s2596860_AssignmentI

2024-02-28

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

QUESTION 1)(a) - DAG Representation of the Model

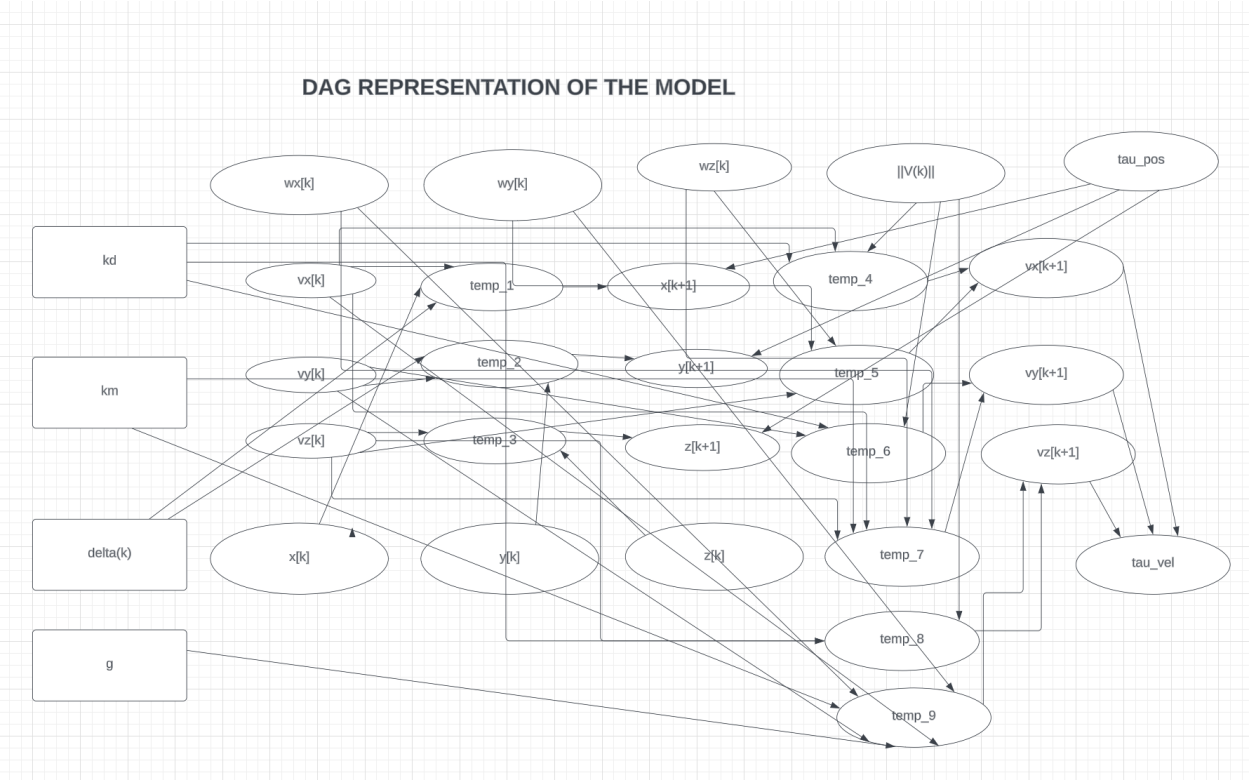


Figure 1: DAG Representation of Euler-Mayurama Model

QUESTION 1(b) - Explanation on Setting Up the Priors

The initial values for the position (x,y,z) and the velocity (vx,vy,vz) components are set to have priors sampled from Normal distribution.And when it comes to setting priors for the **tau_components** (tau_pos,tau_vel,tau_o_pos,tau_o_vel), **gamma distribution** since the domain of the distribution is non negative (0,infinity) and here the data we have is also non negative continuous data. It is thereby optimal to choose this as a prior for the precision of a normal distribution.

On the other hand , when it comes to setting priors for the **angular velocity** components (wx, wy, wz), since we know that the posterior distribution is going to be a Normal distribution , and thereon by looking at the model equations and the formulae involved, we need to go for a conjugate prior on the angular velocity components, thus **Normal distribution** would be optimal.

The link below here supports on the arguments made above on choosing the priors.

LINK FOR REFERENCE : 'https://math.stackexchange.com/questions/2158300/why-choose-gamma-distribution-as-prior'

QUESTION 2 - IMPLEMENTATION OF THE MODEL IN JAGS AND EVALUATING VARIOUS STATISTICS

Loading the dataset and depicting a 3D plot of that one selected trajectory

```
library(rhdf5)
```

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

```
#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)];
```

The below provided R code deploys JAGS, a probabilistic programming language, to construct a Bayesian hierarchical model for the trajectory of a spinning ping-pong ball without wind interference. The model encompasses continuous motion dynamics using Euler-Mayurama discretization and incorporates Gaussian observation noise. Initial positions **(x[1],y[1],z[1])** , velocities **(vx[1],vy[1],vz[1])**, angular velocities **(wx,wy,wz)** and precision parameters (tau_pos,tau_vel,tau_o_pos,tau_o_vel)are assigned prior distributions. The likelihood function encapsulates the ball's positional and velocity evolution over time, integrating physical equations and accounting for observational noise. Constants such as gravity (g), drag coefficient(kd) and lift coefficient(km) are specified, and the model undergoes Markov Chain Monte Carlo (MCMC) sampling to derive posterior distributions for key parameters. The resulting samples, including those for **tau_pos, tau_vel, tau_o_pos, tau_o_vel, wx, wy, wz** are then subjected to inference and prediction.

```
#install.packages("rjags")
library(rjags)
```

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

```
## Linked to JAGS 4.3.2
```

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

```
library(coda)

# Model string
model_string_jags <- "
model {

x[1] ~ dnorm(0,1)
y[1] ~ dnorm(0,1)
z[1] ~ dnorm(0,1)
vx[1] ~ dnorm(0,25)
vy[1] ~ dnorm(0,25)
vz[1] ~ dnorm(0,25)

# Setting gamma priors for the 'tau' values
tau_pos ~ dgamma(1,1)
tau_vel ~ dgamma(1,1)
tau_o_pos ~ dgamma(1,1)
tau_o_vel ~ dgamma(1,1)

# Setting normal priors for the 'omega' values
wx ~ dnorm(0,1)
wy ~ dnorm(0,1)
wz ~ dnorm(0,1)

# Likelihood calculation
for (k in 1:(n-1)) {

   V_magnitude[k] <- sqrt(vx[k]^2 + vy[k]^2 + vz[k]^2)

   x[k+1] ~ dnorm(x[k] +(vx[k] * delta[k]), tau_pos)
   xrep[k+1] ~ dnorm(x[k] +(vx[k] * delta[k]), tau_pos)

   y[k+1] ~ dnorm(y[k] +(vy[k] * delta[k]), tau_pos)
   yrep[k+1] ~ dnorm(y[k] +(vy[k] * delta[k]), tau_pos)

   z[k+1] ~ dnorm(z[k] +(vz[k] * delta[k]), tau_pos)
   zrep[k+1] ~ dnorm(z[k] +(vz[k] * delta[k]), tau_pos)


   vx[k+1] ~ dnorm(vx[k] + (kd * V_magnitude[k] * vx[k] + km * (wy * vz[k] - wz * vy[k])) * delta[k], ta
   vxrep[k+1] ~ dnorm(vx[k] + (kd * V_magnitude[k] * vx[k] + km * (wy * vz[k] - wz * vy[k])) * delta[k]

   vy[k+1] ~ dnorm(vy[k] + (kd * V_magnitude[k] * vy[k] + km * (wz * vx[k] - wx * vz[k])) * delta[k], ta
   vyrep[k+1] ~ dnorm(vy[k] + (kd * V_magnitude[k] * vy[k] + km * (wz * vx[k] - wx * vz[k])) * delta[k]

   vz[k+1] ~ dnorm(vz[k] + (kd * V_magnitude[k] * vz[k] + km * (wx * vy[k] - wy * vx[k]) - g) * delta[k]
   vzrep[k+1] ~ dnorm(vz[k] + (kd * V_magnitude[k] * vz[k] + km * (wx * vy[k] - wy * vx[k]) - g) * delta

}

# Observation model
  for(k in 1:(n-1)){
```

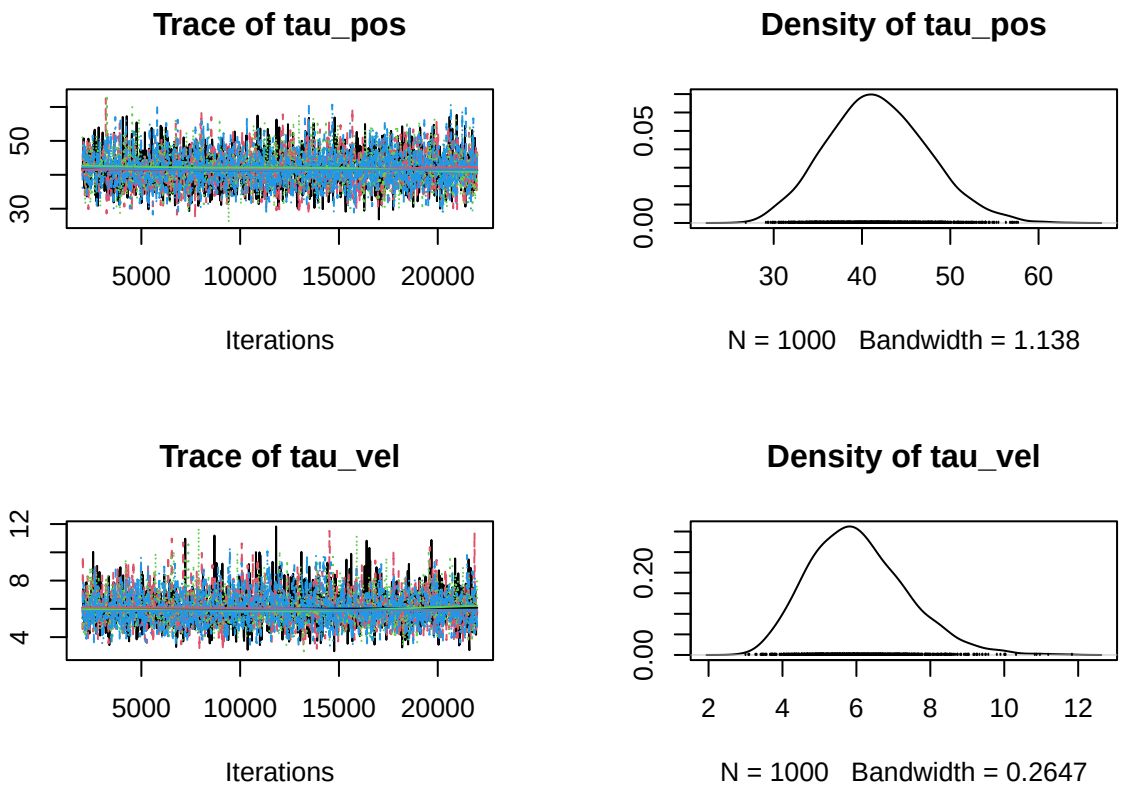
```
    xo[k] ~ dnorm(x[k],tau_o_pos)
    yo[k] ~ dnorm(y[k],tau_o_pos)
    zo[k] ~ dnorm(z[k],tau_o_pos)
    vxo[k] ~ dnorm(vx[k],tau_o_vel)
    vyo[k] ~ dnorm(vy[k],tau_o_vel)
    vzo[k] ~ dnorm(vz[k],tau_o_vel)
  }
}"

g <- 9.827
kd <- (-1/(2*0.0027))*0.456*1.29*0.001256
km <- (1/(2*0.0027))*1*1.29*0.001256*0.02

jags_data = list (
       n = n,
       xo = xo,
       yo = yo,
       zo = zo,
       vxo = vxo,
       vyo = vyo,
       vzo = vzo,
       g = g,
       km = km,
       kd = kd,
       delta = diff(T)
       )

jags_model = jags.model(textConnection(model_string_jags), data = jags_data, n.chains = 4)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 354
##    Unobserved stochastic nodes: 721
##    Total graph size: 3272
##
## Initializing model
```

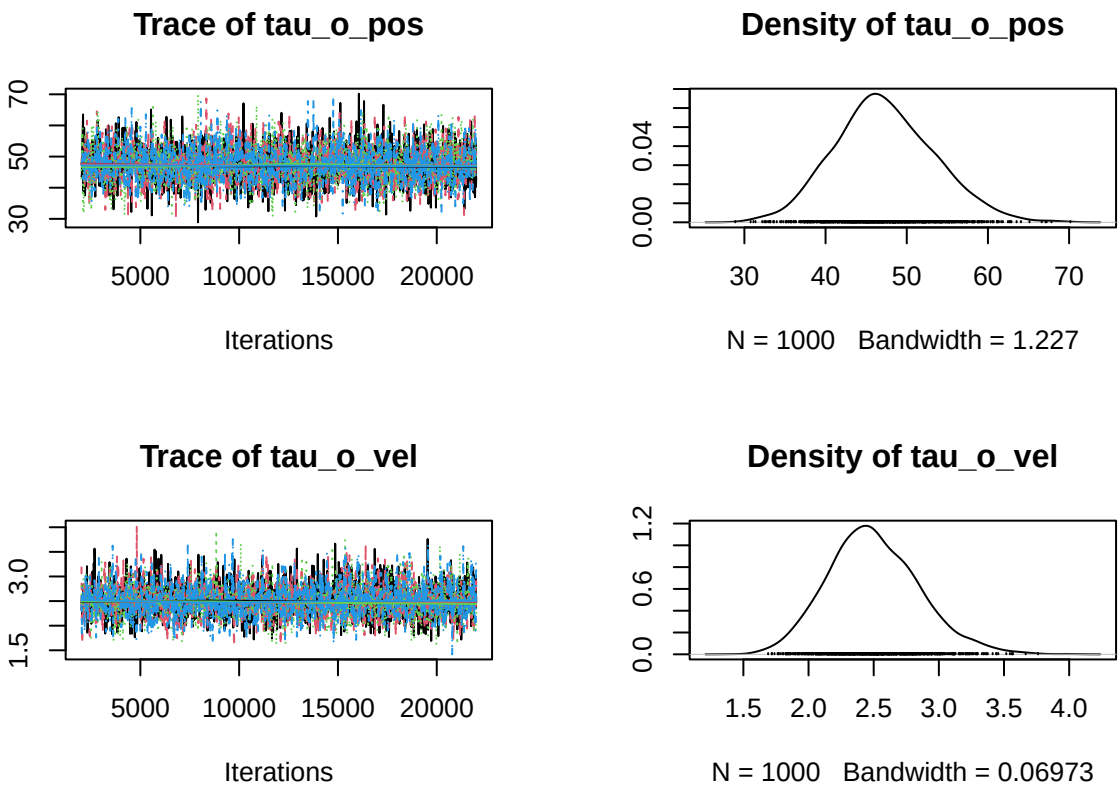
```
update(jags_model,1000)

params_of_interest <- c("tau_pos","tau_vel","tau_o_pos","tau_o_vel","wx","wy","wz")

# Setting the number of iterations, burn-in, and thinning parameters
n.iter <- 20000
n.burn <- 1000
n.thin <- 20

jags_samples=coda.samples(jags_model,variable.names=params_of_interest,n.iter = n.iter, thin = n.thin,p
```

Running the traceplots on various parameters for getting to know on the convergence rate. As we could observe from the below plots that every parameter present does have a good mixing and a faster convergence rate.

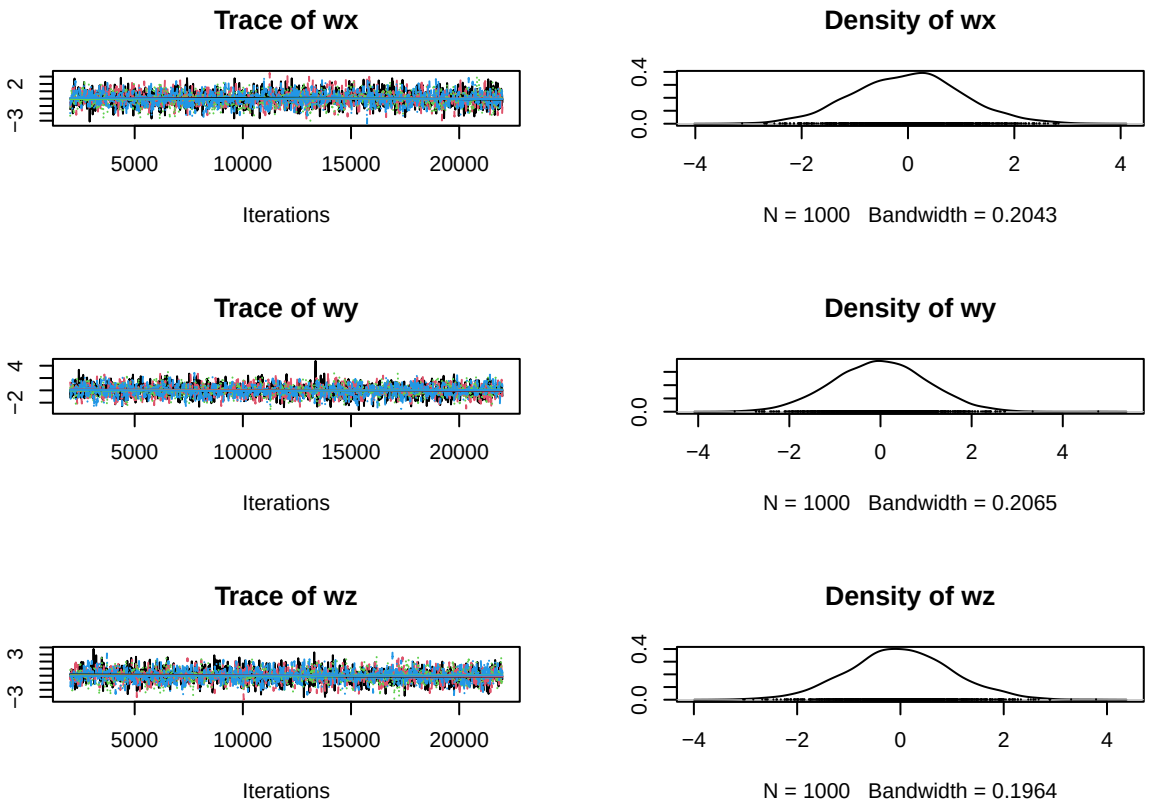
```
par(mfrow=c(2,2))
plot(jags_samples[,c("tau_pos","tau_vel")])
```




```
par(mfrow=c(2,2))
plot(jags_samples[,c("tau_o_pos","tau_o_vel")])
```

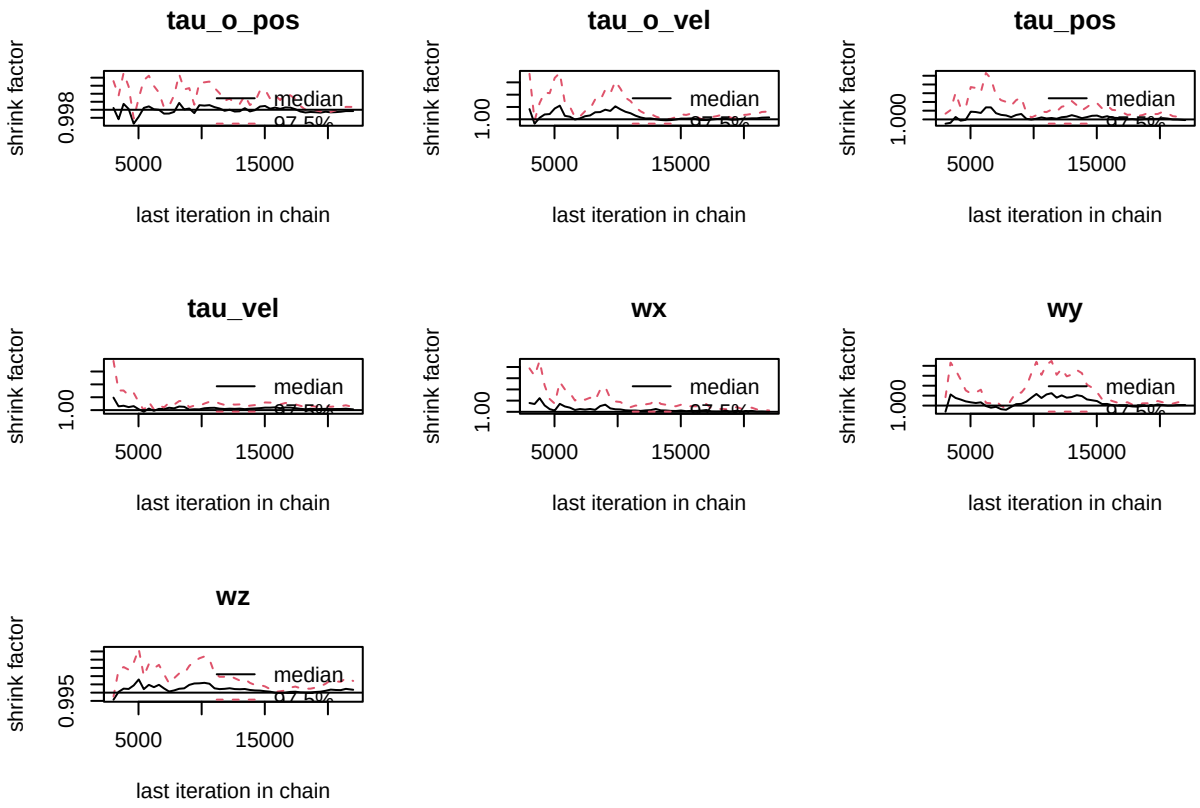



```
par(mfrow=c(2,2))
plot(jags_samples[,c("wx","wy","wz")])
```



The Gelman Rubic statistics and plots are shown in the figures below. They indicate that the BGR statistic value is around 1 reaffirming on the point emphasized by the above trace plots that the chains have converged already.

```
# Computing the Gelman-Rubin statistic
par(mfrow=c(2,2))
gelman.plot(jags_samples)
```




```
gelman.diag(jags_samples)
```

```
## Potential scale reduction factors:
##
##           Point est. Upper C.I.
## tau_o_pos          1       1.00
## tau_o_vel          1       1.01
## tau_pos            1       1.00
## tau_vel            1       1.00
## wx                 1       1.00
## wy                 1       1.00
## wz                 1       1.01
##
## Multivariate psrf
##
## 1
```

The below statistical measures indicate on the effective sample size present for each parameter of interest and we could clearly see from here that every parameter has an effective sample size **(ESS)** of 1000 , corresponding to good mixing and faster convergence of the chains.

```
# Computing the effective sample size
effectiveSize(jags_samples)
```

```
## tau_o_pos tau_o_vel   tau_pos   tau_vel        wx        wy        wz
##  4097.862  4198.174  4156.381  3711.626  4153.094  4000.000  3851.817
```

```r
#Evaluating the effective sample sizes of each parameter.
effectiveSize(jags_samples[[1]][,"tau_pos"])
```

```
##     var1
## 1101.715
```

```r
effectiveSize(jags_samples[[1]][,"tau_vel"])
```

```
## var1
## 1000
```

```r
effectiveSize(jags_samples[[1]][,"tau_o_pos"])
```

```
## var1
## 1000
```

```r
effectiveSize(jags_samples[[1]][,"tau_o_vel"])
```

```
## var1
## 1000
```

```r
effectiveSize(jags_samples[[1]][,"wx"])
```

```
## var1
## 1000
```

```r
effectiveSize(jags_samples[[1]][,"wy"])
```

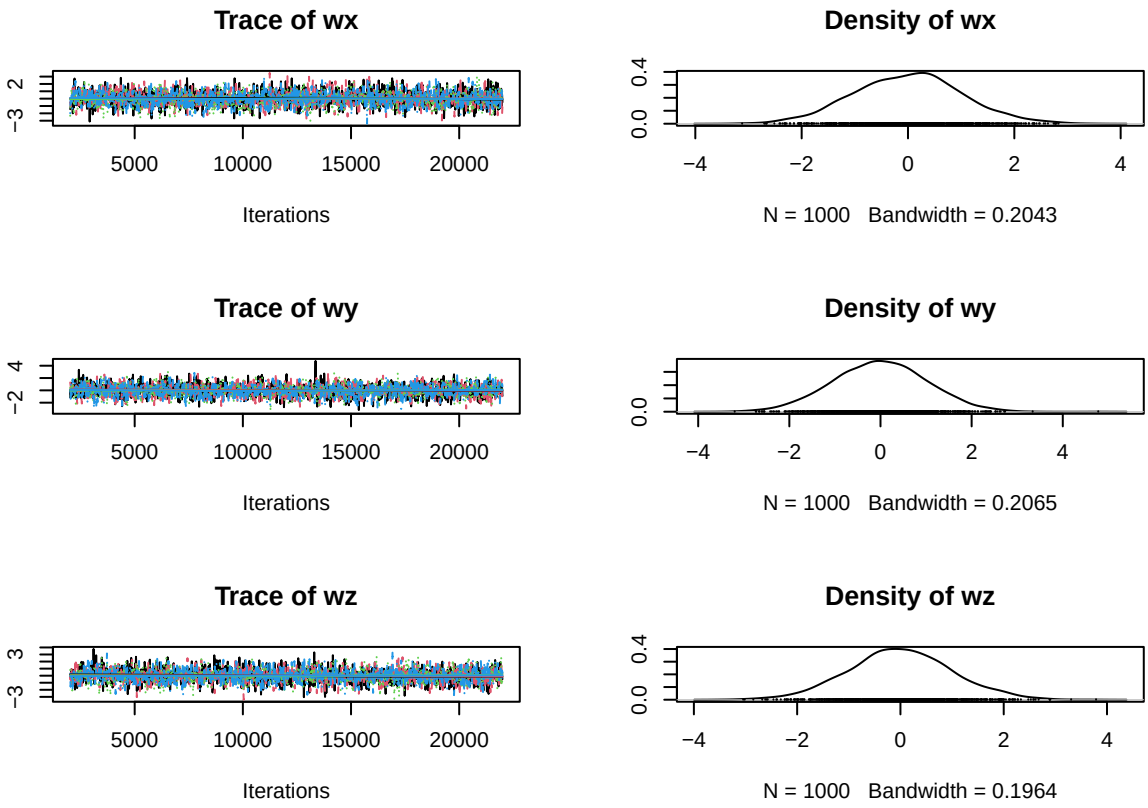
```
## var1
## 1000
```

```r
effectiveSize(jags_samples[[1]][,"wz"])
```

```
##     var1
## 913.0652
```

Below are the posterior density plots for the angular velocity components. Every posterior plot exhibits unimodal,symmetric distribution, centered around 0 with a standard deviation of 1 eliciting minimal uncertainity over the angular velocity components in all directions present.

```r
# Plotting the Posterior density plots for the angular velocity components.
plot(jags_samples[, c("wx", "wy", "wz")])
```



The Bayesian analysis of the model parameters yielded posterior distributions for various variables. Notably, the mean posterior value for tau_o_pos was estimated to be 47.18 with a standard deviation of 6.1714, suggesting a central tendency around this value. Similarly, tau_o_vel had a mean of 2.491 with a standard deviation of 0.3468. The parameter tau_pos exhibited a mean of 41.76 and a standard deviation of 5.7411. The parameters tau_vel, wx, wy, and wz showed mean values of 6.052, 0.0168, 0.000799, and 0.004596, respectively. These values, along with their associated standard deviations and quantiles, provide insights into the distribution and uncertainty of the model parameters. The quantiles further illustrate the range of probable values, enabling a nuanced interpretation of the Bayesian inference results.

```
# Printing out the overall summary of the model
summary(jags_samples)
```

```
##
## Iterations = 2020:22000
## Thinning interval = 20
## Number of chains = 4
## Sample size per chain = 1000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                Mean     SD Naive SE Time-series SE
## tau_o_pos 47.292817 6.1776 0.097677       0.096957
## tau_o_vel  2.499834 0.3492 0.005521       0.005398
## tau_pos   41.800913 5.6403 0.089180       0.087535
## tau_vel    6.044860 1.3117 0.020739       0.021603
```

```
## wx         0.006053 1.0123 0.016006       0.015744
## wy        -0.007993 1.0235 0.016183       0.016187
## wz        -0.002984 0.9983 0.015784       0.016107
##
## 2. Quantiles for each variable:
##
##              2.5%     25%      50%     75%  97.5%
## tau_o_pos 35.933 43.1367 46.92024 51.2853 59.987
## tau_o_vel  1.872  2.2605  2.47777  2.7235  3.259
## tau_pos   31.312 37.8347 41.55287 45.5294 53.481
## tau_vel    3.825  5.0918  5.92206  6.8599  8.972
## wx        -1.986 -0.6952  0.02557  0.6647  2.004
## wy        -2.023 -0.7046 -0.01048  0.6800  1.950
## wz        -1.981 -0.6506 -0.01010  0.6538  1.990
```

QUESTION 3 - PERFORMING POSTERIOR PREDICTIVE CHECKS

Posterior Predictive checks are implemented here, where the same JAGS model is altered to hold same values under different names instantiated like **xrep,yrep,zrep,vxrep,vyrep,vzrep** and now we try to sample based on these new parameters of interest.

```
library("fBasics")
no_of_iterations <- 500000
resthin2=coda.samples(jags_model,variable.names=c("xrep","yrep","zrep","vxrep","vyrep","vzrep"),n.iter=
```

In this below block of code, where we fetch out different matrices for the each parameter of interest from the newly sampled model. Each matrix now holds the corresponding values for every parameter of interest from time step 1 to 60.

```
xrep <- yrep <- zrep <- vxrep <- vyrep <- vzrep <- matrix(NA, nrow = no_of_iterations, ncol = (n-1))

for(i in 1:(n-1)) {

  xrep[,i] <- resthin2[[1]][,paste0('xrep[',i+1,']')]
  yrep[,i] <- resthin2[[1]][,paste0('yrep[',i+1,']')]
  zrep[,i] <- resthin2[[1]][,paste0('zrep[',i+1,']')]
  vxrep[,i] <- resthin2[[1]][,paste0('vxrep[',i+1,']')]
  vyrep[,i] <- resthin2[[1]][,paste0('vyrep[',i+1,']')]
  vzrep[,i] <- resthin2[[1]][,paste0('vzrep[',i+1,']')]

}
```

So here in this block of code, where I fetch out all the five different statistical measures namely **max,median,skewness,kurtosis** for all the three different position components present. And according to our problem statement, for analyzing the movement of a ping pong ball, the **median and mean** is calculated to understand the central tendency of the data, *skewness* to check out for any asymmetry and *kurtosis* to assess the appropriate tail behaviour, whereas we do not consider *min* measure since the ping pong ball movement would get influenced by a number of external factors.

```
require(fBasics)
#statistics of interest in this case (and different from the ones included in the model)

xrepmax=apply(xrep,1,max)
```

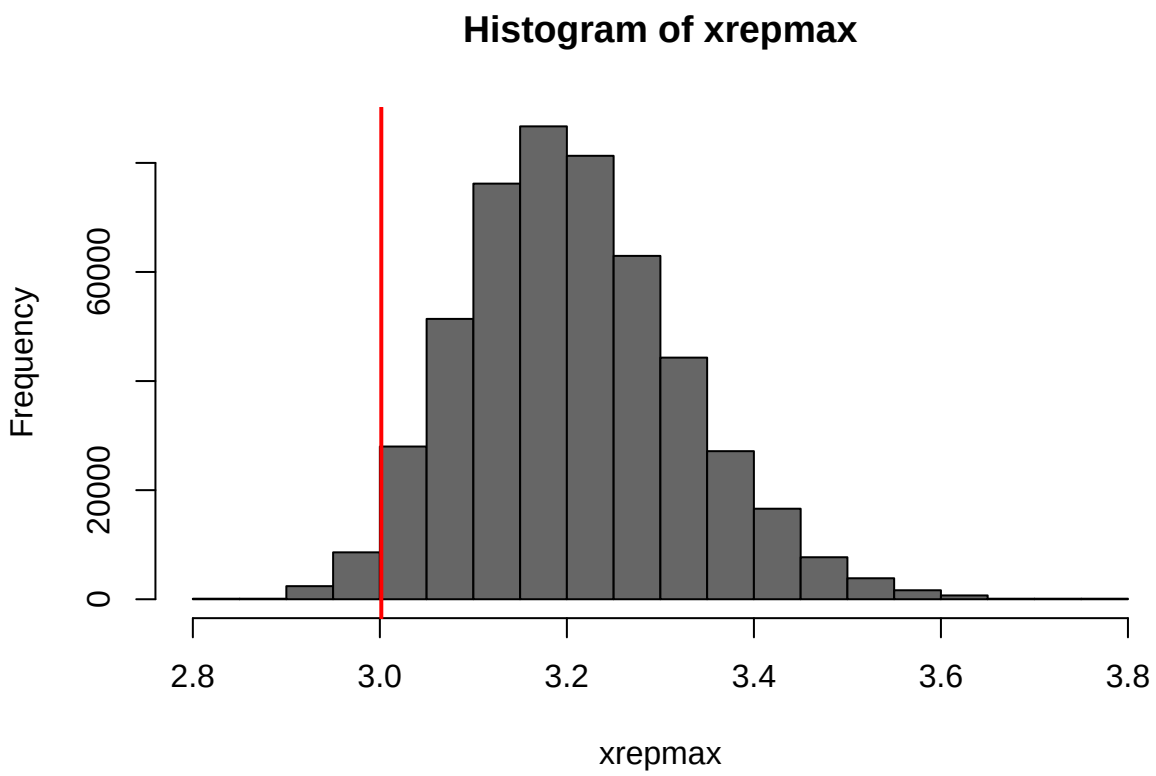
```
xrepmedian=apply(xrep,1,median)
xrepskewness=apply(xrep,1,skewness)
xrepkurtosis=apply(xrep,1,kurtosis)


yrepmax=apply(yrep,1,max)
yrepmedian=apply(yrep,1,median)
yrepskewness=apply(yrep,1,skewness)
yrepkurtosis=apply(yrep,1,kurtosis)


zrepmax=apply(zrep,1,max)
zrepmedian=apply(zrep,1,median)
zrepskewness=apply(zrep,1,skewness)
zrepkurtosis=apply(zrep,1,kurtosis)
```

In this block of code , where all the five different statistical measures is calculated on all the velocity components present.

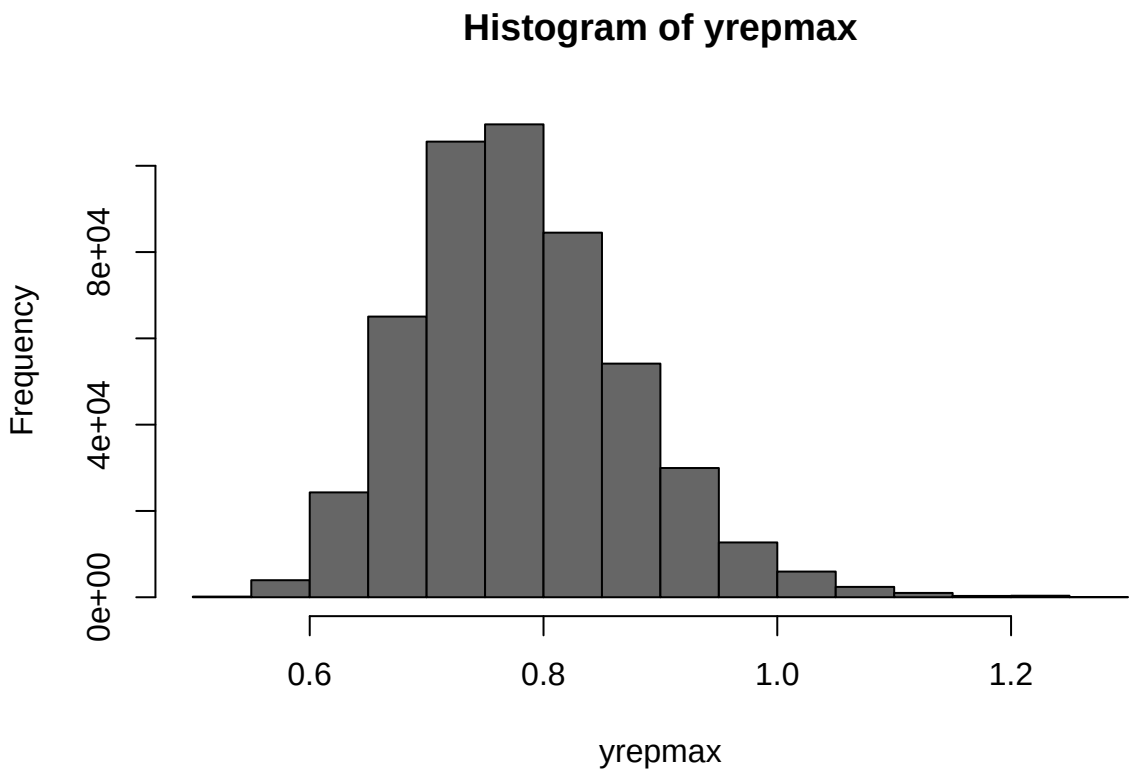
```
vxrepmax=apply(vxrep,1,max)
vxrepmedian=apply(vxrep,1,median)
vxrepskewness=apply(vxrep,1,skewness)
vxrepkurtosis=apply(vxrep,1,kurtosis)


vyrepmax=apply(vyrep,1,max)
vyrepmedian=apply(vyrep,1,median)
vyrepskewness=apply(vyrep,1,skewness)
vyrepkurtosis=apply(vyrep,1,kurtosis)


vzrepmax=apply(vzrep,1,max)
vzrepmedian=apply(vzrep,1,median)
vzrepskewness=apply(vzrep,1,skewness)
vzrepkurtosis=apply(vzrep,1,kurtosis)
```

Maximum posterior predictive checks are plotted for the replicated data

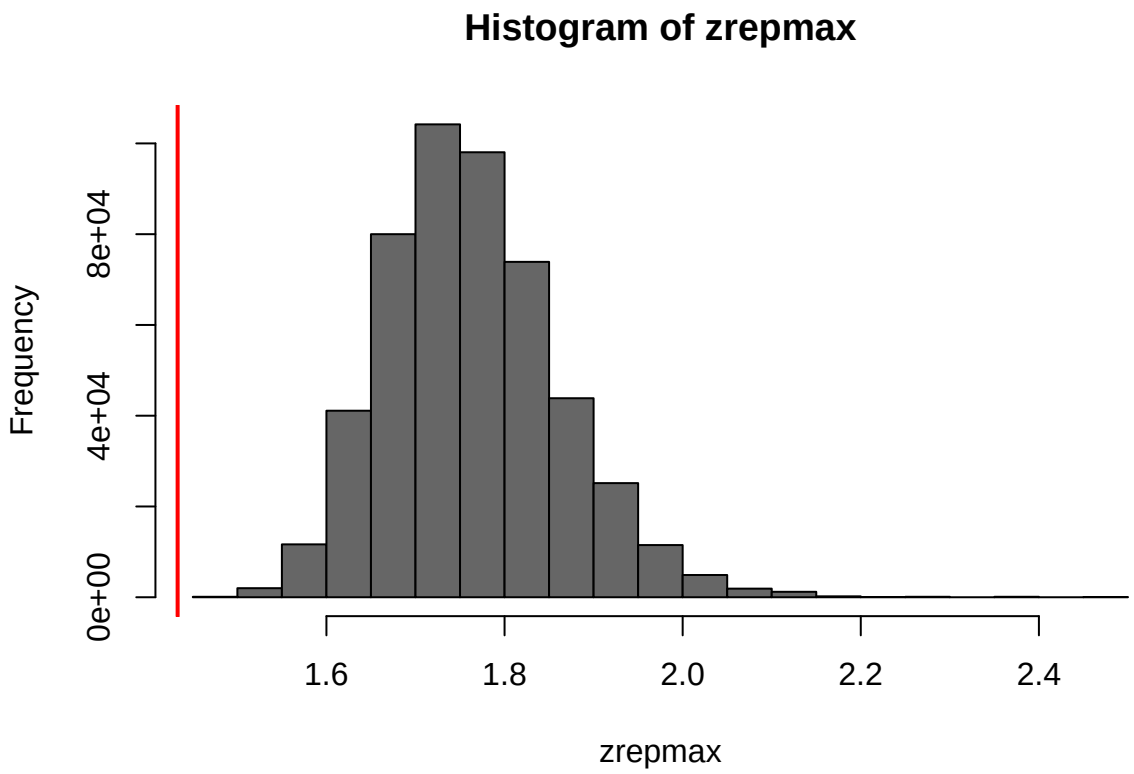
```
#Predictive checks using replicated data - maximum

hist(xrepmax,col="gray40")
abline(v=max(xo),col="red",lwd=2)
```



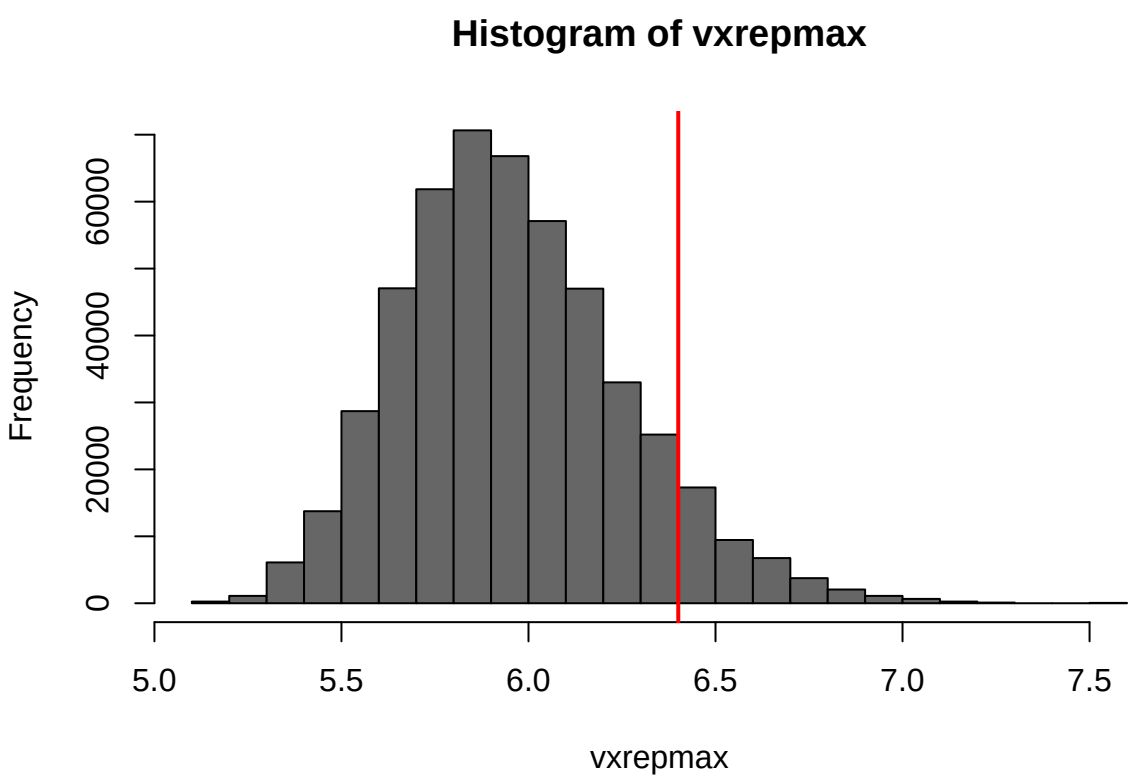

```
hist(yrepmax,col="gray40")
abline(v=max(yo),col="red",lwd=2)
```



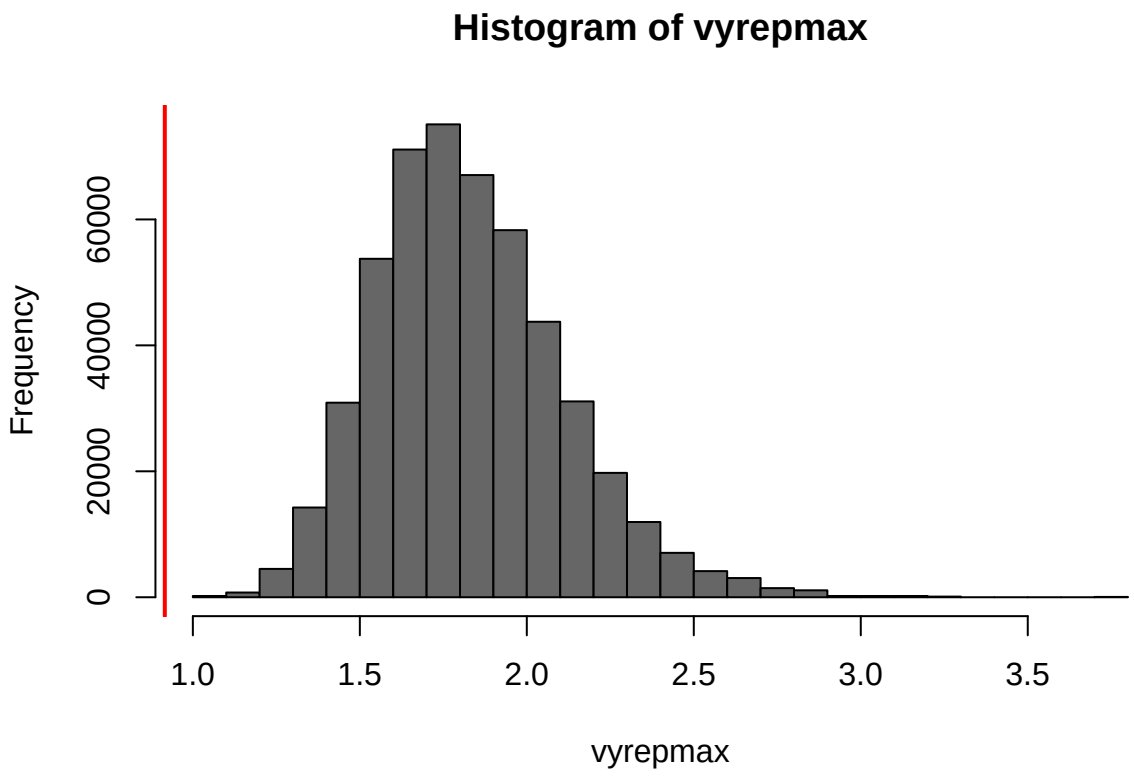

```
hist(zrepmax,col="gray40")
abline(v=max(zo),col="red",lwd=2)
```



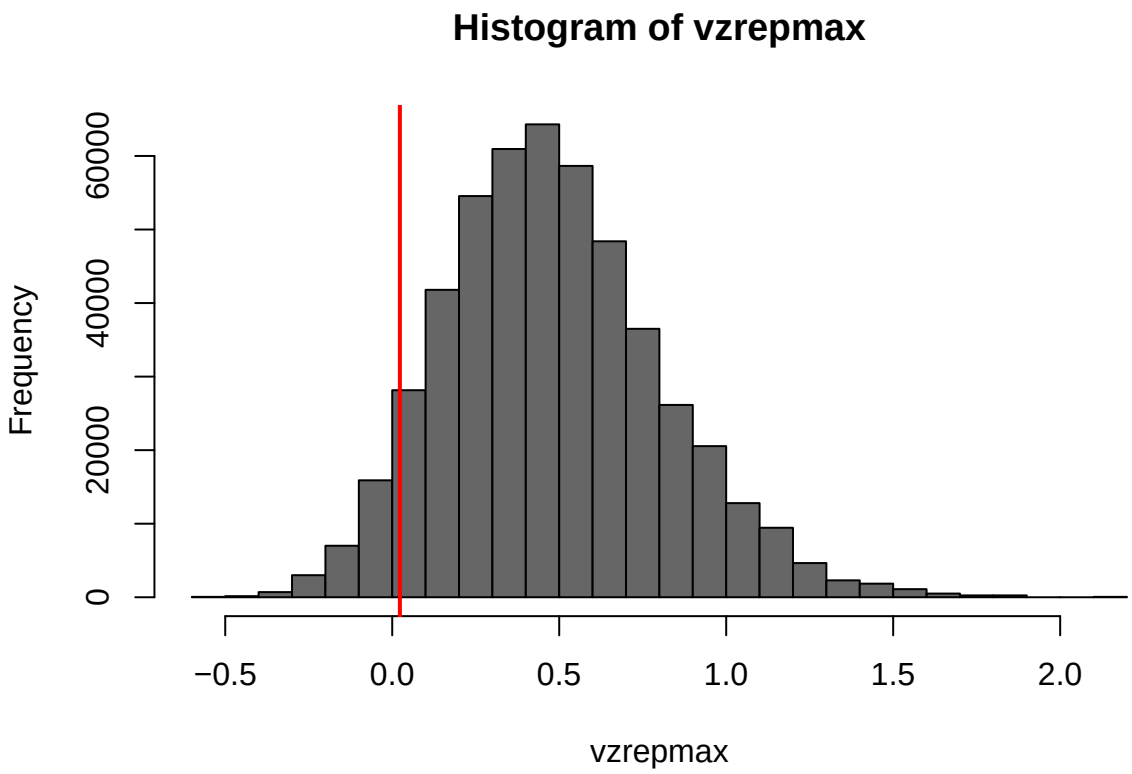

```
hist(vxrepmax,col="gray40")
abline(v=max(vxo),col="red",lwd=2)
```




```
hist(vyrepmax,col="gray40")
abline(v=max(vyo),col="red",lwd=2)
```

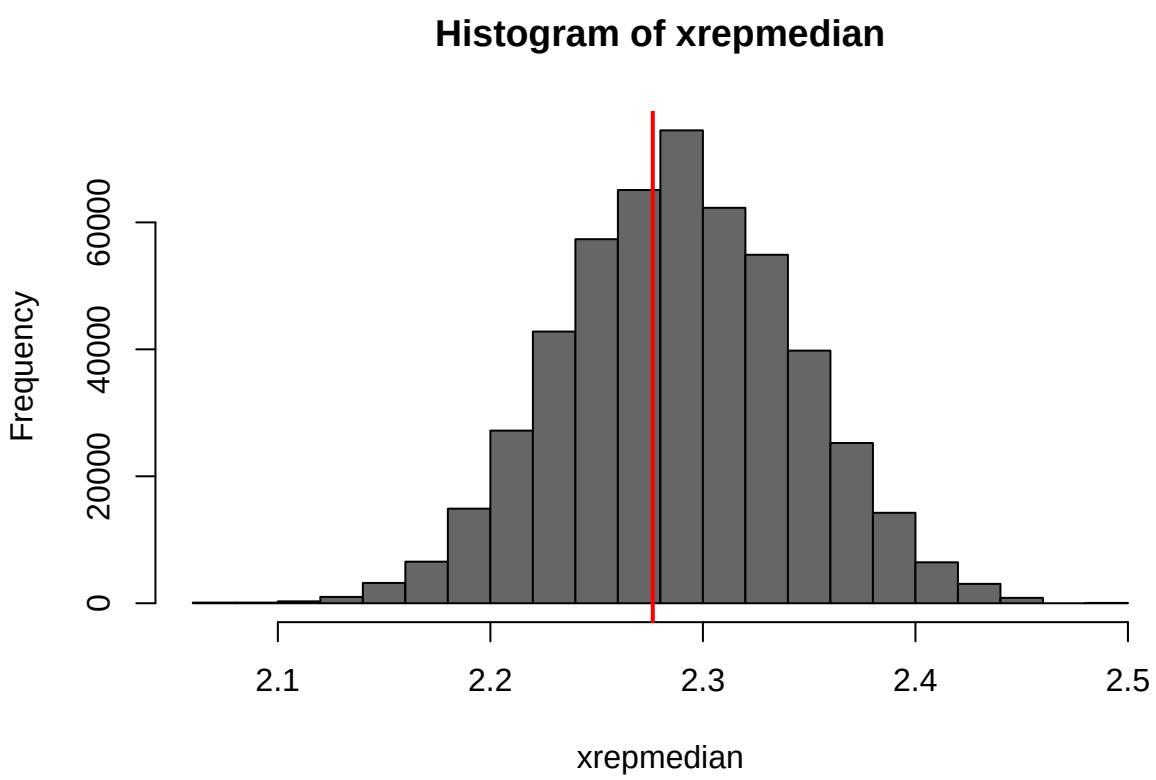



```
hist(vzrepmax,col="gray40")
abline(v=max(vzo),col="red",lwd=2)
```

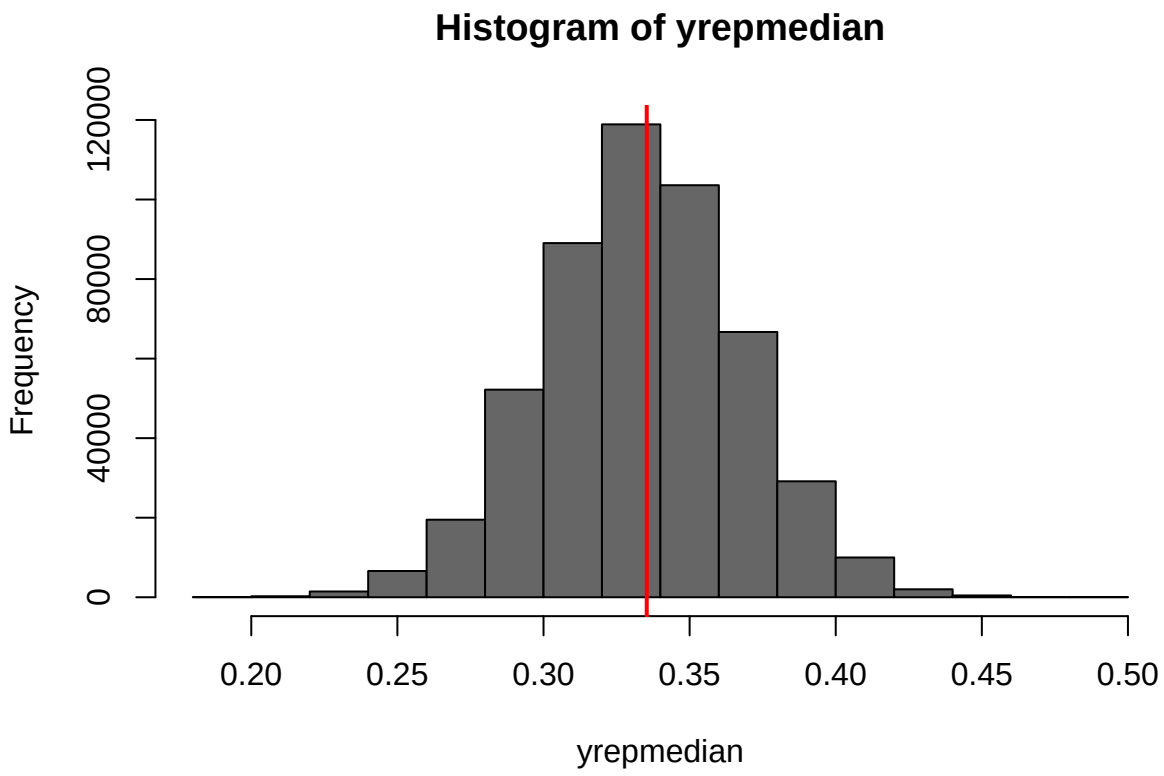


Median posterior predictive checks are plotted for the replicated data

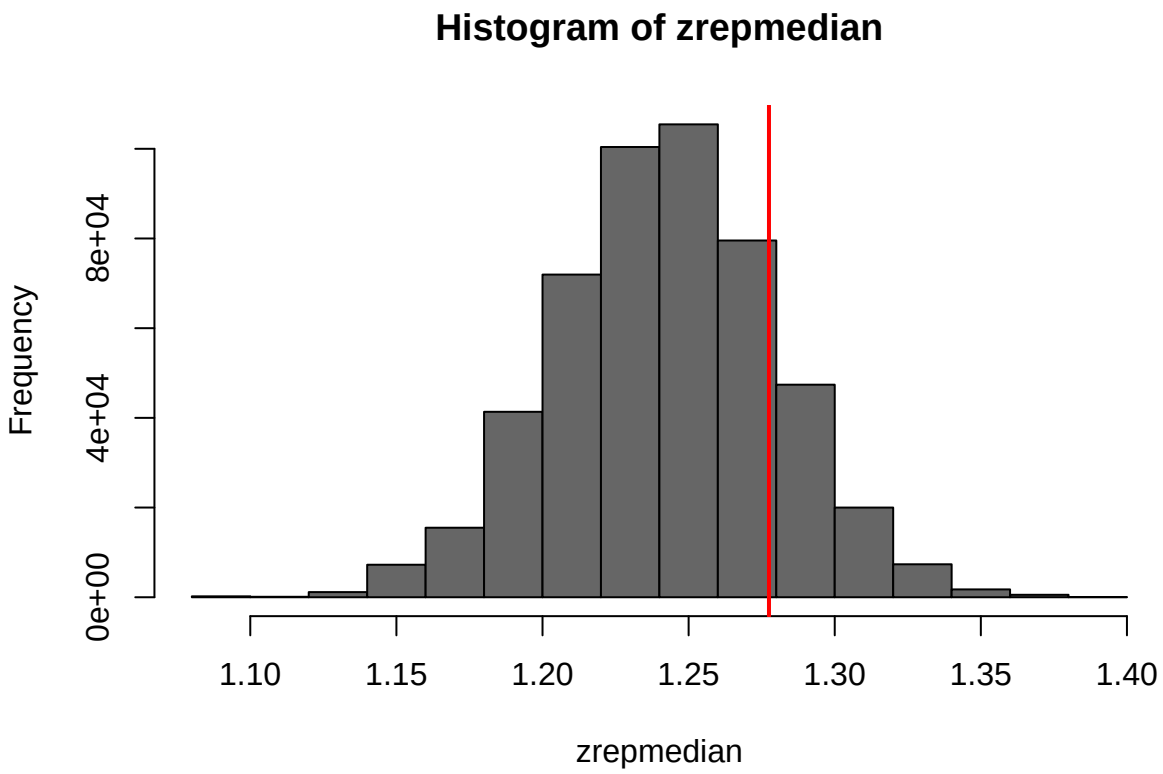
```
#Predictive checks using replicated data - median

hist(xrepmedian,col="gray40")
abline(v=median(xo),col="red",lwd=2)
```



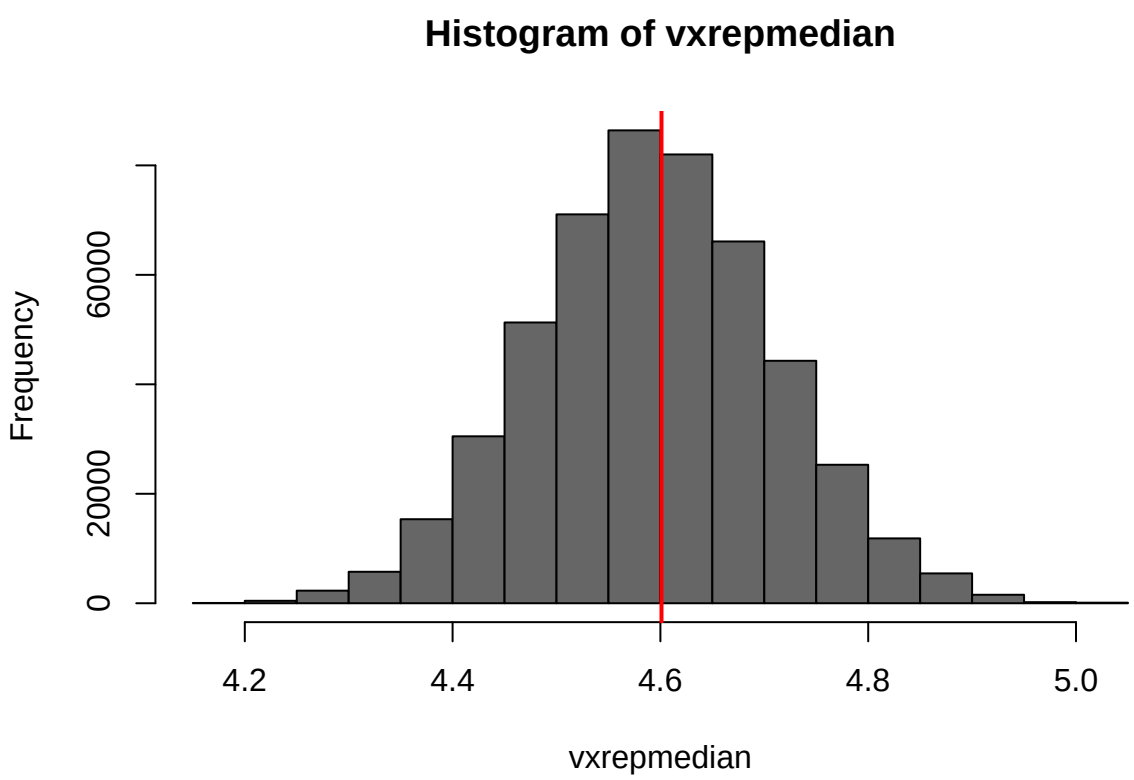

```
hist(yrepmedian,col="gray40")
abline(v=median(yo),col="red",lwd=2)
```



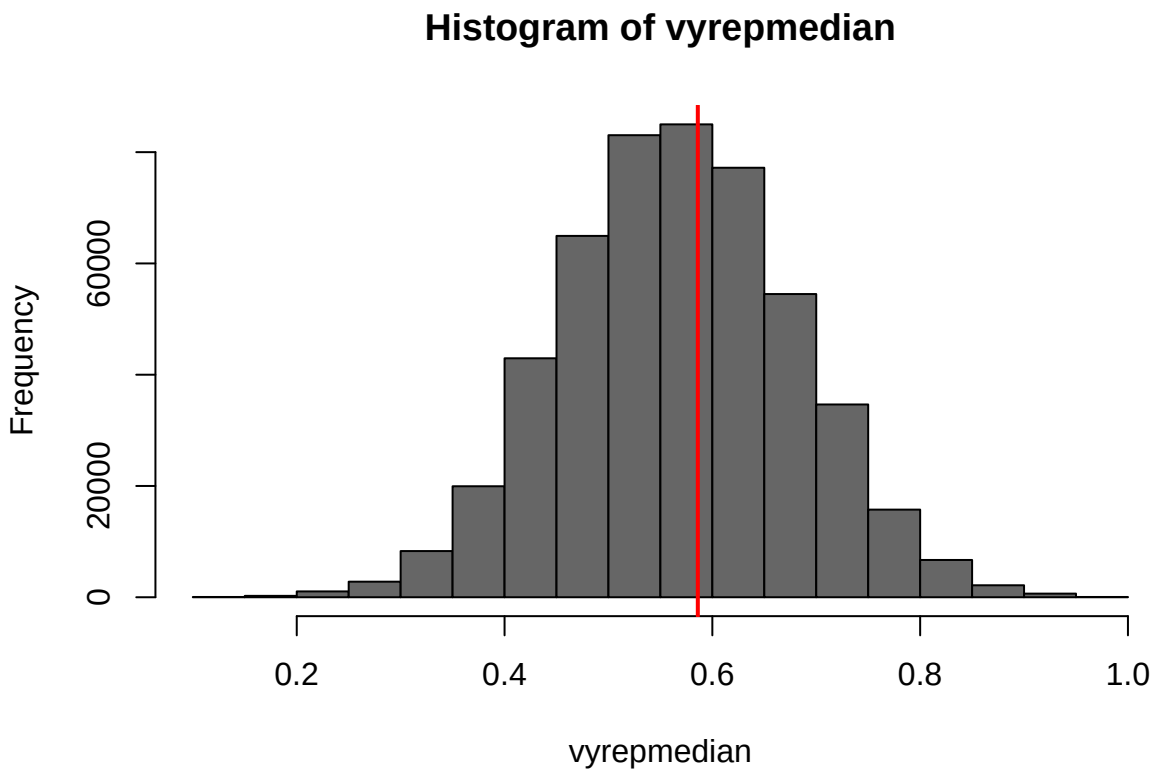

```
hist(zrepmedian,col="gray40")
abline(v=median(zo),col="red",lwd=2)
```



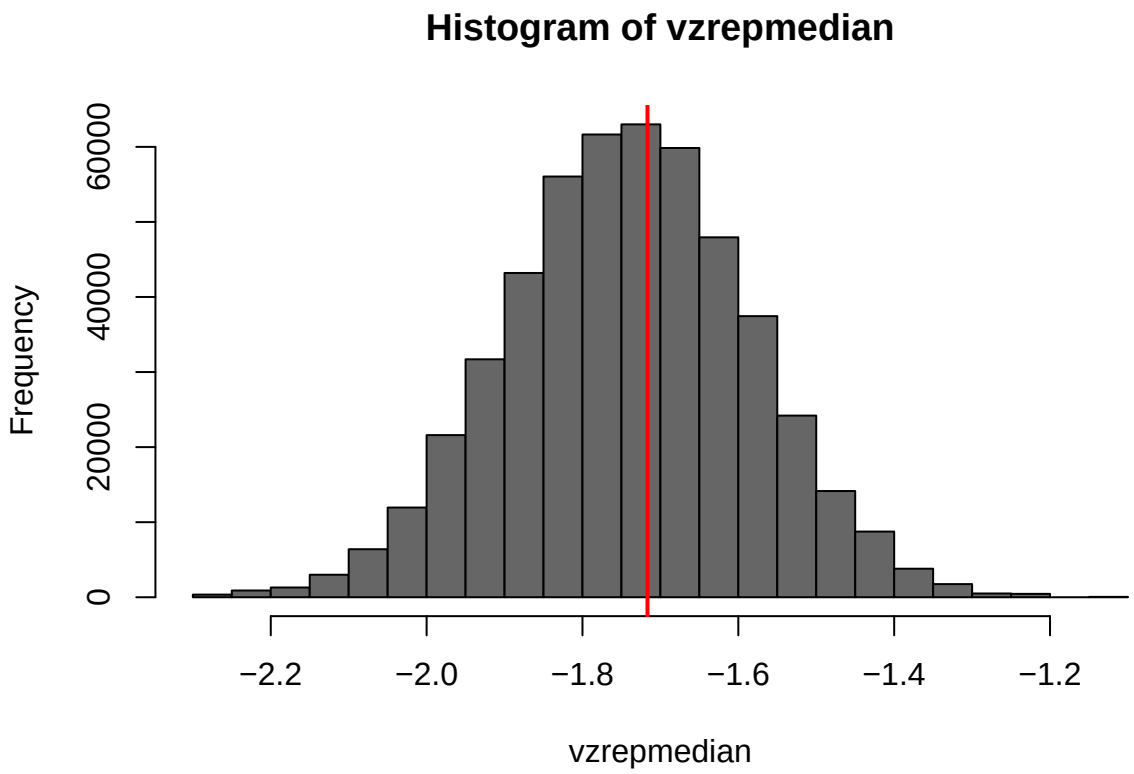

```
hist(vxrepmedian,col="gray40")
abline(v=median(vxo),col="red",lwd=2)
```




```
hist(vyrepmedian,col="gray40")
abline(v=median(vyo),col="red",lwd=2)
```

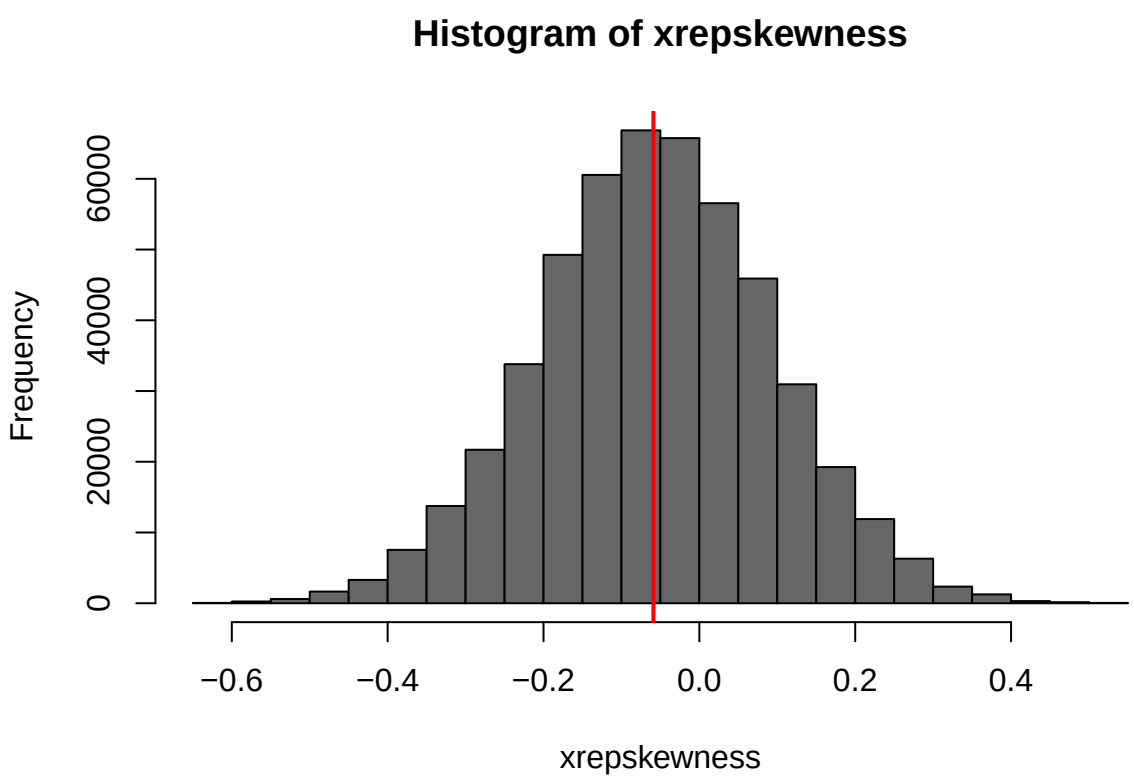



```
hist(vzrepmedian,col="gray40")
abline(v=median(vzo),col="red",lwd=2)
```

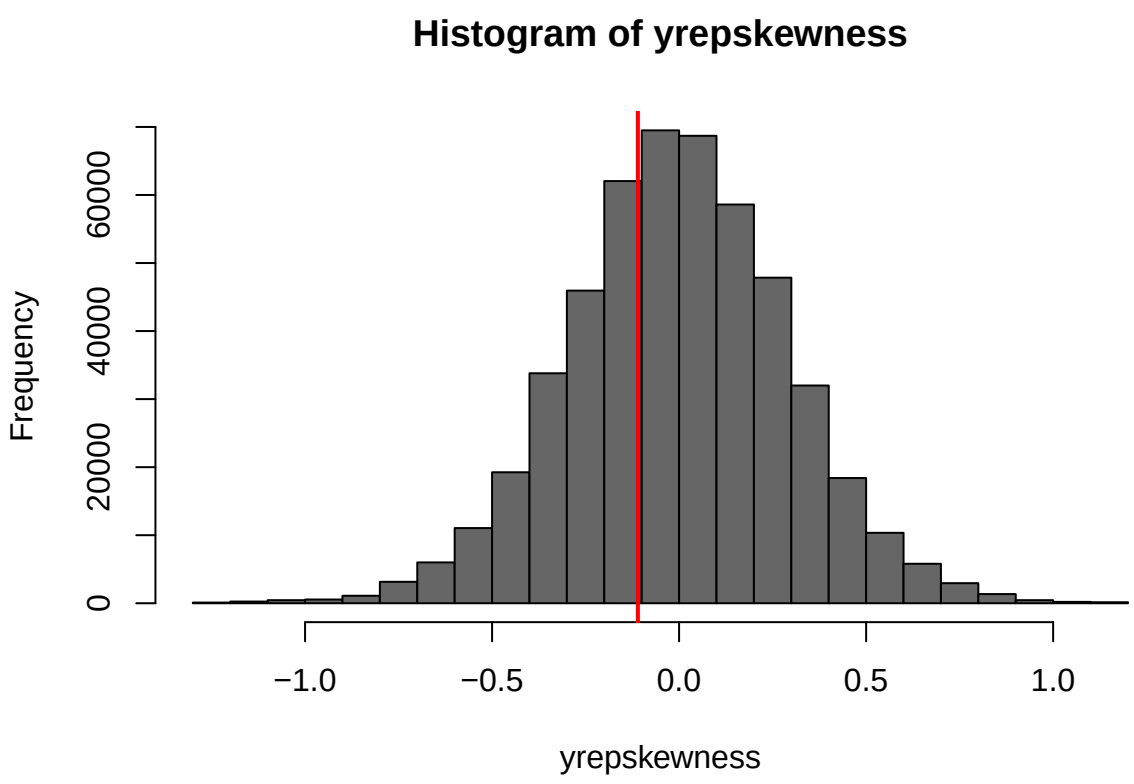


Skewness posterior predictive checks are plotted for the replicated data

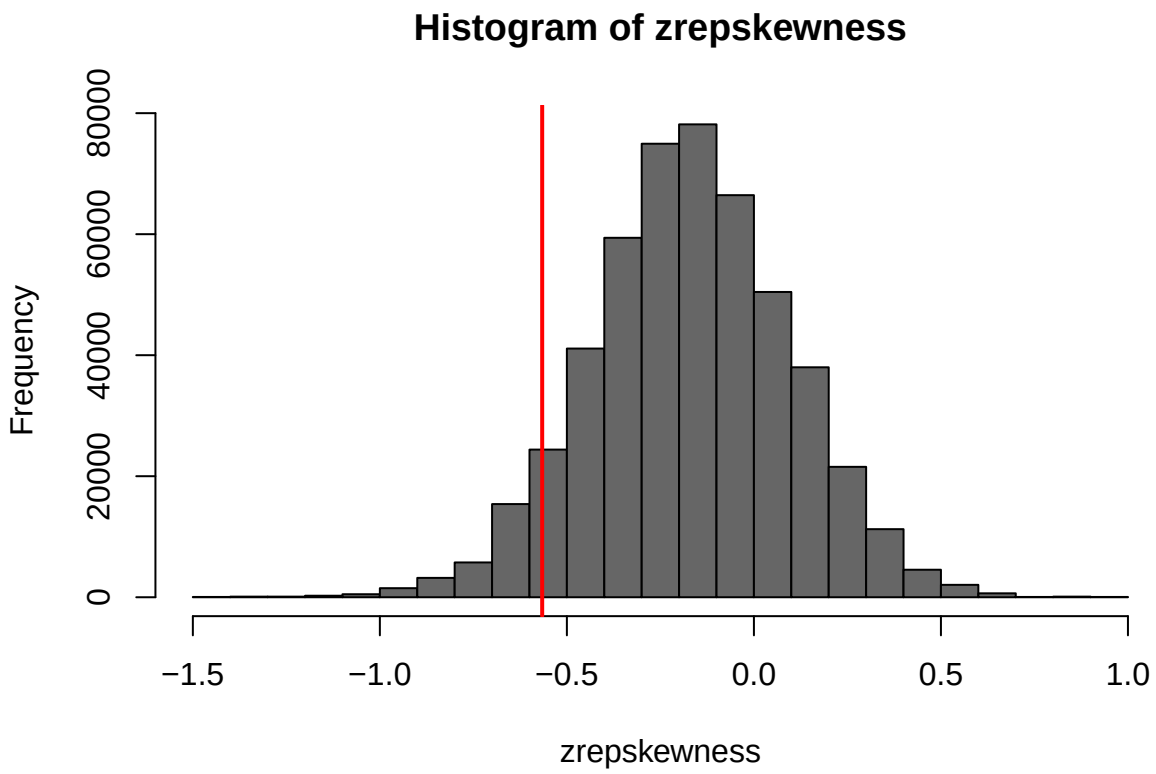
```
#Predictive checks using replicated data - skewness

hist(xrepskewness,col="gray40")
abline(v=skewness(xo),col="red",lwd=2)
```



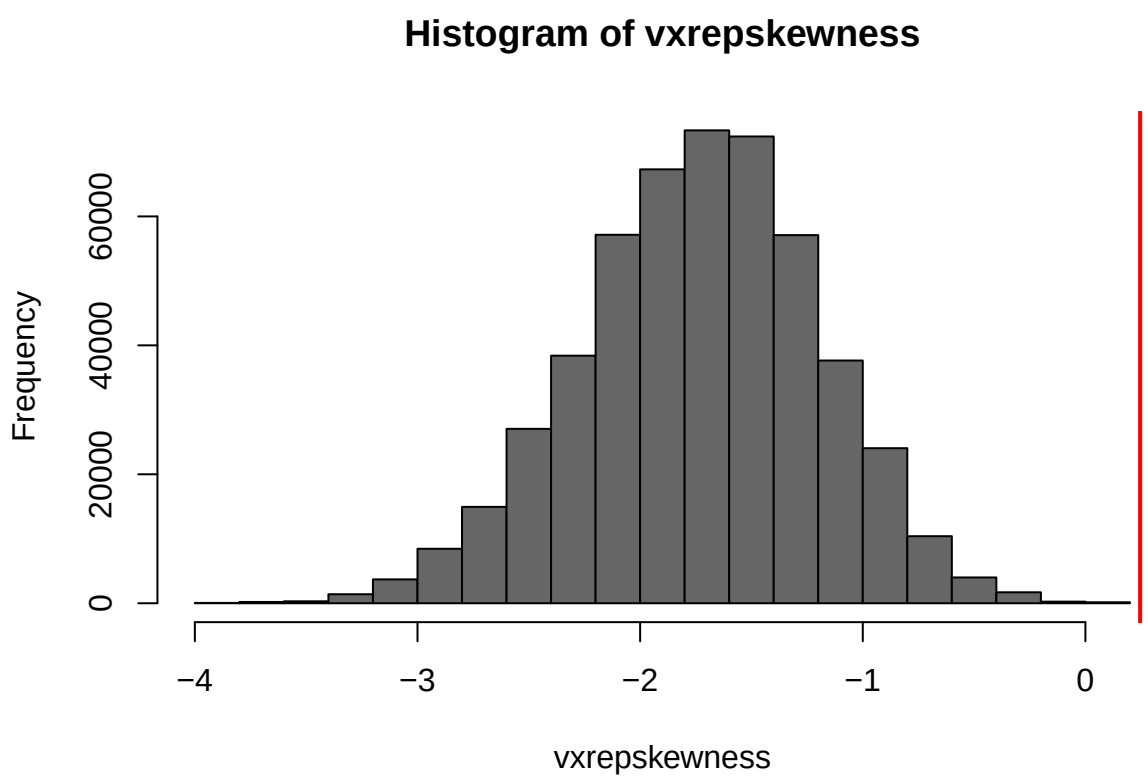

```
hist(yrepskewness,col="gray40")
abline(v=skewness(yo),col="red",lwd=2)
```



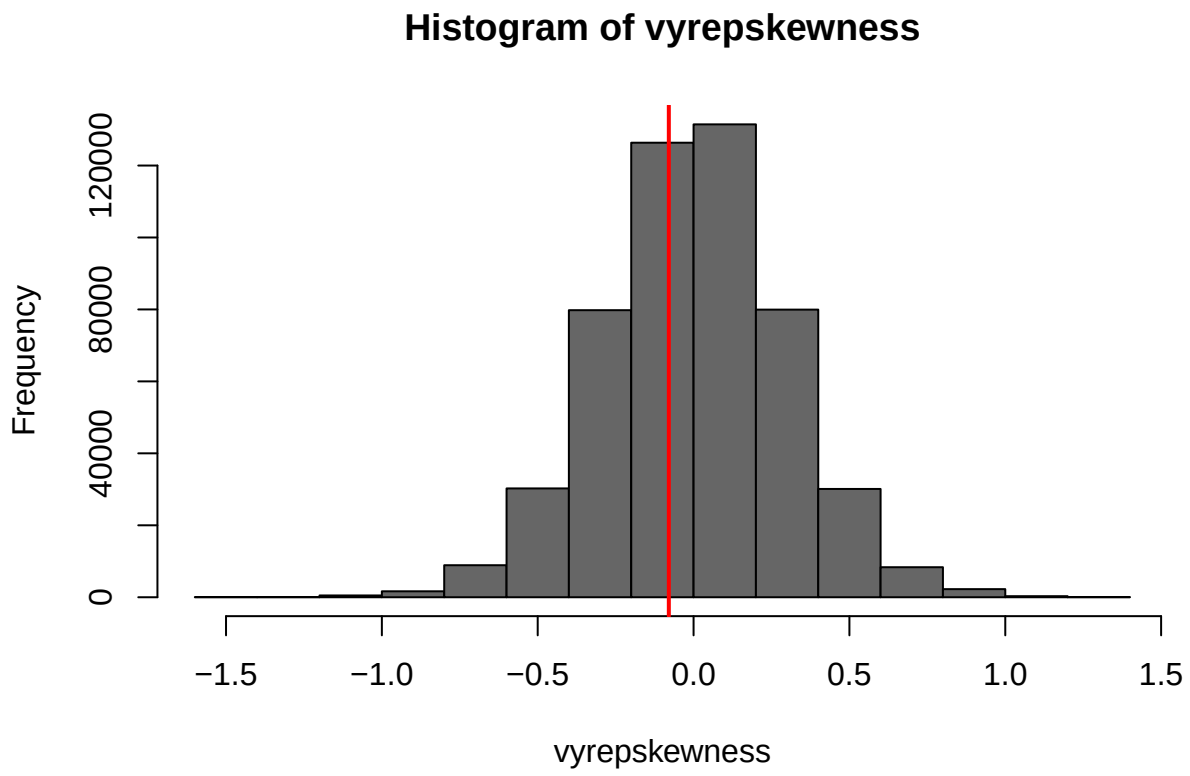

```
hist(zrepskewness,col="gray40")
abline(v=skewness(zo),col="red",lwd=2)
```



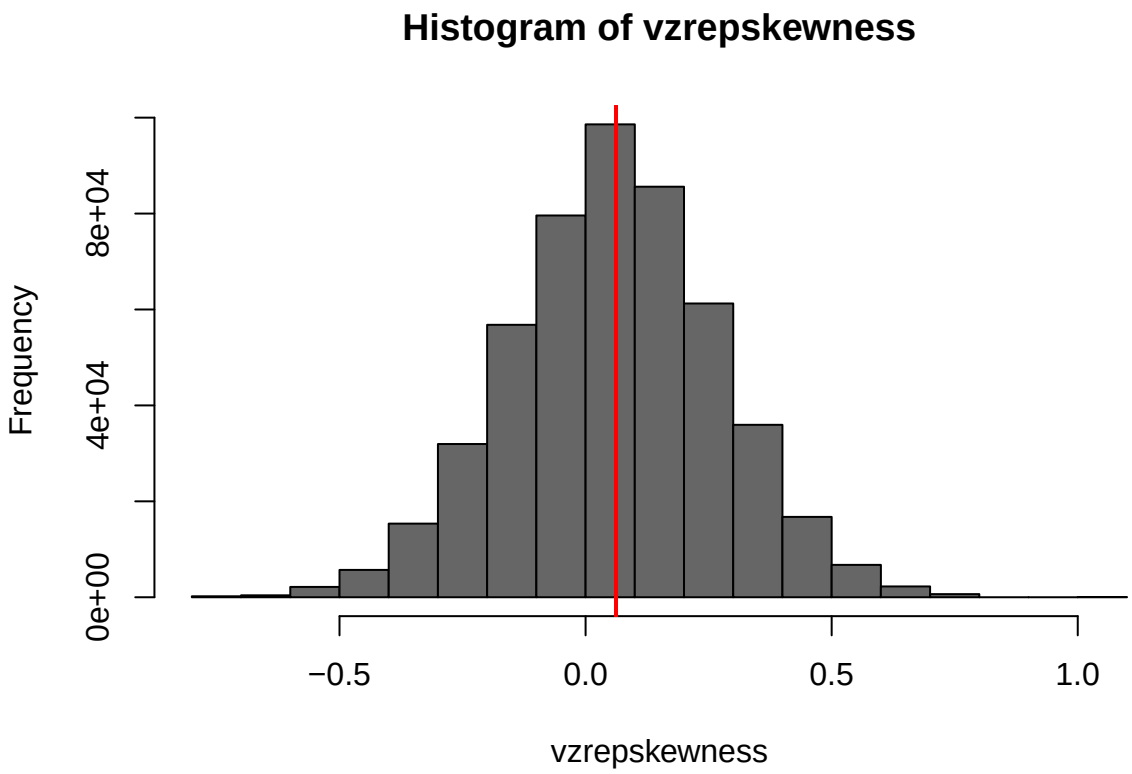

```
hist(vxrepskewness,col="gray40")
abline(v=skewness(vxo),col="red",lwd=2)
```




```
hist(vyrepskewness,col="gray40")
abline(v=skewness(vyo),col="red",lwd=2)
```

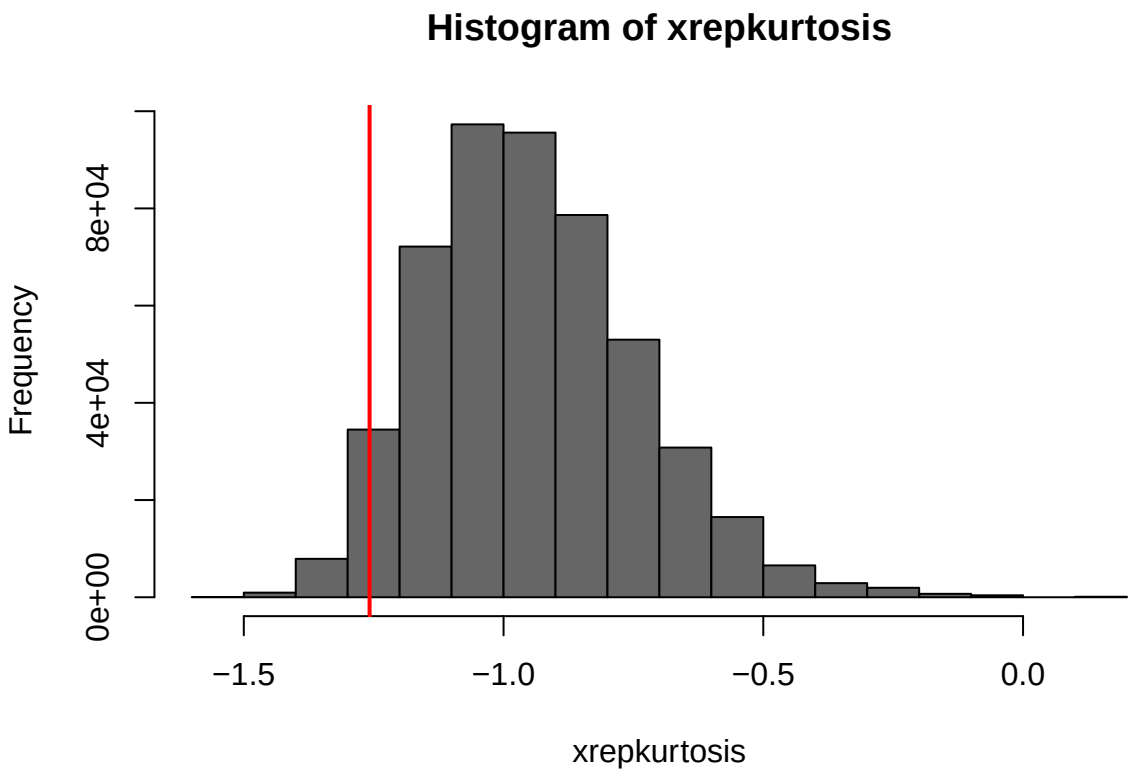



```
hist(vzrepskewness,col="gray40")
abline(v=skewness(vzo),col="red",lwd=2)
```

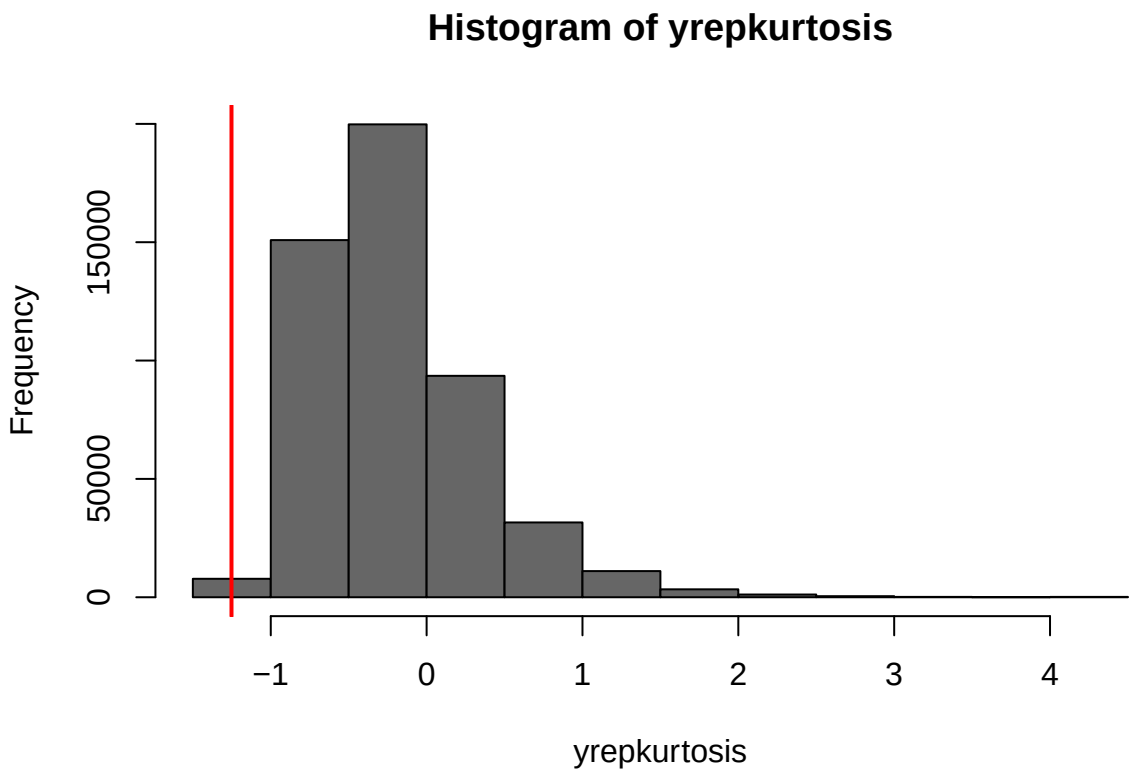


Kurtosis posterior predictive checks are plotted for the replicated data

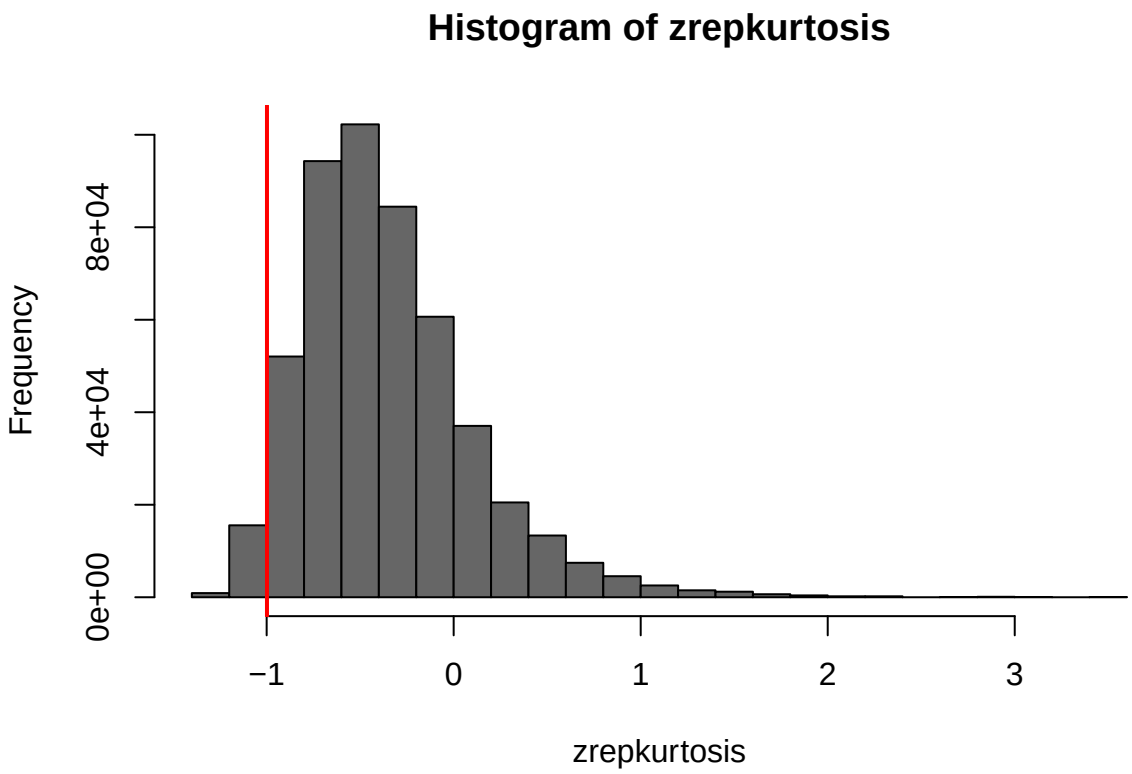
```
#Predictive checks using replicated data - kurtosis

hist(xrepkurtosis,col="gray40")
abline(v=kurtosis(xo),col="red",lwd=2)
```



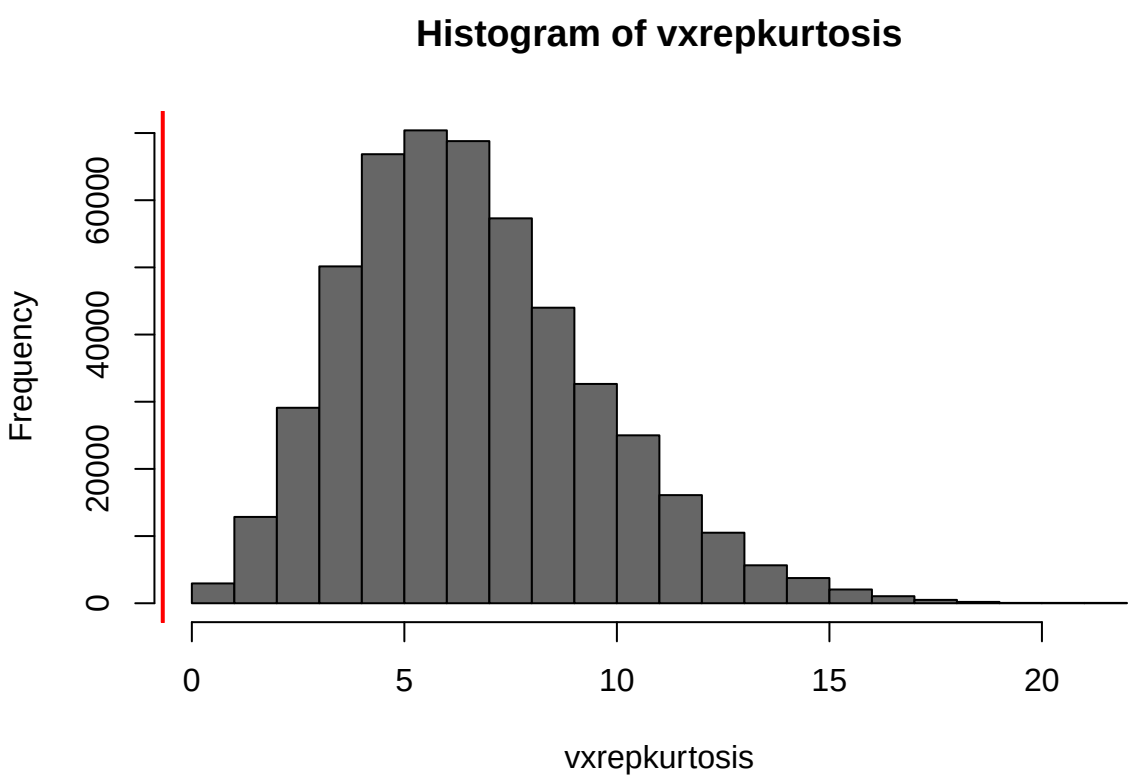

```
hist(yrepkurtosis,col="gray40")
abline(v=kurtosis(yo),col="red",lwd=2)
```



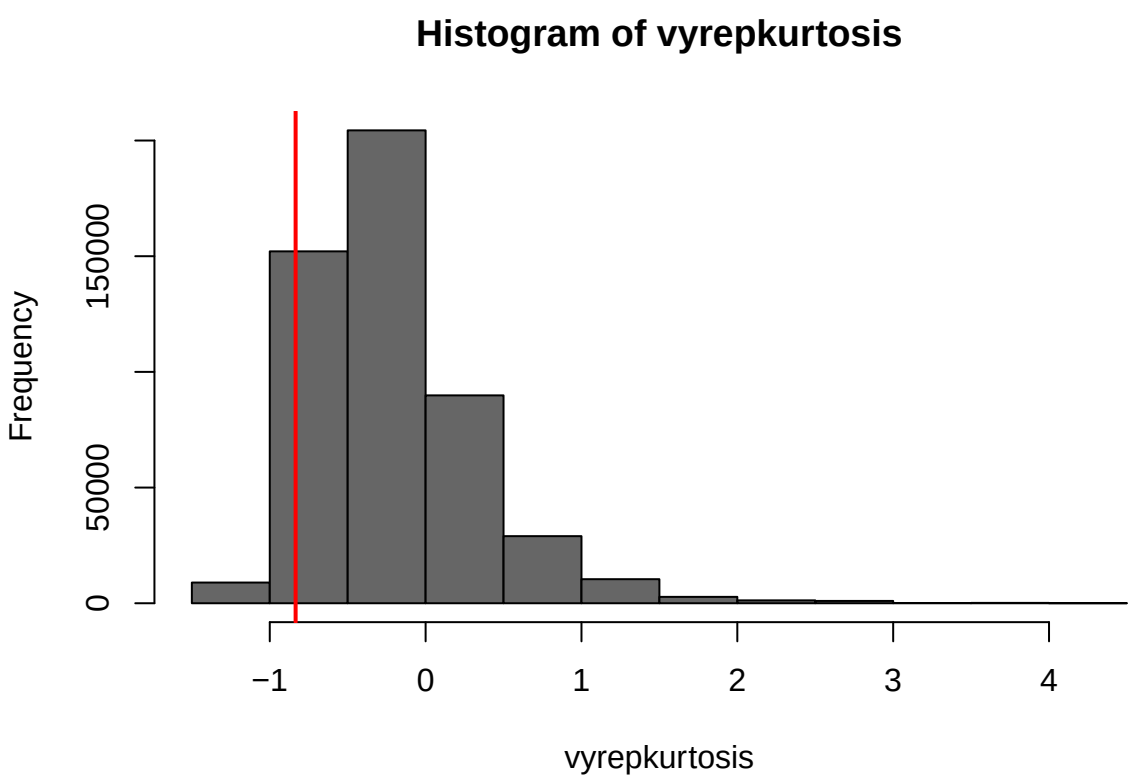

```
hist(zrepkurtosis,col="gray40")
abline(v=kurtosis(zo),col="red",lwd=2)
```



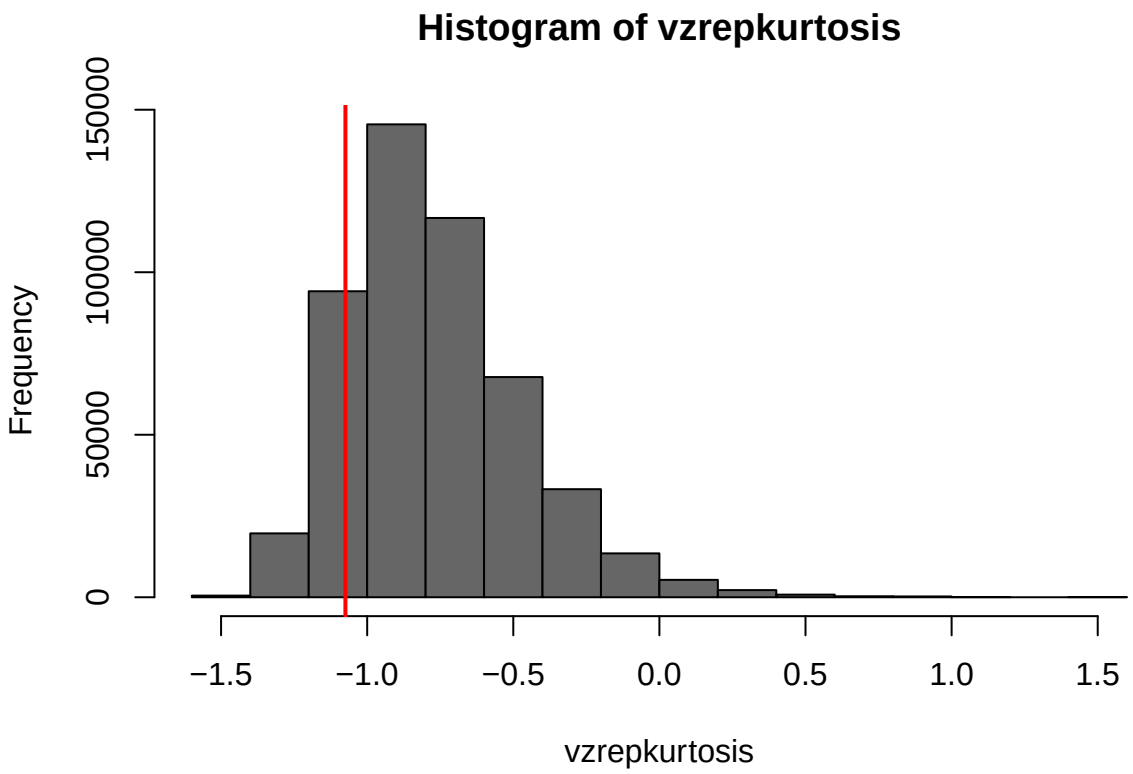

```
hist(vxrepkurtosis,col="gray40")
abline(v=kurtosis(vxo),col="red",lwd=2)
```




```
hist(vyrepkurtosis,col="gray40")
abline(v=kurtosis(vyo),col="red",lwd=2)
```

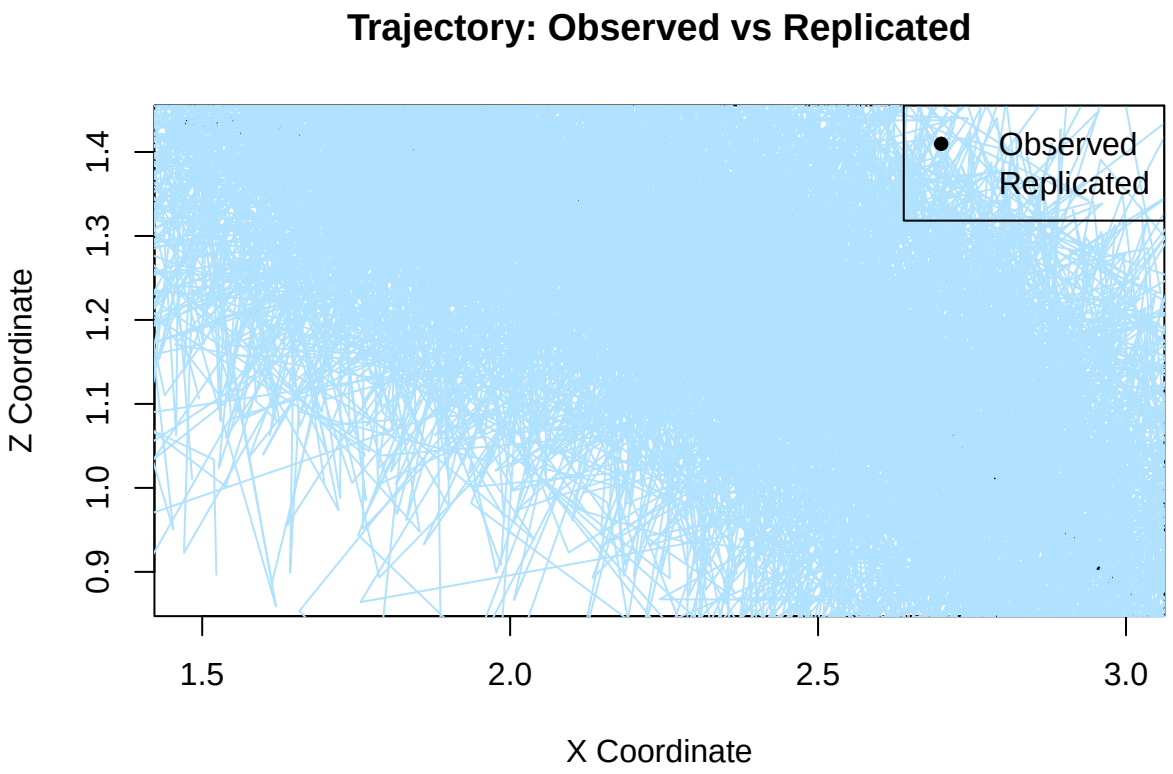



```
hist(vzrepkurtosis,col="gray40")
abline(v=kurtosis(vzo),col="red",lwd=2)
```



Plot of the x coordinate of the trajectory against the z coordinate

```
# Plotting the observed trajectory
plot(xo, zo, col = "black", pch = 16, main = "Trajectory: Observed vs Replicated", xlab = "X Coordinate

# Plotting with 100 such posterior replicates
for (i in 1:100) {
  lines(xrep[i, ], zrep[i, ], col = "lightskyblue1", lty = 1)
}

# Adding the legend for the same
legend("topright", legend = c("Observed", "Replicated"), col = c("black", "lightskyblue1"), pch = c(16,
```



QUESTION 4 - COMPUTING THE BAYESIAN PREDICTIVE ACCURACY

Loading the new data with points included for the next six time steps as well, saving the last six values before masking them with **NA**. Thereon, modelling through the JAGS model, fetching out the desired number of samples. Now, fetching out the last six column entries of every parameter of interest, computing the posterior predictive mean of all the position and velocity components, now thereby computing the euclidean distance measure at every time step , where each time step has two measures - one for the position (x,y,z) and other one for the velocity (vx,vy,vz), therefore printing out the corresponding values.

The Euclidean distance measure on an average for every time step seems to be **very low** for both the position and velocity components indicative of a good model, also emphasizes on the fact that the model has a higher bayesian predictive accuracy with low errors.

```
n=66;
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)];

x_observed = xo[(n-5):n]
```

```
y_observed = yo[(n-5):n]
z_observed = zo[(n-5):n]
vx_observed = vxo[(n-5):n]
vy_observed = vyo[(n-5):n]
vz_observed = vzo[(n-5):n]

### masking the values

xo[(n-5):n] <- NA
yo[(n-5):n] <- NA
zo[(n-5):n] <- NA
vxo[(n-5):n] <- NA
vyo[(n-5):n] <- NA
vzo[(n-5):n] <- NA

g <- 9.827
km <- 1/(2*0.0027)*1*1.29*0.001256*0.02
kd <- -1/(2*0.0027)*0.456*1.29*0.001256

jags_data = list (
       n = n,
       xo = xo,
       yo = yo,
       zo = zo,
       vxo = vxo,
       vyo = vyo,
       vzo = vzo,
       g = g,
       km = km,
       kd = kd,
       delta = diff(T)
       )

jags_model = jags.model(textConnection(model_string_jags), data = jags_data, n.chains = 1)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 360
##    Unobserved stochastic nodes: 823
##    Total graph size: 3596
##
## Initializing model
```

```
update(jags_model,1000,progress.bar="none")

jags_samples <- coda.samples(jags_model, variable.names = c("x", "y", "z", "vx", "vy", "vz"), n.iter = 5

x_selected_cols <- c("x[61]","x[62]","x[63]","x[64]","x[65]","x[66]")
y_selected_cols <- c("y[61]","y[62]","y[63]","y[64]","y[65]","y[66]")
z_selected_cols <- c("z[61]","z[62]","z[63]","z[64]","z[65]","z[66]")
vx_selected_cols <- c("vx[61]","vx[62]","vx[63]","vx[64]","vx[65]","vx[66]")
```

```r
vy_selected_cols <- c("vy[61]","vy[62]","vy[63]","vy[64]","vy[65]","vy[66]")
vz_selected_cols <- c("vz[61]","vz[62]","vz[63]","vz[64]","vz[65]","vz[66]")

# Fetching out the appropriate columns and computing the correspondingposterior predictive mean
x_predicted <- as.matrix(jags_samples[,x_selected_cols])
x_predicted <- colMeans(x_predicted, na.rm = TRUE)

y_predicted <- as.matrix(jags_samples[,y_selected_cols])
y_predicted <- colMeans(y_predicted, na.rm = TRUE)

z_predicted <- as.matrix(jags_samples[,z_selected_cols])
z_predicted <- colMeans(z_predicted, na.rm = TRUE)

vx_predicted <- as.matrix(jags_samples[,vx_selected_cols])
vx_predicted <- colMeans(vx_predicted, na.rm = TRUE)

vy_predicted <- as.matrix(jags_samples[,vy_selected_cols])
vy_predicted <- colMeans(vy_predicted, na.rm = TRUE)

vz_predicted <- as.matrix(jags_samples[,vz_selected_cols])
vz_predicted <- colMeans(vz_predicted, na.rm = TRUE)

# Running a for loop for every time step starting from t=61 to t=66, where we compute the euclidean dis

for (t in 1:6) {
  cat("At time step:", t + 60, "\n")

  distance_measure <- sqrt((x_observed[t] - x_predicted[t])^2 + (y_observed[t] - y_predicted[t])^2 + (z_
  cat("  Euclidean distance on Position :", distance_measure, "\n")

  velocity_measure <- sqrt((vx_observed[t] - vx_predicted[t])^2 + (vy_observed[t] - vy_predicted[t])^2 +
  cat("  Euclidean distance on Velocity :", velocity_measure, "\n")

}
```

```
## At time step: 61
##   Euclidean distance on Position : 0.003774736
##   Euclidean distance on Velocity : 0.5740447
## At time step: 62
##   Euclidean distance on Position : 0.006816362
##   Euclidean distance on Velocity : 1.013555
## At time step: 63
##   Euclidean distance on Position : 0.0044015
##   Euclidean distance on Velocity : 0.5812965
## At time step: 64
##   Euclidean distance on Position : 0.005939956
##   Euclidean distance on Velocity : 0.4441614
## At time step: 65
##   Euclidean distance on Position : 0.005314273
##   Euclidean distance on Velocity : 0.3401609
## At time step: 66
##   Euclidean distance on Position : 0.007987187
##   Euclidean distance on Velocity : 1.079226
```

QUESTION 5 - IMPROVING THE MODEL THROUGH MILSTEIN SCHEME

MOTIVATION BEHIND USING THIS SCHEME

The Milstein scheme is favored over the Euler-Maruyama scheme in the Bayesian model for ping-pong ball movement due to its superior precision and stability. Unlike the Euler-Maruyama scheme, the Milstein scheme minimizes biases and maintains crucial dynamic properties, making it especially advantageous in scenarios where inaccuracies and drift biases need to be minimized. With its ability to effectively handle stiff equations and reduce strong order errors, the Milstein scheme ensures more reliable simulations, particularly in systems involving angular velocity. In summary, opting for the Milstein scheme is motivated by its capacity to deliver accurate and realistic predictions by preserving the model's dynamics with higher fidelity compared to the Euler-Maruyama scheme.

COMPARITIVE RESULTS BETWEEN BOTH THE APPROACHES

The comparative analysis of the Milstein and Euler-Maruyama schemes indicates consistent and comparable performance in predicting the ping-pong ball's position and velocity over successive time steps. Both schemes exhibit minor variations in Euclidean distances for position and velocity, with the Milstein scheme demonstrating slightly enhanced accuracy in specific instances. However, the overall reliability and stability of the Bayesian model persist, highlighting its capacity to effectively capture the nuanced dynamics of the ball's motion. The inferences suggest that the Milstein scheme may offer marginal advantages in certain scenarios, but the Euler-Mayurama scheme remains an optimal choice for this Bayesian model of ping pong simulation.

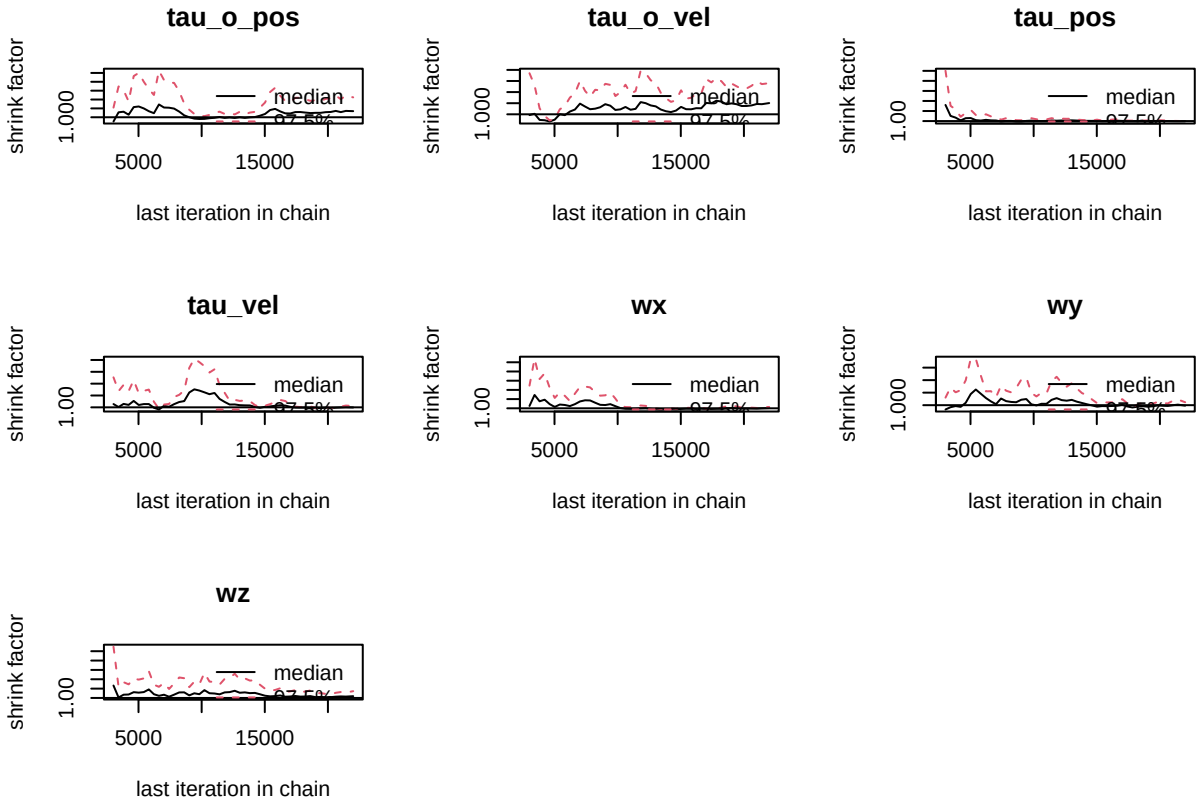
```
##### JAGS model corresponding to Milstein scheme


library(rjags)
library(coda)

# Model string
model_string_jags <- "
model {

x[1] ~ dnorm(0,1)
y[1] ~ dnorm(0,1)
z[1] ~ dnorm(0,1)
vx[1] ~ dnorm(0,25)
vy[1] ~ dnorm(0,25)
vz[1] ~ dnorm(0,25)

# Setting gamma priors for the 'tau' values
tau_pos ~ dgamma(1,1)
tau_vel ~ dgamma(1,1)
tau_o_pos ~ dgamma(1,1)
tau_o_vel ~ dgamma(1,1)

# Setting normal priors for the 'omega' values
wx ~ dnorm(0,1)
wy ~ dnorm(0,1)
wz ~ dnorm(0,1)

# Likelihood calculation
for (k in 1:(n-1)) {

   V_magnitude[k] <- sqrt(vx[k]^2 + vy[k]^2 + vz[k]^2)
```

```
   x[k+1] ~ dnorm(x[k] +(vx[k] * delta[k]), tau_pos)
   xrep[k+1] ~ dnorm(x[k] +(vx[k] * delta[k]), tau_pos)

   y[k+1] ~ dnorm(y[k] +(vy[k] * delta[k]), tau_pos)
   yrep[k+1] ~ dnorm(y[k] +(vy[k] * delta[k]), tau_pos)

   z[k+1] ~ dnorm(z[k] +(vz[k] * delta[k]), tau_pos)
   zrep[k+1] ~ dnorm(z[k] +(vz[k] * delta[k]), tau_pos)


   vx[k+1] ~ dnorm(vx[k] + (0.5 * kd * V_magnitude[k] * vx[k] + 0.5 * km * (wy * vz[k] - wz * vy[k])) *
   vxrep[k+1] ~ dnorm(vx[k] + (0.5 * kd * V_magnitude[k] * vx[k] + 0.5 * km * (wy * vz[k] - wz * vy[k]))

   vy[k+1] ~ dnorm(vy[k] + (0.5 * kd * V_magnitude[k] * vy[k] + 0.5 * km * (wz * vx[k] - wx * vz[k])) *
   vyrep[k+1] ~ dnorm(vy[k] + (kd * V_magnitude[k] * vy[k] + 0.5 * km * (wz * vx[k] - wx * vz[k])) * del

   vz[k+1] ~ dnorm(vz[k] + (0.5 * kd * V_magnitude[k] * vz[k] + 0.5 * km * (wx * vy[k] - wy * vx[k]) - g
   vzrep[k+1] ~ dnorm(vz[k] + (0.5 * kd * V_magnitude[k] * vz[k] + 0.5 * km * (wx * vy[k] - wy * vx[k])

}

# Observation model
  for(k in 1:(n-1)){

    xo[k] ~ dnorm(x[k],tau_o_pos)
    yo[k] ~ dnorm(y[k],tau_o_pos)
    zo[k] ~ dnorm(z[k],tau_o_pos)
    vxo[k] ~ dnorm(vx[k],tau_o_vel)
    vyo[k] ~ dnorm(vy[k],tau_o_vel)
    vzo[k] ~ dnorm(vz[k],tau_o_vel)
  }
}"

g <- 9.827
kd <- (-1/(2*0.0027))*0.456*1.29*0.001256
km <- (1/(2*0.0027))*1*1.29*0.001256*0.02

jags_data = list (
       n = n,
       xo = xo,
       yo = yo,
       zo = zo,
       vxo = vxo,
       vyo = vyo,
       vzo = vzo,
       g = g,
       km = km,
       kd = kd,
       delta = diff(T)
       )

updated_jags_model = jags.model(textConnection(model_string_jags), data = jags_data, n.chains = 4)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 360
##    Unobserved stochastic nodes: 823
##    Total graph size: 3857
##
## Initializing model
```

```
update(updated_jags_model,1000)

params_of_interest <- c("tau_pos","tau_vel","tau_o_pos","tau_o_vel","wx","wy","wz")

# Setting the number of iterations, burn-in, and thinning parameters
n.iter <- 20000
n.burn <- 1000
n.thin <- 20

updated_jags_samples=coda.samples(updated_jags_model,variable.names=params_of_interest,n.iter = n.iter,
```

```
# Computing the Gelman-Rubin statistic

par(mfrow=c(2,2))
gelman.plot(updated_jags_samples)
```



```r
gelman.diag(updated_jags_samples)
```

```
## Potential scale reduction factors:
##
##           Point est. Upper C.I.
## tau_o_pos      1.003       1.01
## tau_o_vel      1.005       1.01
## tau_pos        0.999       1.00
## tau_vel        1.000       1.00
## wx             1.000       1.00
## wy             1.000       1.00
## wz             1.002       1.01
##
## Multivariate psrf
##
## 1.01
```

```r
effectiveSize(updated_jags_samples)
```

```
## tau_o_pos tau_o_vel   tau_pos   tau_vel        wx        wy        wz
##  3832.831  4569.423  3926.748  3707.589  4162.119  4000.000  4000.000
```

```r
summary(updated_jags_samples)
```

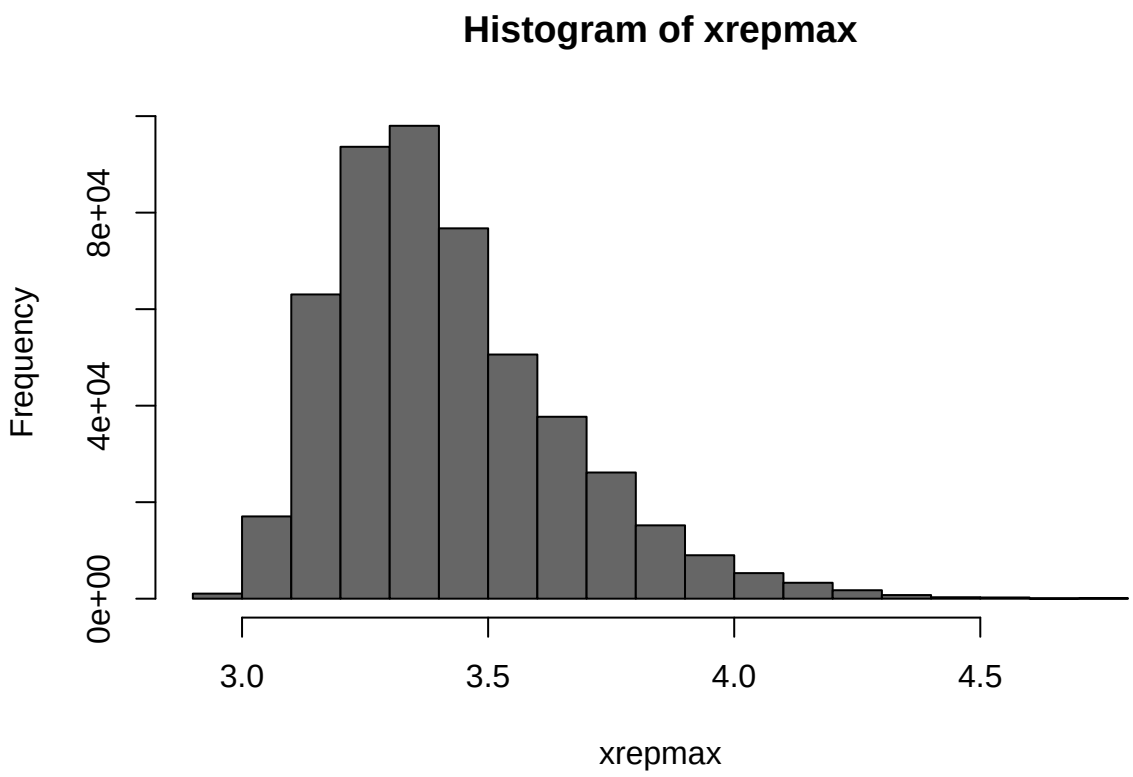
```
##
## Iterations = 2020:22000
## Thinning interval = 20
## Number of chains = 4
## Sample size per chain = 1000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                Mean     SD Naive SE Time-series SE
## tau_o_pos 48.072539 6.2640 0.099043       0.101318
## tau_o_vel  2.504909 0.3445 0.005447       0.005135
## tau_pos   42.454489 5.8006 0.091716       0.092649
## tau_vel    6.159045 1.3345 0.021101       0.021997
## wx        -0.002161 1.0086 0.015947       0.015653
## wy         0.015316 0.9964 0.015754       0.015753
## wz        -0.006518 1.0132 0.016020       0.016001
##
## 2. Quantiles for each variable:
##
##             2.5%     25%      50%     75%  97.5%
## tau_o_pos 36.653 43.5782 47.78428 52.1354 61.246
## tau_o_vel  1.875  2.2684  2.48840  2.7178  3.241
## tau_pos   32.150 38.3824 42.11976 46.3102 54.559
## tau_vel    3.927  5.2047  6.03572  7.0013  9.091
## wx        -1.953 -0.6799  0.00401  0.6775  2.012
## wy        -1.925 -0.6593  0.02469  0.7044  1.934
## wz        -2.048 -0.6812  0.01689  0.6784  1.965
```

```
no_of_iterations <- 500000
millstein_samples = coda.samples(updated_jags_model,variable.names=c("xrep","yrep","zrep","vxrep","vyrep
```

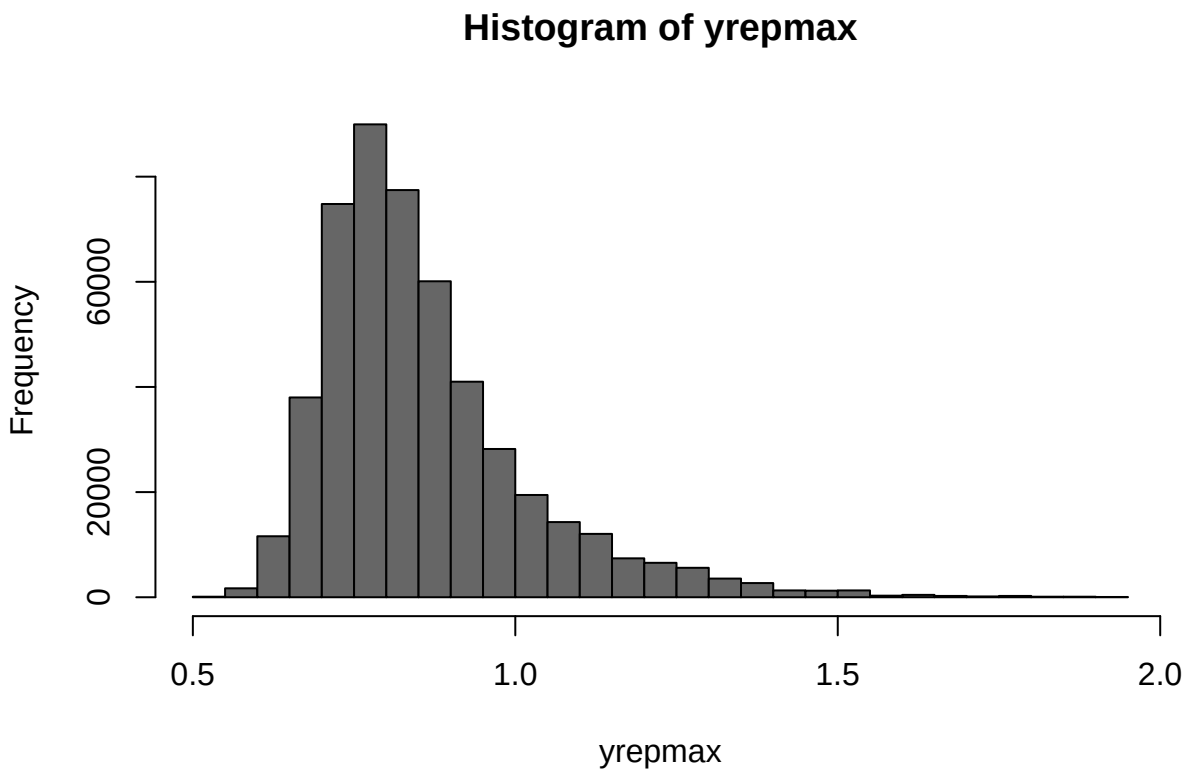
```
xrep <- yrep <- zrep <- vxrep <- vyrep <- vzrep <- matrix(NA, nrow = no_of_iterations, ncol = (n-1))
for(i in 1:(n-1)) {

xrep[,i] <- millstein_samples[[1]][,paste0('xrep[',i+1,']')]
yrep[,i] <- millstein_samples[[1]][,paste0('yrep[',i+1,']')]
zrep[,i] <- millstein_samples[[1]][,paste0('zrep[',i+1,']')]
vxrep[,i] <- millstein_samples[[1]][,paste0('vxrep[',i+1,']')]
vyrep[,i] <- millstein_samples[[1]][,paste0('vyrep[',i+1,']')]
vzrep[,i] <- millstein_samples[[1]][,paste0('vzrep[',i+1,']')]

}
```

```
require(fBasics) #statistics of interest in this case (and different from the ones included in the mode

xrepmax=apply(xrep,1,max)
xrepmedian=apply(xrep,1,median)
xrepskewness=apply(xrep,1,skewness)
xrepkurtosis=apply(xrep,1,kurtosis)


yrepmax=apply(yrep,1,max)
yrepmedian=apply(yrep,1,median)
yrepskewness=apply(yrep,1,skewness)
yrepkurtosis=apply(yrep,1,kurtosis)


zrepmax=apply(zrep,1,max)
zrepmedian=apply(zrep,1,median)
zrepskewness=apply(zrep,1,skewness)
zrepkurtosis=apply(zrep,1,kurtosis)


vxrepmax=apply(vxrep,1,max)
vxrepmedian=apply(vxrep,1,median)
vxrepskewness=apply(vxrep,1,skewness)
vxrepkurtosis=apply(vxrep,1,kurtosis)


vyrepmax=apply(vyrep,1,max)
vyrepmedian=apply(vyrep,1,median)
vyrepskewness=apply(vyrep,1,skewness)
vyrepkurtosis=apply(vyrep,1,kurtosis)


vzrepmax=apply(vzrep,1,max)
vzrepmedian=apply(vzrep,1,median)
vzrepskewness=apply(vzrep,1,skewness)
vzrepkurtosis=apply(vzrep,1,kurtosis)
```

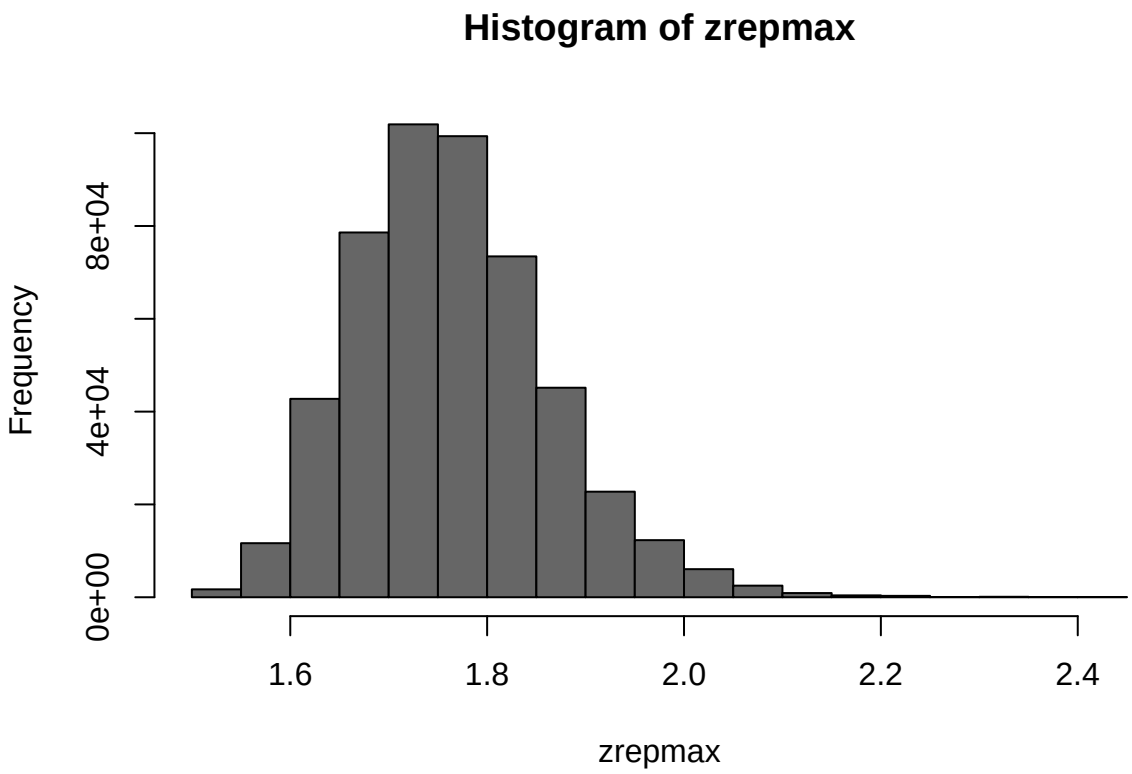
```
hist(xrepmax,col="gray40")
abline(v=max(xo),col="red",lwd=2)
```



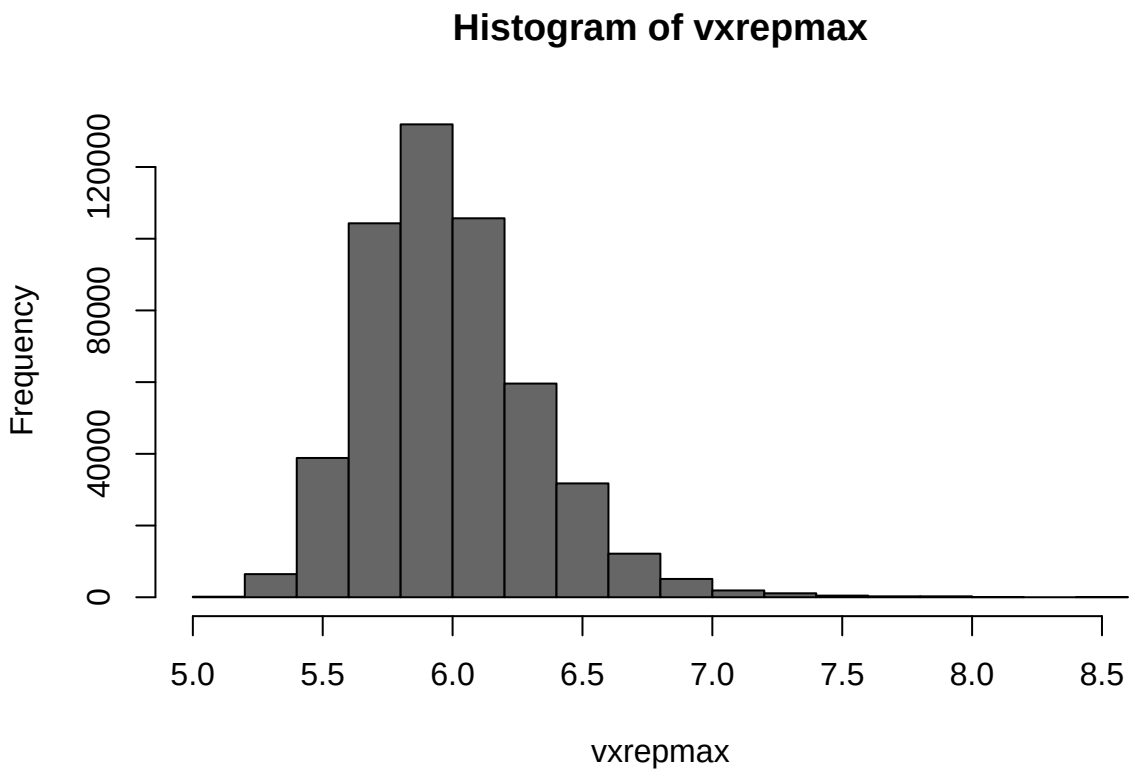

```
hist(yrepmax,col="gray40")
abline(v=max(yo),col="red",lwd=2)
```



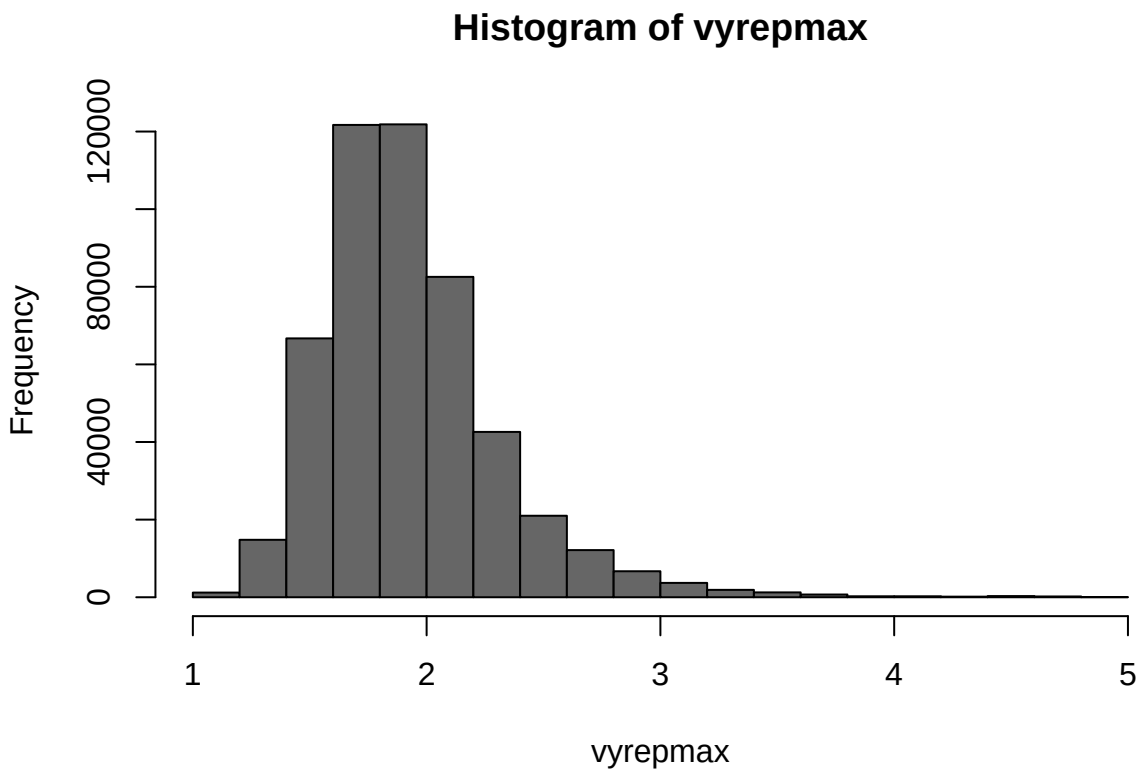

```
hist(zrepmax,col="gray40")
abline(v=max(zo),col="red",lwd=2)
```



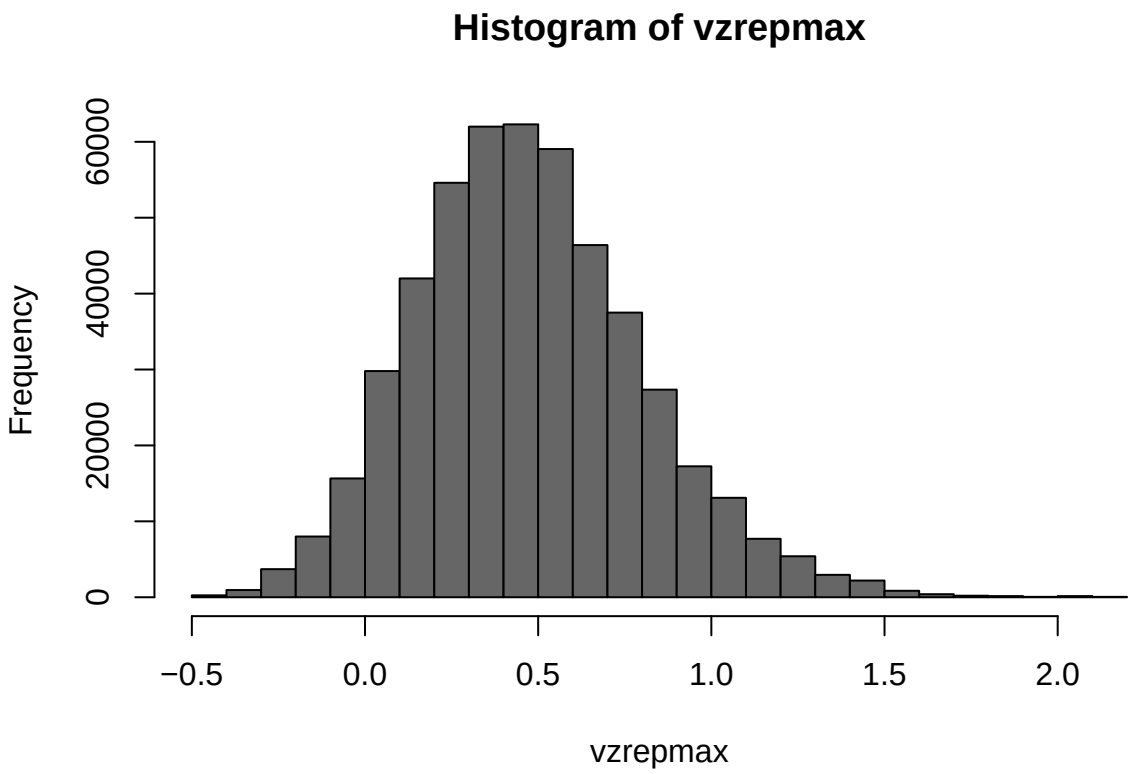

```
hist(vxrepmax,col="gray40")
abline(v=max(vxo),col="red",lwd=2)
```



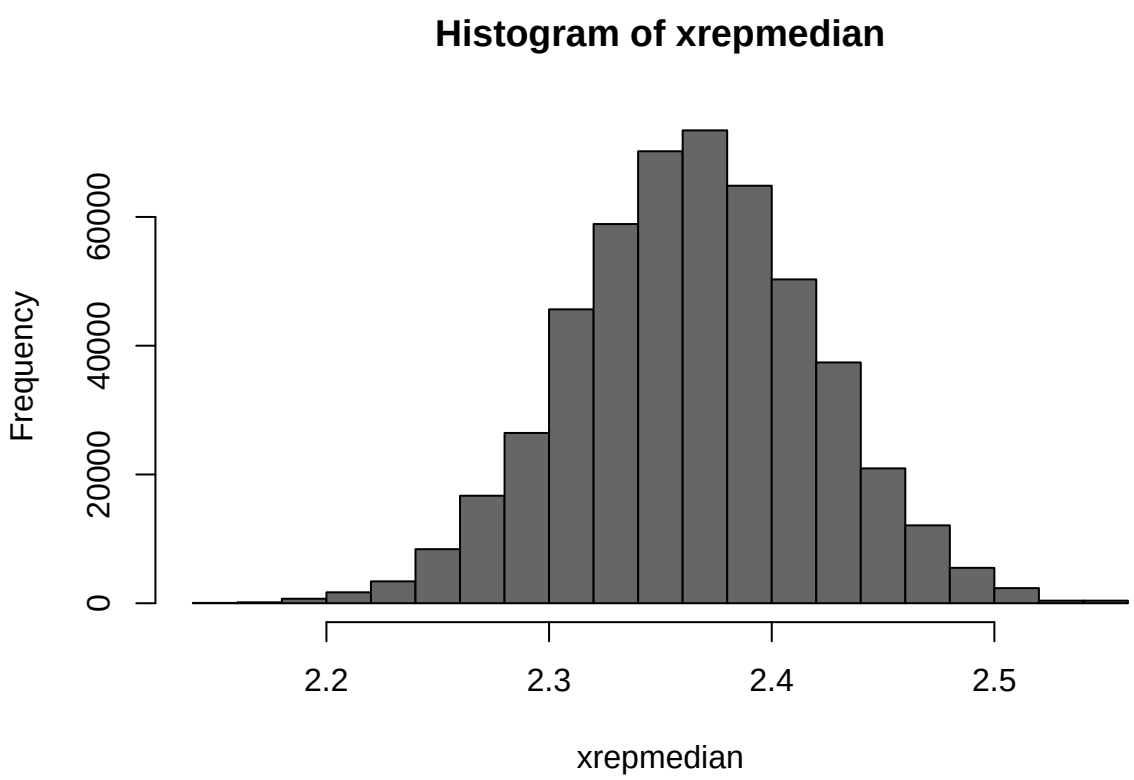

```
hist(vyrepmax,col="gray40")
abline(v=max(vyo),col="red",lwd=2)
```



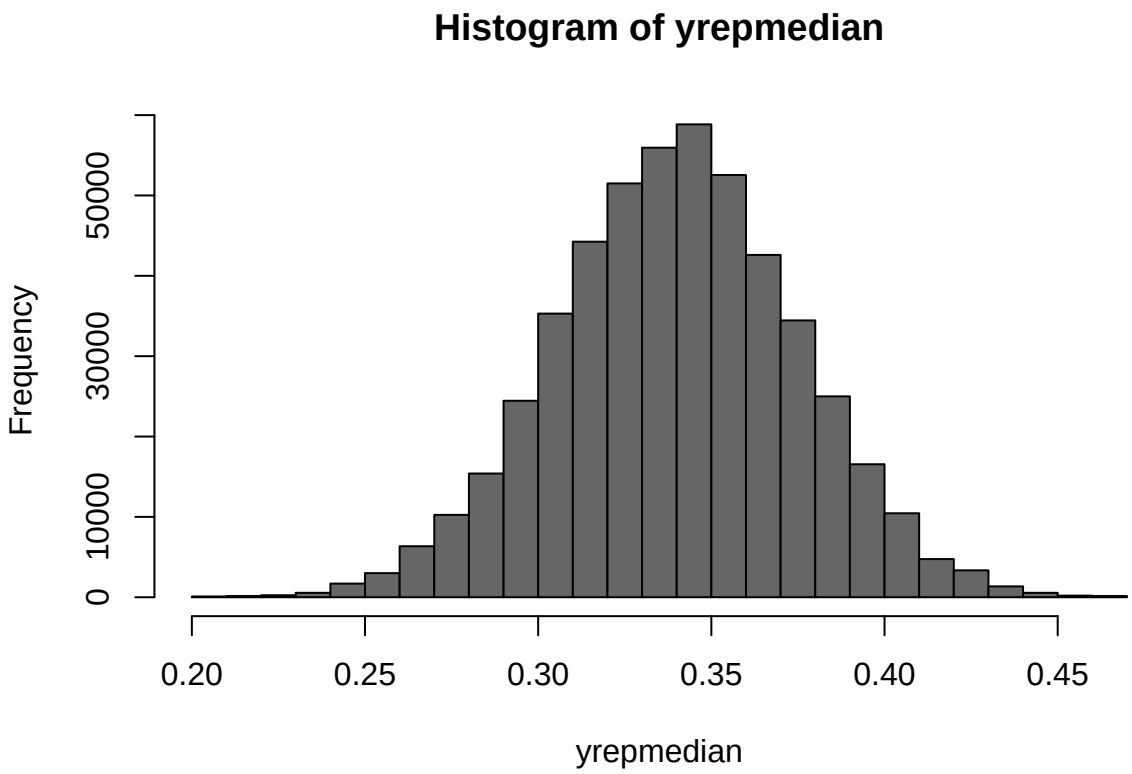

```
hist(vzrepmax,col="gray40")
abline(v=max(vzo),col="red",lwd=2)
```



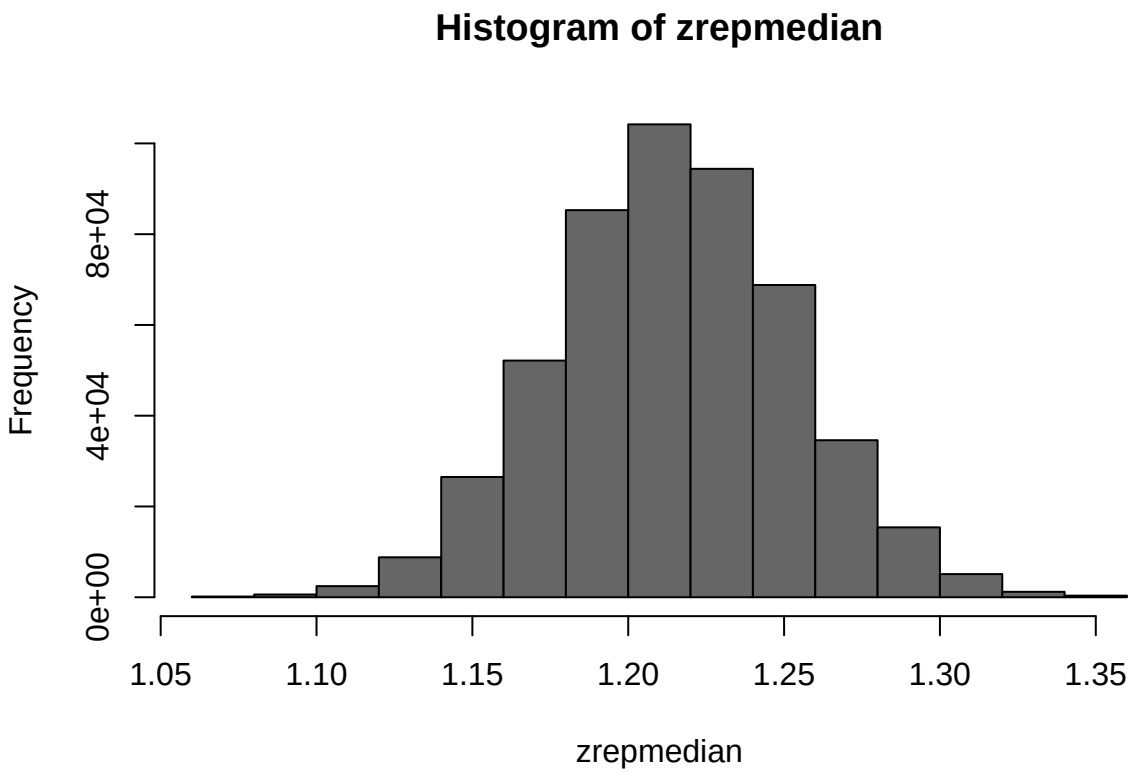

```
#Predictive checks using replicated data - median
hist(xrepmedian,col="gray40")
abline(v=median(xo),col="red",lwd=2)
```



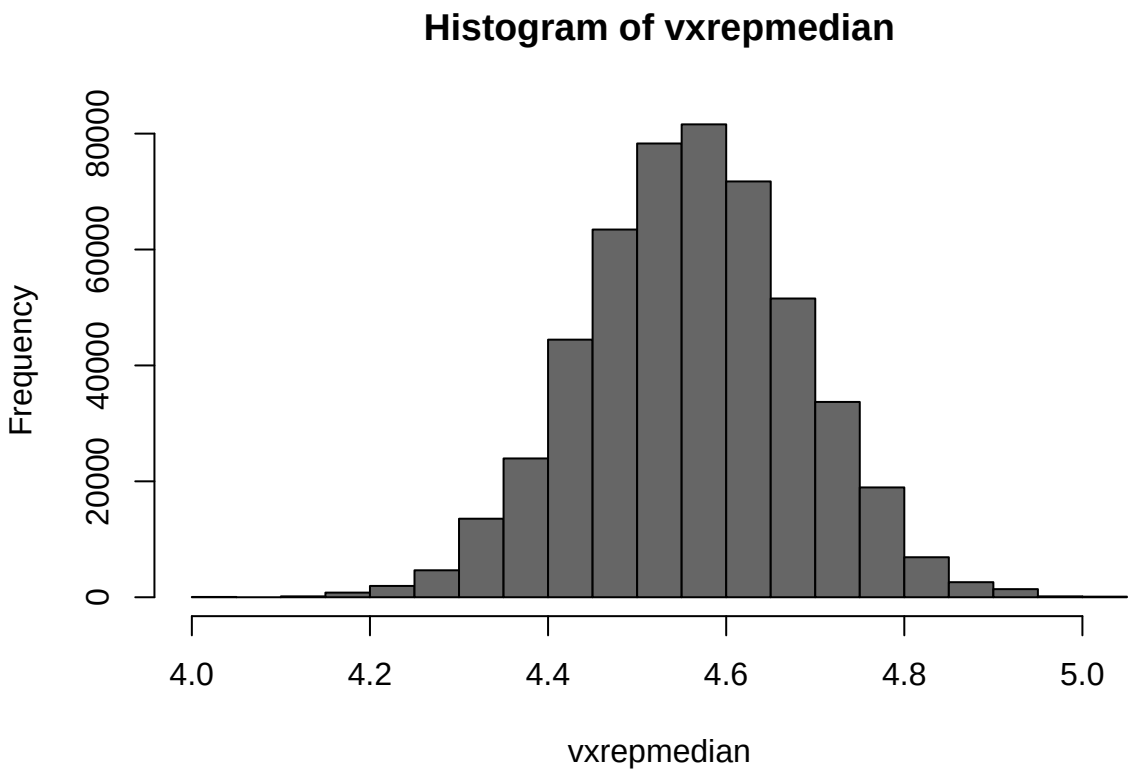

```
hist(yrepmedian,col="gray40")
abline(v=median(yo),col="red",lwd=2)
```



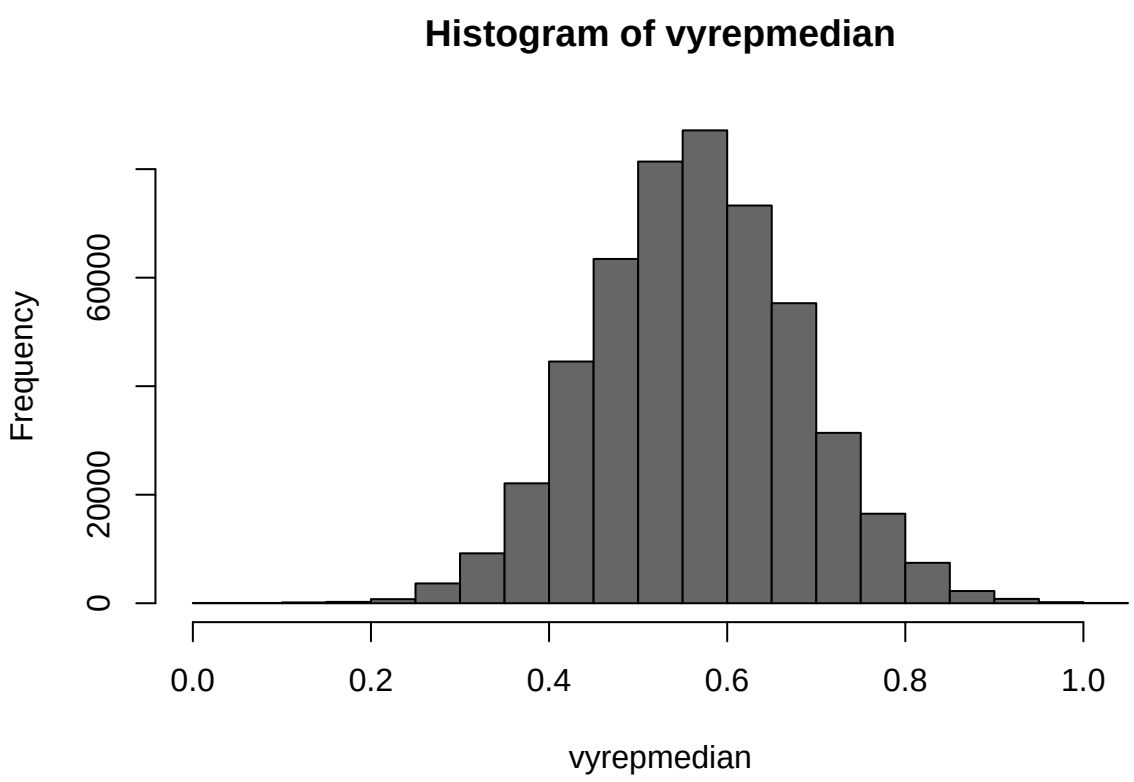

```
hist(zrepmedian,col="gray40")
abline(v=median(zo),col="red",lwd=2)
```



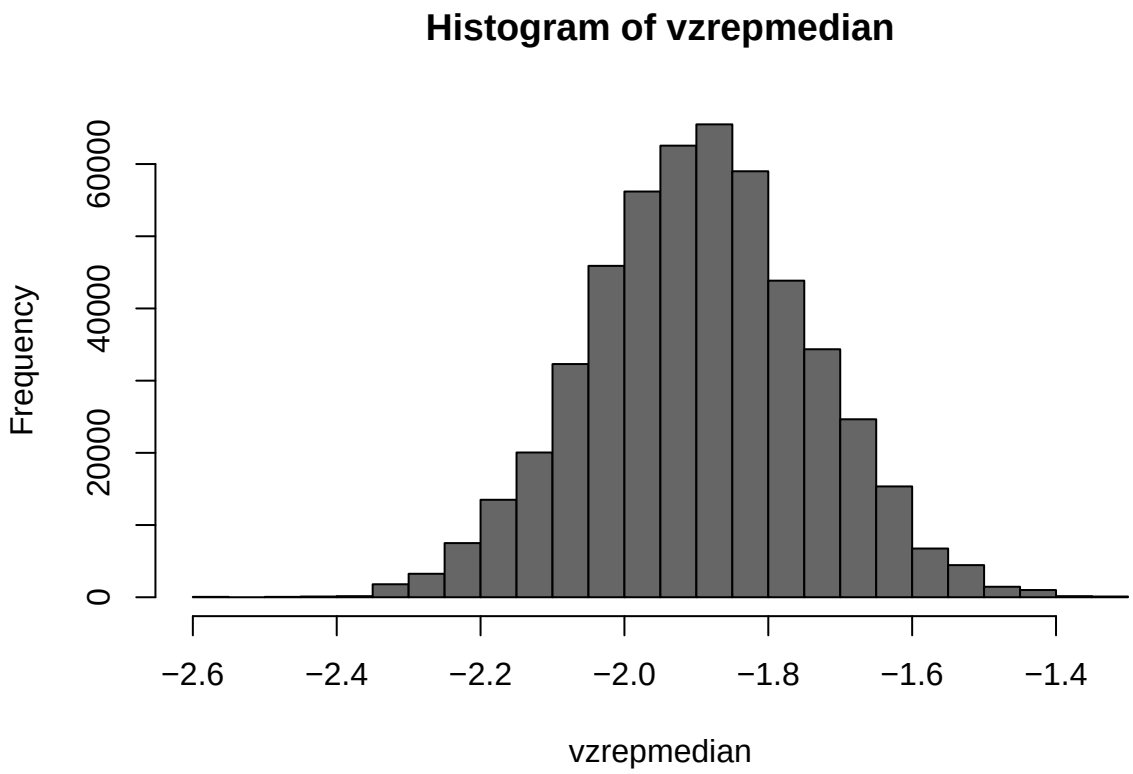

```
hist(vxrepmedian,col="gray40")
abline(v=median(vxo),col="red",lwd=2)
```



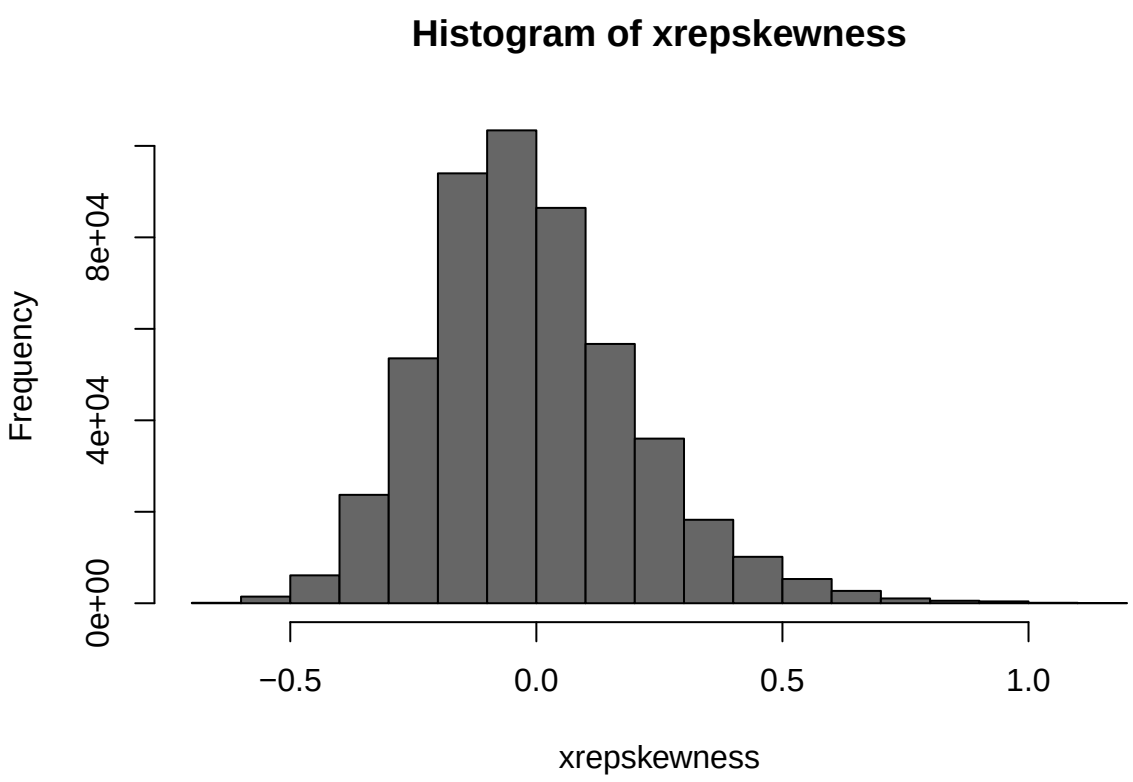

```
hist(vyrepmedian,col="gray40")
abline(v=median(vyo),col="red",lwd=2)
```



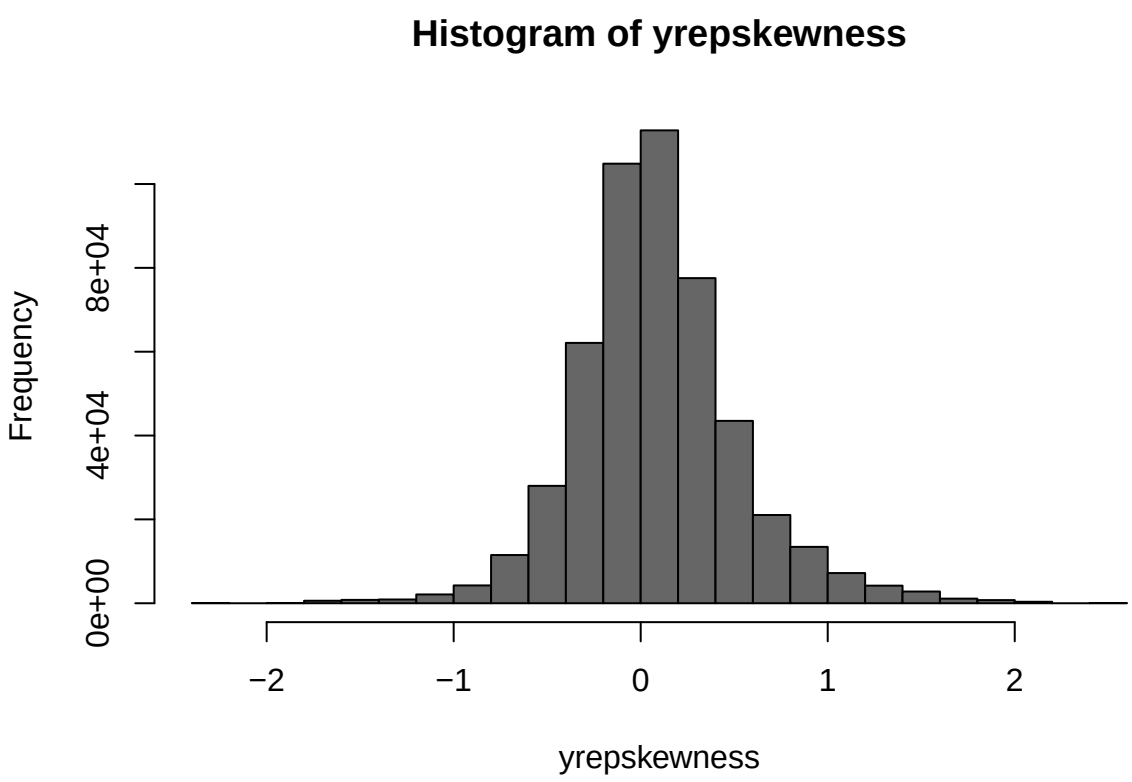

```
hist(vzrepmedian,col="gray40")
abline(v=median(vzo),col="red",lwd=2)
```



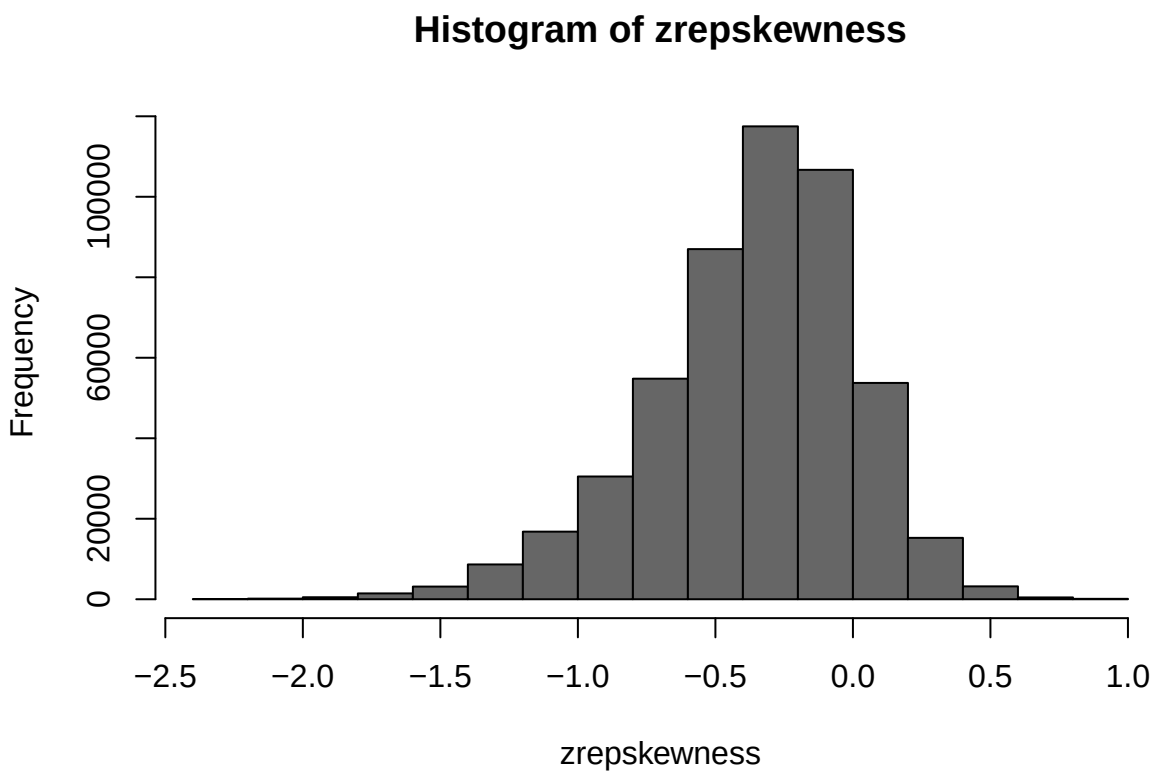

```
#Predictive checks using replicated data - skewness

hist(xrepskewness,col="gray40")
abline(v=skewness(xo),col="red",lwd=2)
```



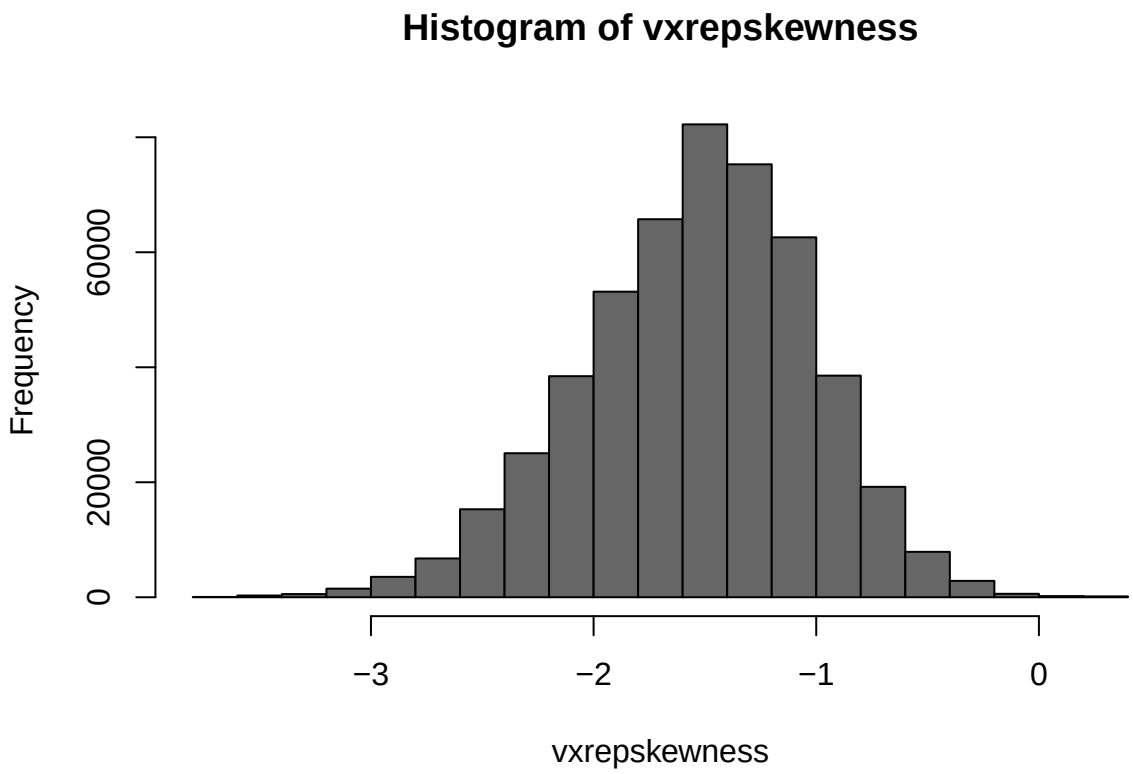

```
hist(yrepskewness,col="gray40")
abline(v=skewness(yo),col="red",lwd=2)
```



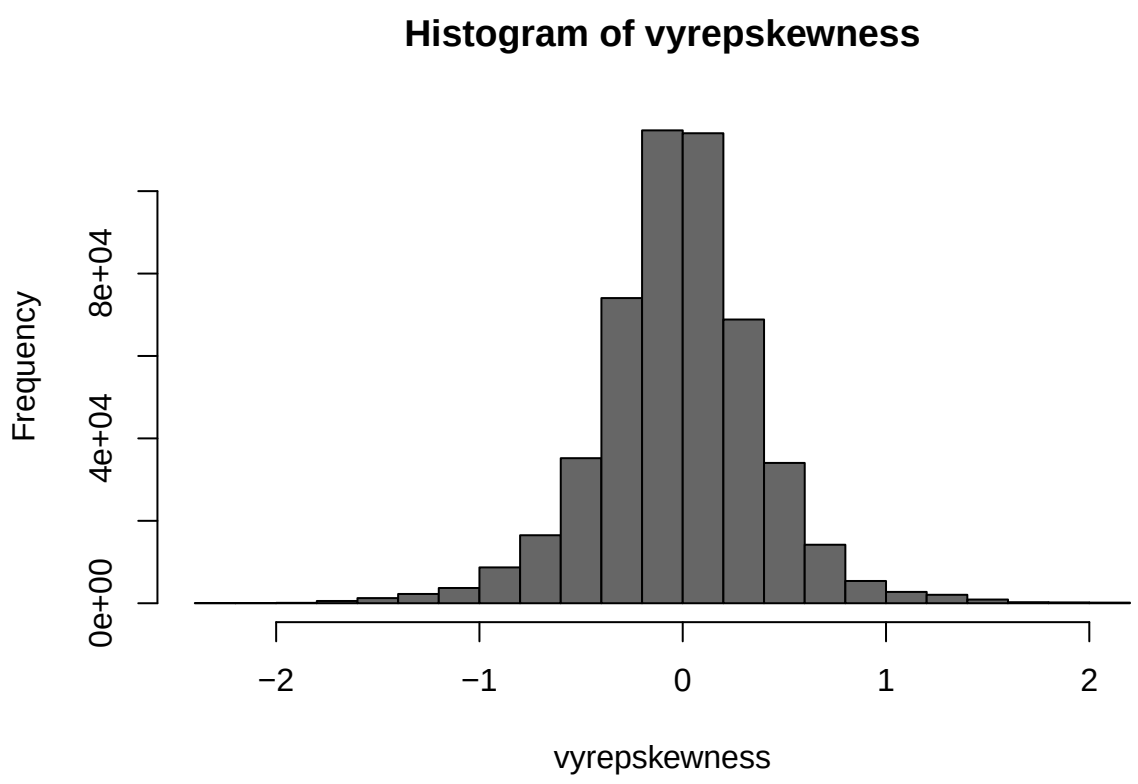

```
hist(zrepskewness,col="gray40")
abline(v=skewness(zo),col="red",lwd=2)
```



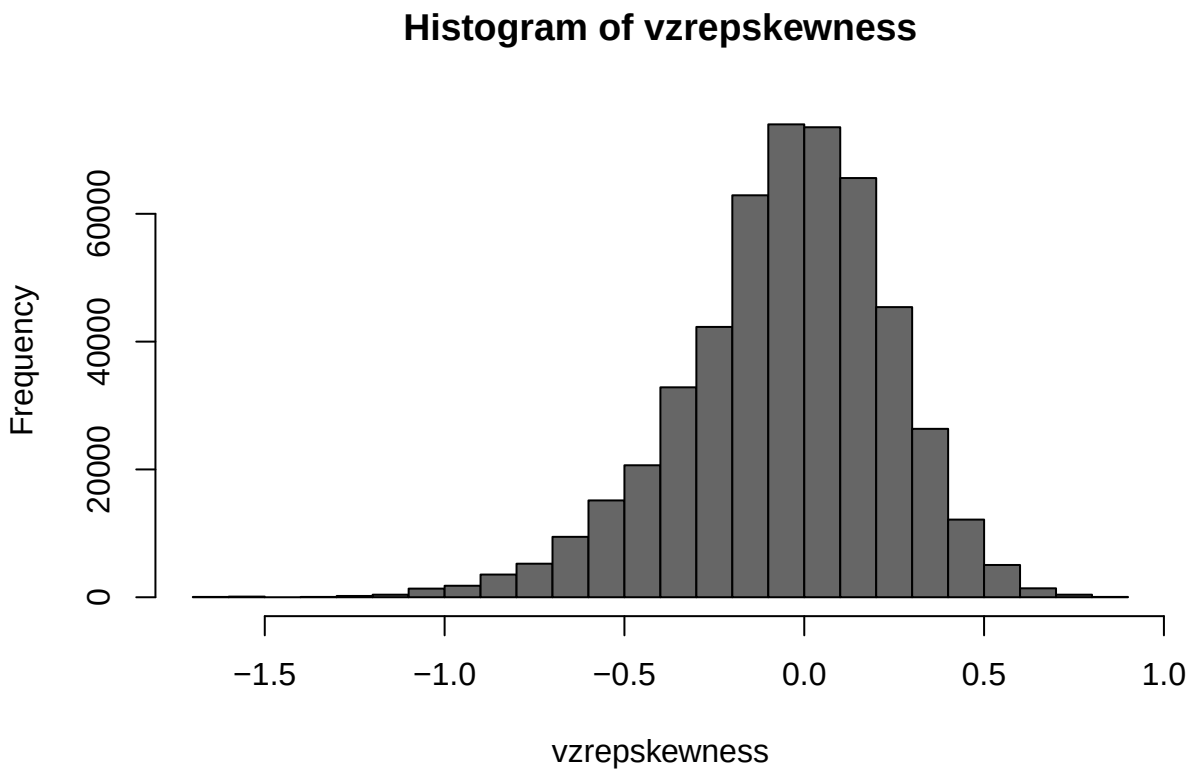

```
hist(vxrepskewness,col="gray40")
abline(v=skewness(vxo),col="red",lwd=2)
```



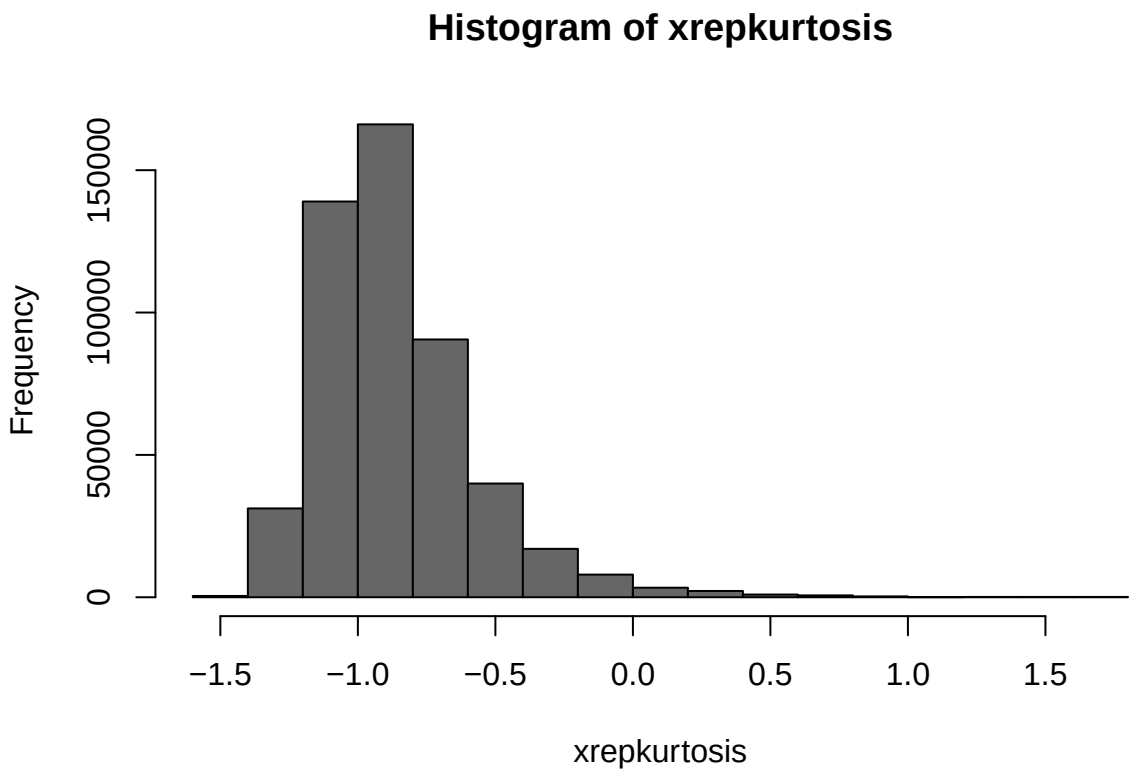

```
hist(vyrepskewness,col="gray40")
abline(v=skewness(vyo),col="red",lwd=2)
```



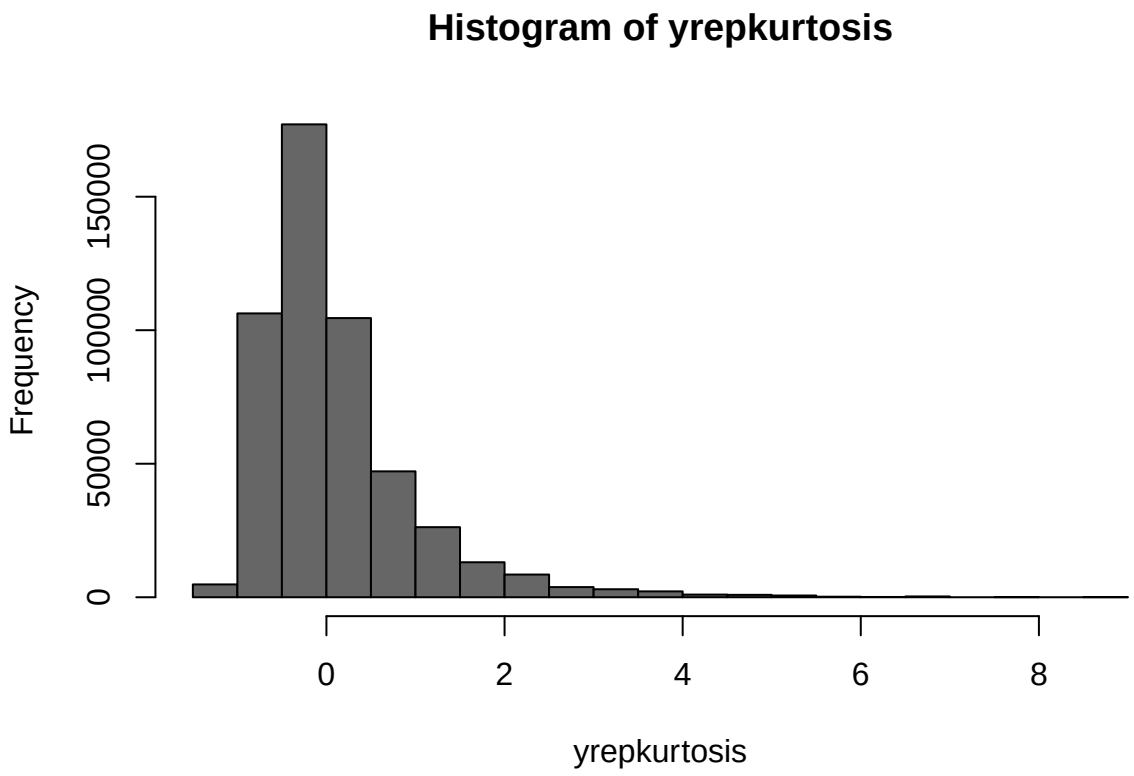

```
hist(vzrepskewness,col="gray40")
abline(v=skewness(vzo),col="red",lwd=2)
```



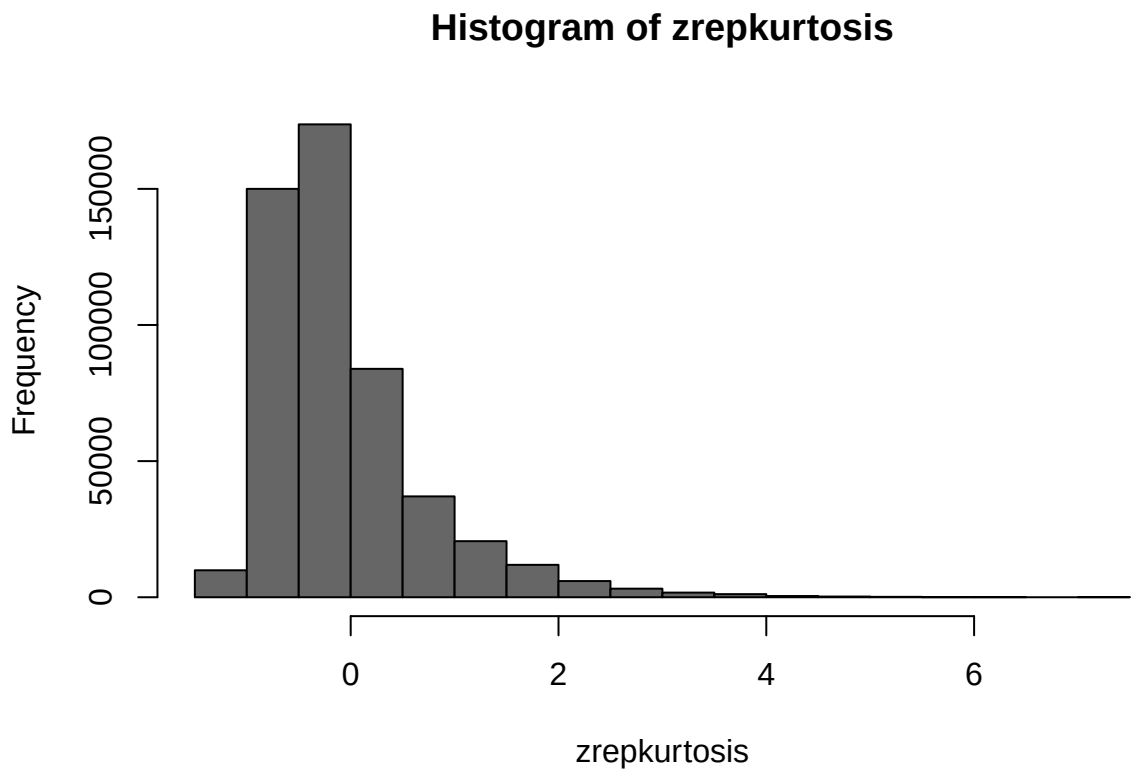

```
#Predictive checks using replicated data - kurtosis
hist(xrepkurtosis,col="gray40")
abline(v=kurtosis(xo),col="red",lwd=2)
```



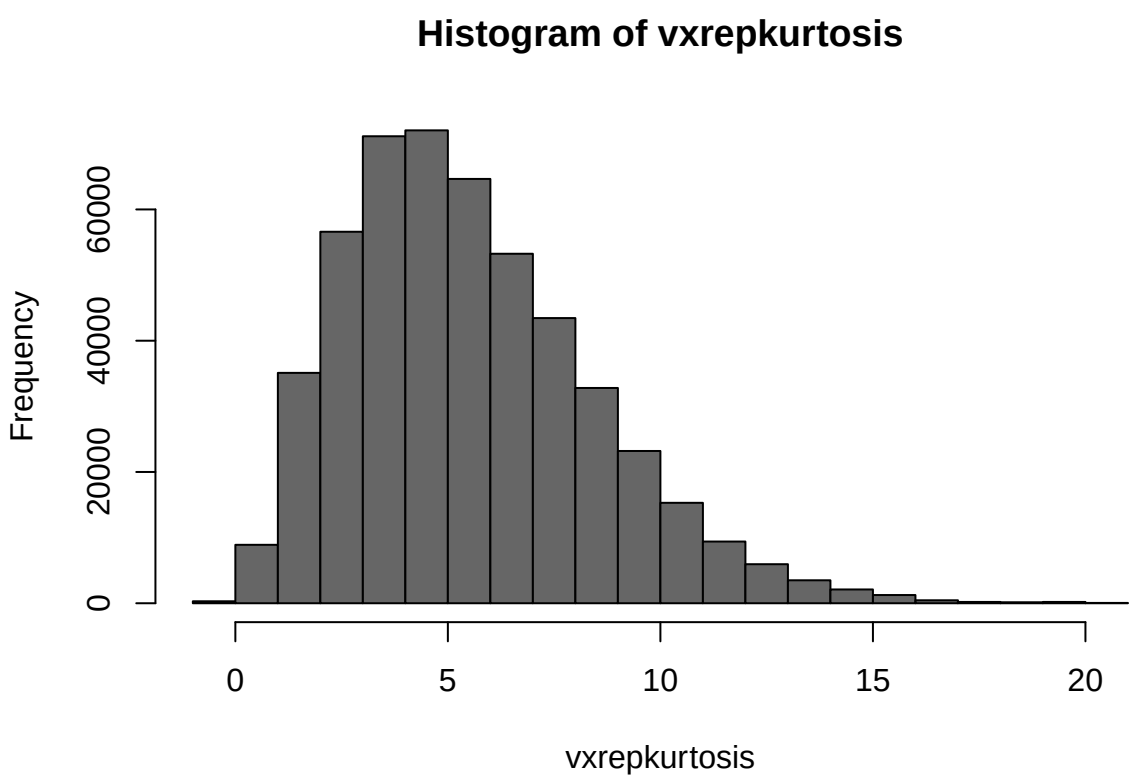

```
hist(yrepkurtosis,col="gray40")
abline(v=kurtosis(yo),col="red",lwd=2)
```



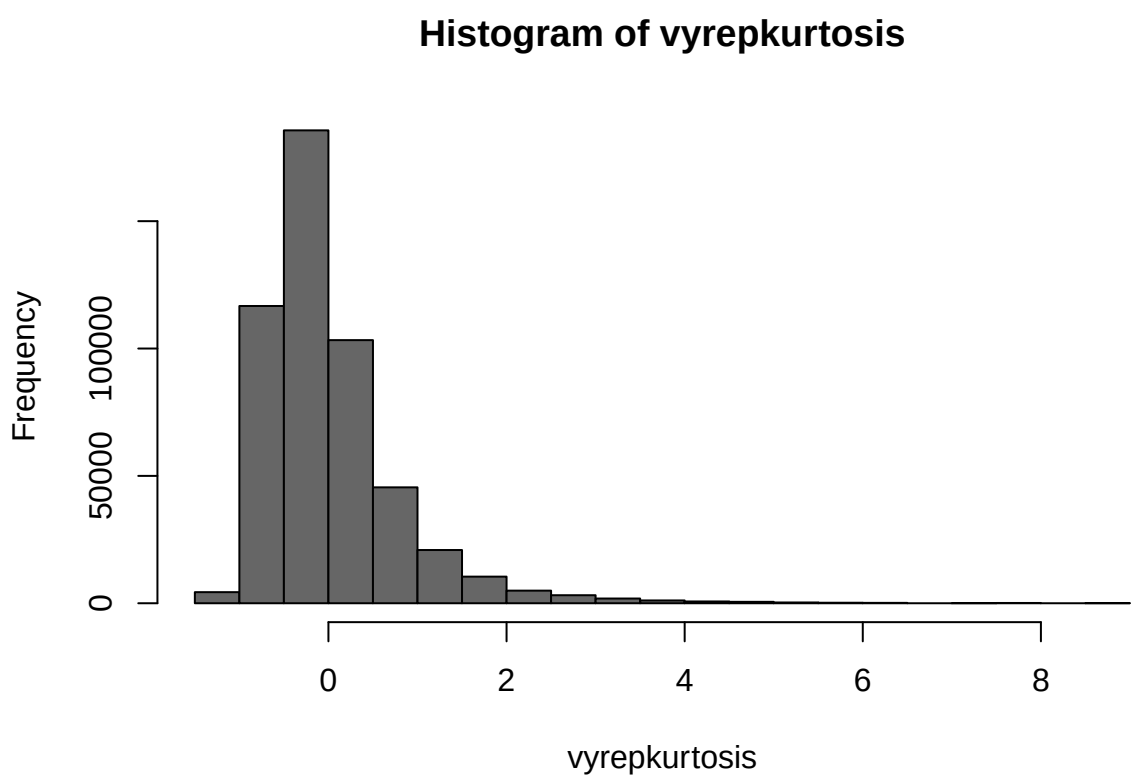

```
hist(zrepkurtosis,col="gray40")
abline(v=kurtosis(zo),col="red",lwd=2)
```



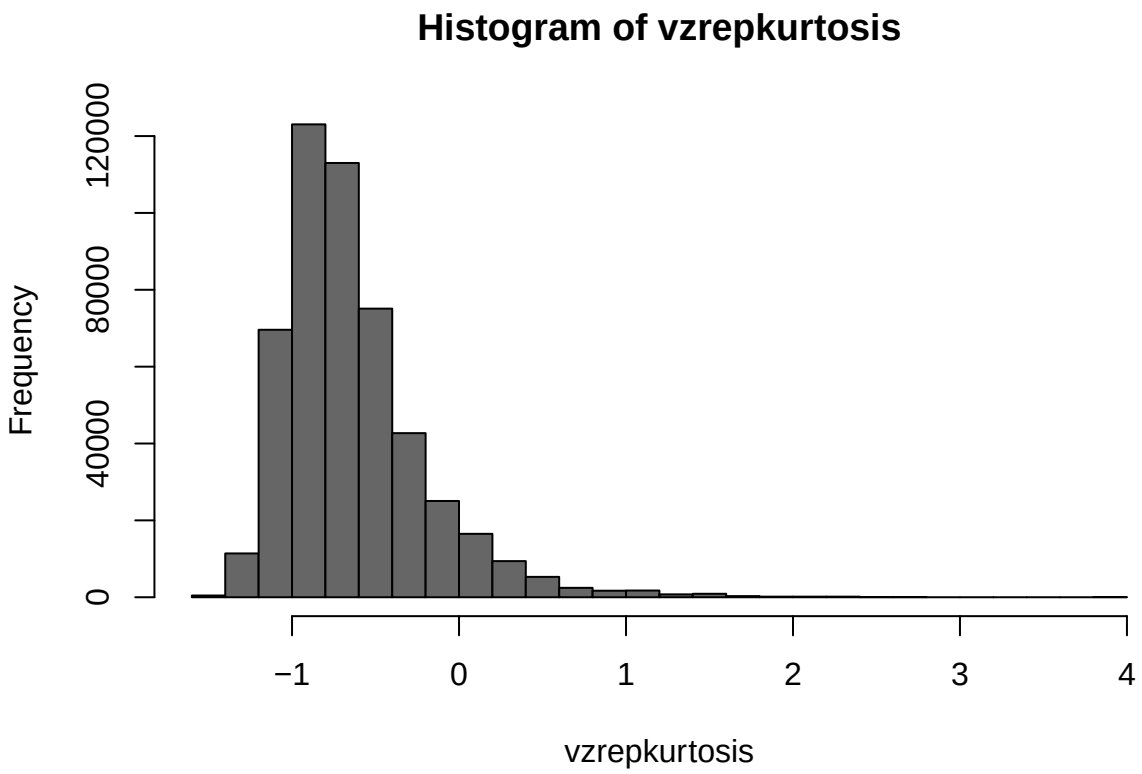

```
hist(vxrepkurtosis,col="gray40")
abline(v=kurtosis(vxo),col="red",lwd=2)
```



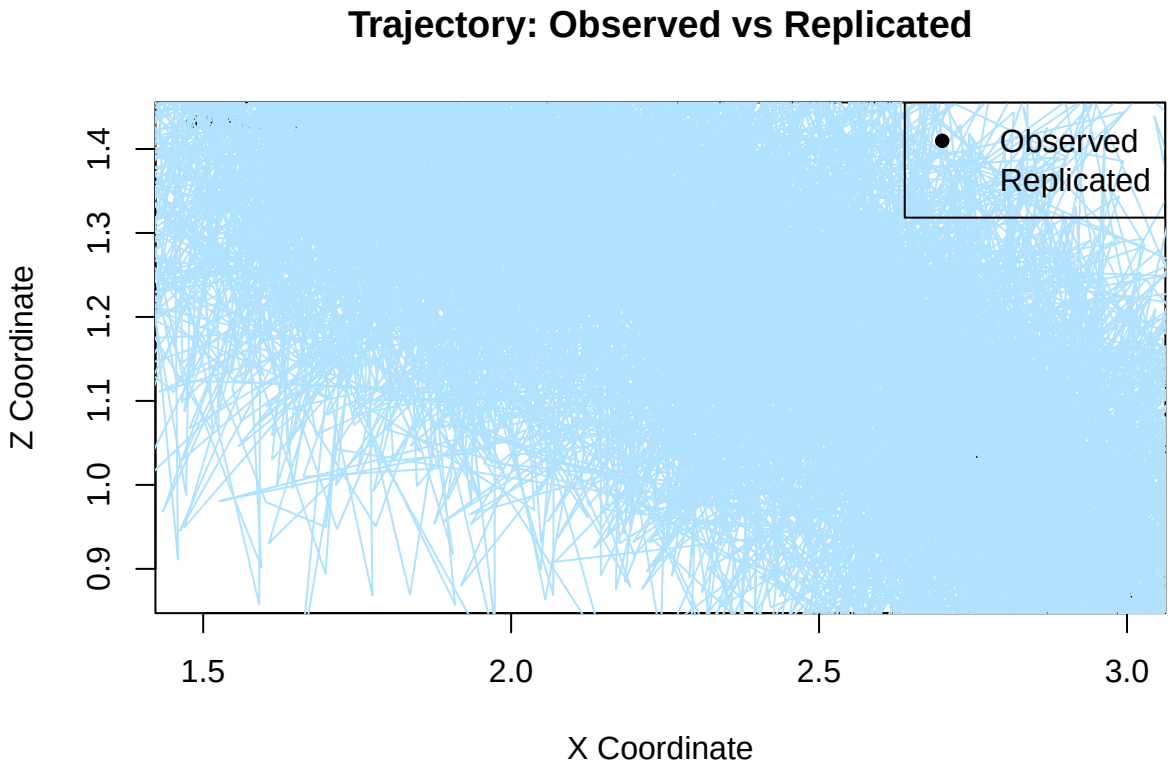

```
hist(vyrepkurtosis,col="gray40")
abline(v=kurtosis(vyo),col="red",lwd=2)
```




```
hist(vzrepkurtosis,col="gray40")
abline(v=kurtosis(vzo),col="red",lwd=2)
```




```r
# Assuming xrep and zrep are your replicated x and z coordinates
# Plot observed trajectory
plot(xo, zo, col = "black", pch = 16, main = "Trajectory: Observed vs Replicated", xlab = "X Coordinate
# Plot 100 replicated trajectories
for (i in 1:100) { lines(xrep[i, ], zrep[i, ], col = "lightskyblue1", lty = 1) }

# Add legend
legend("topright", legend = c("Observed", "Replicated"), col = c("black", "lightskyblue1"), pch = c(16,
```




```
n=66;
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)];

x_observed = xo[(n-5):n]
y_observed = yo[(n-5):n]
z_observed = zo[(n-5):n]
vx_observed = vxo[(n-5):n]
vy_observed = vyo[(n-5):n]
vz_observed = vzo[(n-5):n]

### masking the values

xo[(n-5):n] <- NA
yo[(n-5):n] <- NA
zo[(n-5):n] <- NA
vxo[(n-5):n] <- NA
```

```
vyo[(n-5):n] <- NA
vzo[(n-5):n] <- NA

g <- 9.827
km <- 1/(2*0.0027)*1*1.29*0.001256*0.02
kd <- -1/(2*0.0027)*0.456*1.29*0.001256

jags_data = list (
       n = n,
       xo = xo,
       yo = yo,
       zo = zo,
       vxo = vxo,
       vyo = vyo,
       vzo = vzo,
       g = g,
       km = km,
       kd = kd,
       delta = diff(T)
       )

mill_jags_model = jags.model(textConnection(model_string_jags), data = jags_data, n.chains = 1)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 360
##    Unobserved stochastic nodes: 823
##    Total graph size: 3857
##
## Initializing model
```

```
update(mill_jags_model,1000,progress.bar="none")

mill_jags_samples <- coda.samples(mill_jags_model, variable.names = c("x", "y", "z", "vx", "vy", "vz"),

x_selected_cols <- c("x[61]","x[62]","x[63]","x[64]","x[65]","x[66]")
y_selected_cols <- c("y[61]","y[62]","y[63]","y[64]","y[65]","y[66]")
z_selected_cols <- c("z[61]","z[62]","z[63]","z[64]","z[65]","z[66]")
vx_selected_cols <- c("vx[61]","vx[62]","vx[63]","vx[64]","vx[65]","vx[66]")
vy_selected_cols <- c("vy[61]","vy[62]","vy[63]","vy[64]","vy[65]","vy[66]")
vz_selected_cols <- c("vz[61]","vz[62]","vz[63]","vz[64]","vz[65]","vz[66]")

# Fetching out the appropriate columns and computing the correspondingposterior predictive mean
x_predicted <- as.matrix(mill_jags_samples[,x_selected_cols])
x_predicted <- colMeans(x_predicted, na.rm = TRUE)

y_predicted <- as.matrix(mill_jags_samples[,y_selected_cols])
y_predicted <- colMeans(y_predicted, na.rm = TRUE)

z_predicted <- as.matrix(mill_jags_samples[,z_selected_cols])
z_predicted <- colMeans(z_predicted, na.rm = TRUE)
```

```r
vx_predicted <- as.matrix(mill_jags_samples[,vx_selected_cols])
vx_predicted <- colMeans(vx_predicted, na.rm = TRUE)

vy_predicted <- as.matrix(mill_jags_samples[,vy_selected_cols])
vy_predicted <- colMeans(vy_predicted, na.rm = TRUE)

vz_predicted <- as.matrix(mill_jags_samples[,vz_selected_cols])
vz_predicted <- colMeans(vz_predicted, na.rm = TRUE)

# Running a for loop for every time step starting from t=61 to t=66, where we compute the euclidean dis

for (t in 1:6) {
  cat("At time step:", t + 60, "\n")

  distance_measure <- sqrt((x_observed[t] - x_predicted[t])^2 + (y_observed[t] - y_predicted[t])^2 + (z_
  cat("  Euclidean distance on Position :", distance_measure, "\n")

  velocity_measure <- sqrt((vx_observed[t] - vx_predicted[t])^2 + (vy_observed[t] - vy_predicted[t])^2 +
  cat("  Euclidean distance on Velocity :", velocity_measure, "\n")

}
```

```
## At time step: 61
##   Euclidean distance on Position : 0.002117151
##   Euclidean distance on Velocity : 0.592077
## At time step: 62
##   Euclidean distance on Position : 0.007144484
##   Euclidean distance on Velocity : 1.019088
## At time step: 63
##   Euclidean distance on Position : 0.004968056
##   Euclidean distance on Velocity : 0.5662396
## At time step: 64
##   Euclidean distance on Position : 0.006864144
##   Euclidean distance on Velocity : 0.4460303
## At time step: 65
##   Euclidean distance on Position : 0.00672342
##   Euclidean distance on Velocity : 0.2832531
## At time step: 66
##   Euclidean distance on Position : 0.01051743
##   Euclidean distance on Velocity : 1.162992
```