 1)) {
    delta[i] = t[i + 1] - t[i]

    V[i] = sqrt(vx[i] ** 2 + vy[i] ** 2 + vz[i] ** 2)

    # Leapfrog
    vx.half[i] ~ dnorm(vx[i] +
                       delta[i] / 2 * (kd * V[i] * vx[i] + km * (wy * vz[i] - wz * vy[i])),
                       tau.v)
    vy.half[i] ~ dnorm(vy[i] +
                       delta[i] / 2 * (kd * V[i] * vy[i] + km * (wz * vx[i] - wx * vz[i])),
                       tau.v)
    vz.half[i] ~ dnorm(vz[i] +
                       delta[i] / 2 * (kd * V[i] * vz[i] + km * (wx * vy[i] - wy * vx[i]) - g),
```

```
                       tau.v)

    x[i + 1] ~ dnorm(x[i] + delta[i] * vx.half[i], tau.p)
    y[i + 1] ~ dnorm(y[i] + delta[i] * vy.half[i], tau.p)
    z[i + 1] ~ dnorm(z[i] + delta[i] * vz.half[i], tau.p)

    V.half[i] = sqrt(vx.half[i] ** 2 + vy.half[i] ** 2 + vz.half[i] ** 2)

    vx[i + 1] ~ dnorm(vx.half[i] +
                      delta[i] / 2 * (kd * V[i] * vx.half[i] + km * (wy * vz.half[i] - wz * vy.half[i])
                      tau.v)
    vy[i + 1] ~ dnorm(vy.half[i] +
                      delta[i] / 2 * (kd * V[i] * vy.half[i] + km * (wz * vx.half[i] - wx * vz.half[i])
                      tau.v)
    vz[i + 1] ~ dnorm(vz.half[i] +
                      delta[i] / 2 * (kd * V[i] * vz.half[i] + km * (wx * vy.half[i] - wy * vx.half[i])
                      tau.v)
  }

  for (i in 1: n) {
    x.o[i] ~ dnorm(x[i], tau.po)
    y.o[i] ~ dnorm(y[i], tau.po)
    z.o[i] ~ dnorm(z[i], tau.po)
    vx.o[i] ~ dnorm(vx[i], tau.vo)
    vy.o[i] ~ dnorm(vy[i], tau.vo)
    vz.o[i] ~ dnorm(vz[i], tau.vo)

    xrep.o[i] ~ dnorm(x[i], tau.po)
    yrep.o[i] ~ dnorm(y[i], tau.po)
    zrep.o[i] ~ dnorm(z[i], tau.po)
    vxrep.o[i] ~ dnorm(vx[i], tau.vo)
    vyrep.o[i] ~ dnorm(vy[i], tau.vo)
    vzrep.o[i] ~ dnorm(vz[i], tau.vo)
  }

}
"

# model running parameters
num.chains <- 3

# running the new jags model
model.lf <- jags.model(textConnection(model_string.lf), data=data, n.chains=num.chains)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 360
##    Unobserved stochastic nodes: 904
##    Total graph size: 5348
##
## Initializing model
```

```r
# burn-in phase
update(model.lf, n.iter=5000, progress.bar="none")
# sampling process
res.lf <- coda.samples(model.lf, n.iter=50000,
                       variable.names=c("tau.p", "tau.v", "tau.po", "tau.vo", "wx", "wy", "wz"),
                       progress.bar="none")
```

```r
# some summary statistics
summary(res.lf)
```

```
##
## Iterations = 6001:56000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 50000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##            Mean      SD Naive SE Time-series SE
## tau.p  423.3068 57.5373 0.148561        0.24240
## tau.po 479.8907 62.1124 0.160373        0.25169
## tau.v  105.8910 27.2919 0.070467        0.45078
## tau.vo   4.8514  0.5514 0.001424        0.00332
## wx      -2.3726 95.8552 0.247497        0.28019
## wy      14.1414 75.1537 0.194046        0.31434
## wz       0.4307 78.1575 0.201802        0.29589
##
## 2. Quantiles for each variable:
##
##            2.5%     25%     50%     75%   97.5%
## tau.p   319.100 383.014 420.634 460.384 543.822
## tau.po  366.669 436.720 477.124 519.845 610.163
## tau.v    60.690  86.223 103.304 122.661 166.223
## tau.vo    3.834   4.469   4.829   5.208   5.997
## wx     -189.715 -67.313  -2.389  62.140 185.642
## wy     -134.349 -35.947  14.296  64.385 161.475
## wz     -152.785 -52.296   0.383  53.135 153.828
```

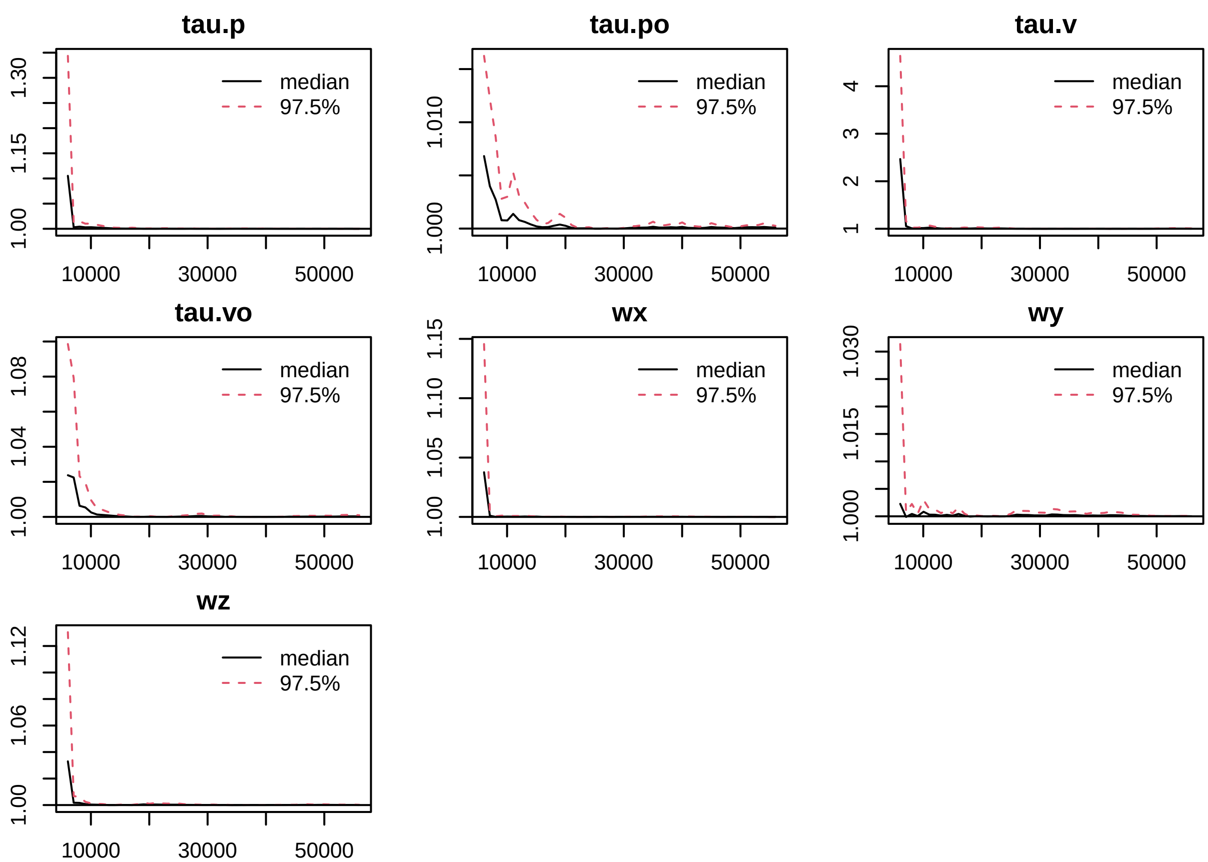
```r
gelman.diag(res.lf)
```

```
## Potential scale reduction factors:
##
##        Point est. Upper C.I.
## tau.p           1       1.00
## tau.po          1       1.00
## tau.v           1       1.01
## tau.vo          1       1.00
## wx              1       1.00
## wy              1       1.00
## wz              1       1.00
##
## Multivariate psrf
##
## 1
```

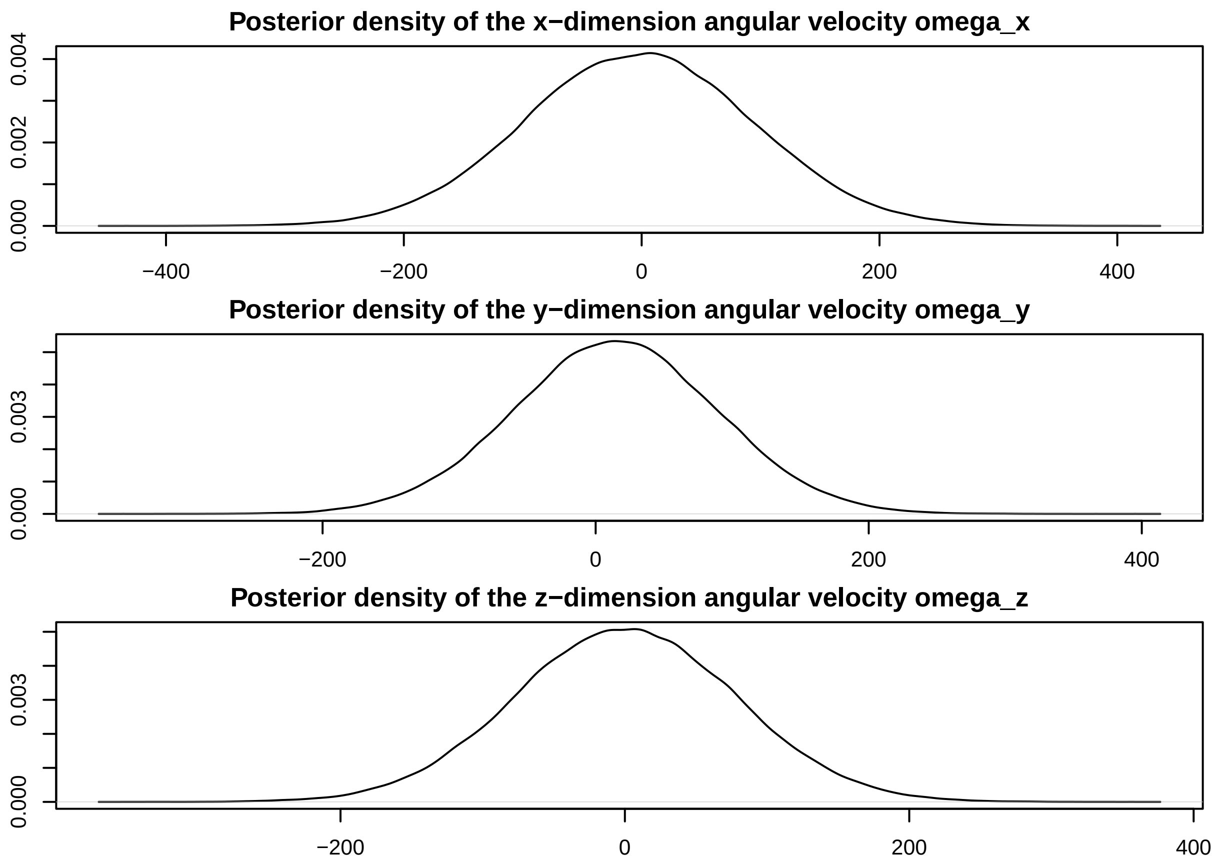
```r
effectiveSize(res.lf[[1]])
```

```
##     tau.p    tau.po     tau.v    tau.vo        wx        wy        wz
## 18542.339 20480.545  1258.623  9524.097 40127.293 20649.000 26065.531
```

```r
# Gelman-Rubin plots
par(mfrow=c(3, 3))
par(mar=c(2, 2, 2, 2))
gelman.plot(res.lf)
```

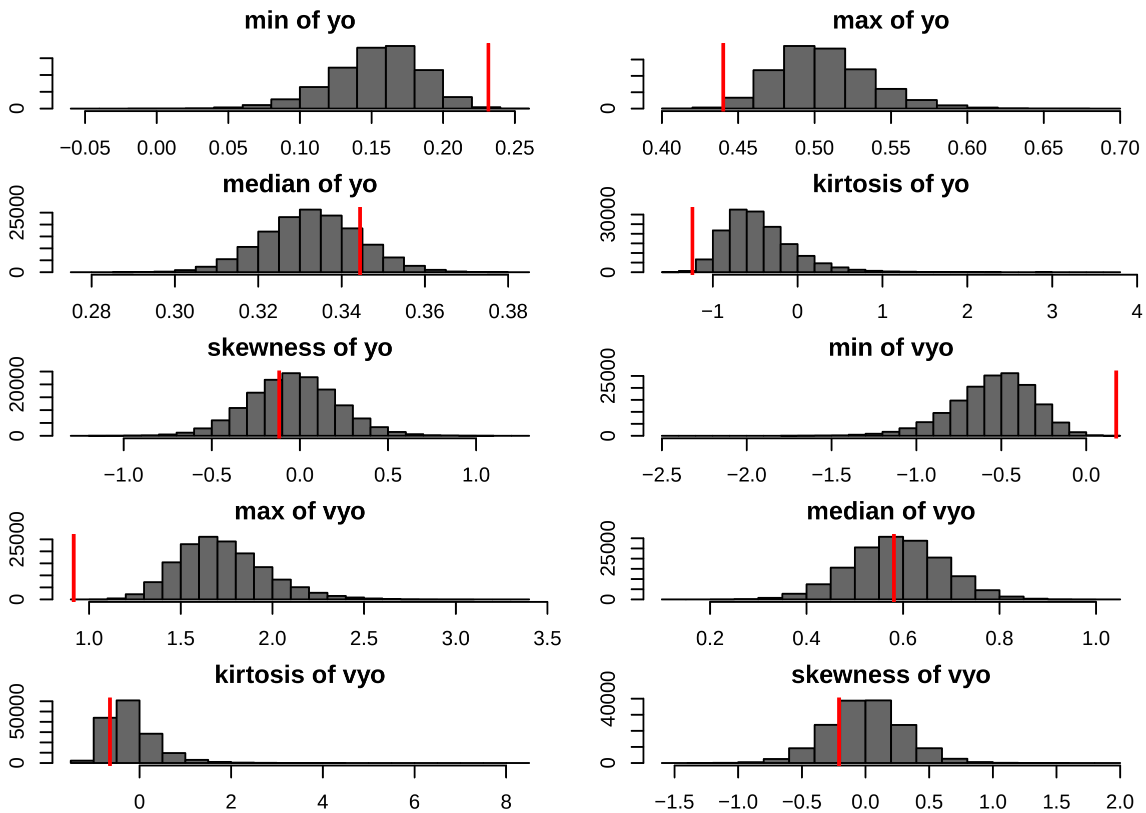



```r
# retrieving samples of omegas and plot
res.lf.mat <- as.matrix(res.lf)
par(mfrow=c(3, 1))
par(mar=c(2, 2, 2, 2))
plot(density(res.lf.mat[, 5]), main="Posterior density of the x-dimension angular velocity omega_x")
plot(density(res.lf.mat[, 6]), main="Posterior density of the y-dimension angular velocity omega_y")
plot(density(res.lf.mat[, 7]), main="Posterior density of the z-dimension angular velocity omega_z")
```



- Posterior predictive checks

```
# sampling process of the replicate data
res.lf.rep <- coda.samples(model.lf, n.iter=50000,
                           variable.names=c("xrep.o", "yrep.o", "zrep.o",
                                            "vxrep.o", "vyrep.o", "vzrep.o"),
                           progress.bar="none")

res.lf.rep <- as.matrix(res.lf.rep)

# retrieving the corresponding samples
vxres.lf.rep <- res.lf.rep[, 1: n]
vyres.lf.rep <- res.lf.rep[, 1: n + n]
vzres.lf.rep <- res.lf.rep[, 1: n + 2 * n]
xres.lf.rep <- res.lf.rep[, 1: n + 3 * n]
yres.lf.rep <- res.lf.rep[, 1: n + 4 * n]
zres.lf.rep <- res.lf.rep[, 1: n + 5 * n]

require(fBasics)

# posterior predictive checks for yo, vy0
par(mfrow=c(5, 2))
par(mar=c(2, 2, 2, 2))
repanalysis(yo, yres.lf.rep, "yo")
repanalysis(vyo, vyres.lf.rep, "vyo")
```

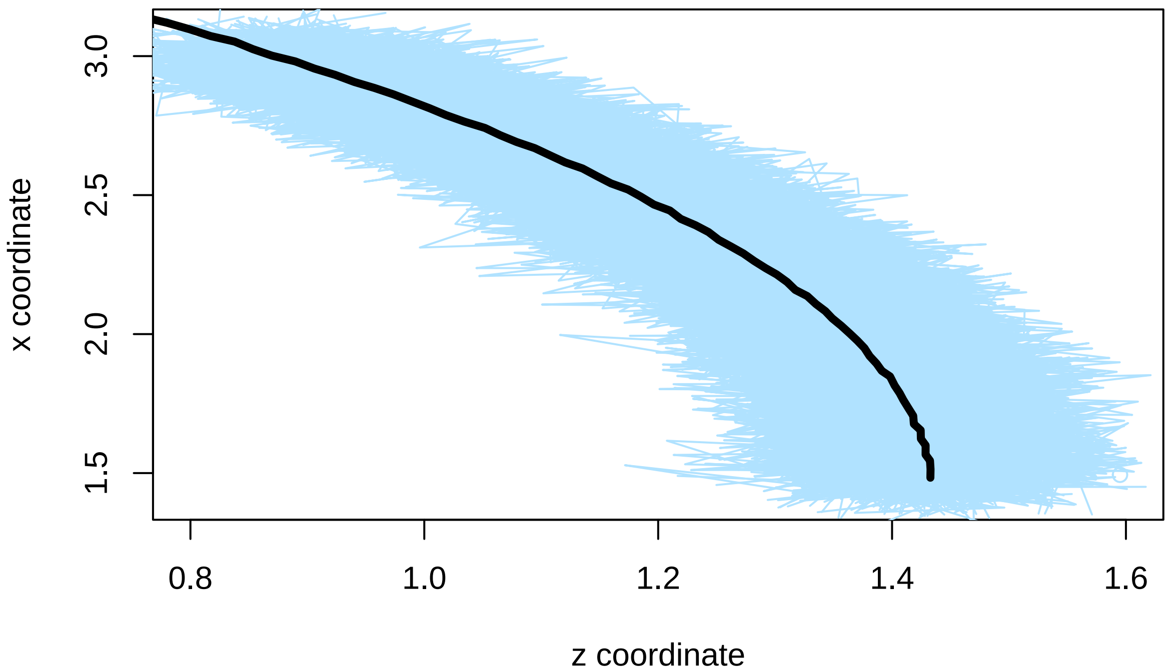


- DIC

```r
# DIC check
dic.samples(model.lf, n.iter=10000)
```

```
## Mean deviance:  -464.1
## penalty 117.8
## Penalized deviance: -346.3
```

```r
# plot as desired in Q3
plot(zres.lf.rep[1, ], xres.lf.rep[1, ], col="lightskyblue1",
     xlab="z coordinate", ylab="x coordinate",
     xlim=c(0.8, 1.6), ylim=c(1.4, 3.1))
for(i in 2: 1000){
  lines(zres.lf.rep[i, ], xres.lf.rep[i, ], col="lightskyblue1")
}
lines(zo, xo, col="black", lwd=4)
```




```r
num.chains <- 1

# running the new model fed with training data
model.lf.pr <- jags.model(textConnection(model_string.lf), data=data.pred, n.chains=num.chains)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 360
##    Unobserved stochastic nodes: 1030
##    Total graph size: 5888
##
## Initializing model
```

```r
# burn-in phase
update(model.lf.pr, n.iter=1000)
# sampling process
pred.varlist <- c("x.o[61:66]", "y.o[61:66]", "z.o[61:66]",
               "vx.o[61:66]", "vy.o[61:66]", "vz.o[61:66]")
res.lf.pr <- coda.samples(model.lf.pr, variable.names=pred.varlist, n.iter=30000)
```

```r
# posterior mean predictions
res.lf.mean <- apply(res.lf.pr[[1]], 2, mean)

pred.res.lf <- as.data.frame(t(matrix(res.lf.mean, ncol=6)))
```

```r
colnames(pred.res.lf) <- as.character((n + 1): n.pr)
rownames(pred.res.lf) <- c("Velocities of x coordinate",
                           "Velocities of y coordinate",
                           "Velocities of z coordinate",
                           "Positions of x coordinate",
                           "Positions of y coordinate",
                           "Positions of z coordinate")

print(pred.res.lf)
```

```
##                                    61         62         63         64
## Velocities of x coordinate  4.1638688  4.1467948  4.1297566  4.1127524
## Velocities of y coordinate  0.4909138  0.4905731  0.4895648  0.4919367
## Velocities of z coordinate -3.3058455 -3.3473126 -3.3892194 -3.4306685
## Positions of x coordinate   3.0265757  3.0499587  3.0718013  3.0942455
## Positions of y coordinate   0.4261043  0.4291045  0.4319555  0.4338891
## Positions of z coordinate   0.8520271  0.8329662  0.8145736  0.7958161
##                                    65         66
## Velocities of x coordinate  4.0943425  4.0749039
## Velocities of y coordinate  0.4849787  0.4896946
## Velocities of z coordinate -3.4686265 -3.5093366
## Positions of x coordinate   3.1171238  3.1431398
## Positions of y coordinate   0.4364318  0.4396313
## Positions of z coordinate   0.7766560  0.7545856
```

```r
# Euclidean distance
pred.dist.res.lf <- as.data.frame(
  rbind(sqrt((pred.res.lf[4, ] - xo[61: 66]) ** 2 +
               (pred.res.lf[5, ] - yo[61: 66]) ** 2 +
               (pred.res.lf[6, ] - zo[61: 66]) ** 2),
        sqrt((pred.res.lf[1, ] - vxo[61: 66]) ** 2 +
               (pred.res.lf[2, ] - vyo[61: 66]) ** 2 +
               (pred.res.lf[3, ] - vzo[61: 66]) ** 2)
        ))

colnames(pred.dist.res.lf) <- as.character((n + 1): n.pr)
rownames(pred.dist.res.lf) <- c("Euclidean distances of positions",
                                "Euclidean distances of velocities")

print(pred.dist.res.lf)
```

```
##                                          61         62          63          64
## Euclidean distances of positions  0.0022809 0.00540492 0.002803403 0.004489006
## Euclidean distances of velocities 0.5070618 0.91560828 0.671454092 0.354620662
##                                            65          66
## Euclidean distances of positions  0.004710865 0.006553046
## Euclidean distances of velocities 0.322344867 1.094487533
```

```r
# mean distances of positions and velocities for the two models
meandist <- rowMeans(pred.dist.res)
meandist.lf <- rowMeans(pred.dist.res.lf)
cat("Old model:\n")
```

```
## Old model:
```

```r
cat("Mean distance:", meandist[1], "\n")
```

```
## Mean distance: 0.003145761
```

```r
cat("Mean distance of velocities:", meandist[2], "\n")
```

```
## Mean distance of velocities: 0.6403819
```

```r
cat("New model:\n")
```

```
## New model:
```

```r
cat("Mean distance of positions:", meandist.lf[1], "\n")
```

```
## Mean distance of positions: 0.00437369
```

```r
cat("Mean distance of velocities:", meandist.lf[2], "\n")
```

```
## Mean distance of velocities: 0.6442629
```

Explanation: (Write your explanation here)

For this question, we replace the Euler-Mayurama scheme with the Leapfrog scheme as the discretization of the given ODE. The changes are mainly in the update rules of the loop in the Jags model, where we include the update of velocities at the half-time steps.

From the results, we can see that the model also demonstrates similarly good results, but the DIC value (-345.8) shows that the new model is a little inferior in performance compared to the original model (-354.5). The main contribution of the falling value is from the penalty term, increasing from 112 to 118.3, with almost the same mean deviance.

In terms of the prediction accuracy, we calculate the mean distance of the last six time steps for prediction. Generally, the two model's prediction accuracy is very much the same, with the new model only slightly better in both velocities (0.6377 compared with 0.6412) and positions (0.003415 compared with 0.004175).

Therefore, we may conclude that the Euler-Mayurama scheme is enough for a decent model. While obtaining similar prediction accuracy, the Leapfrog scheme introduces more parameters, which is unfavorable under this particular circumstance.