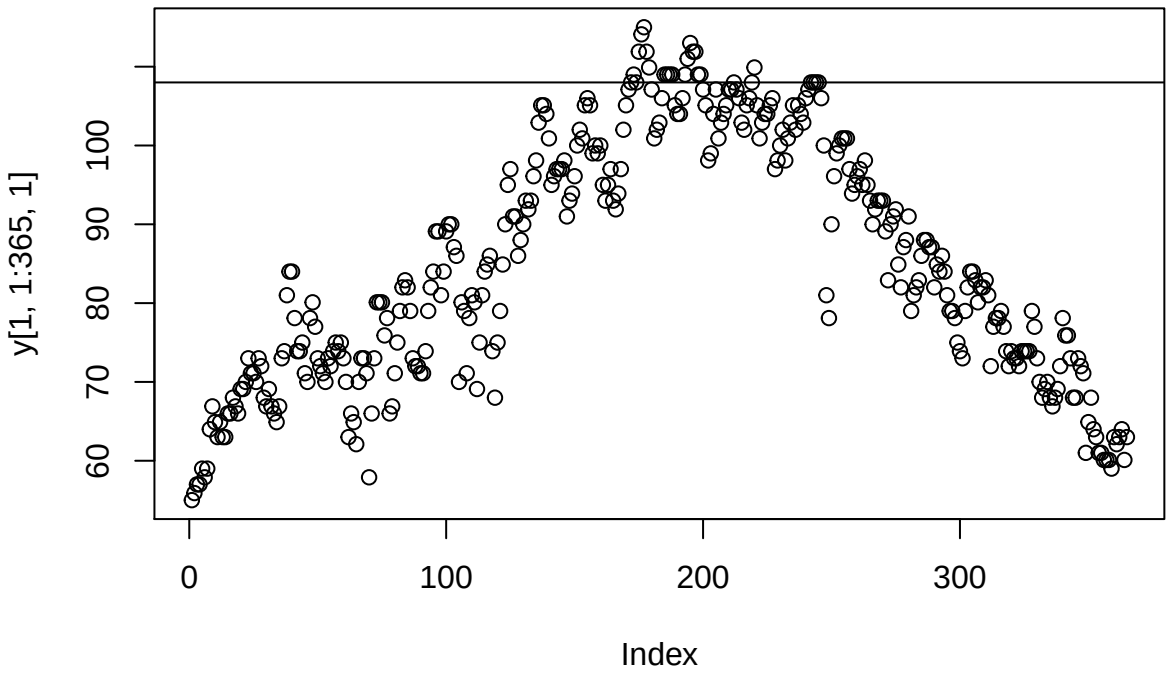
E3

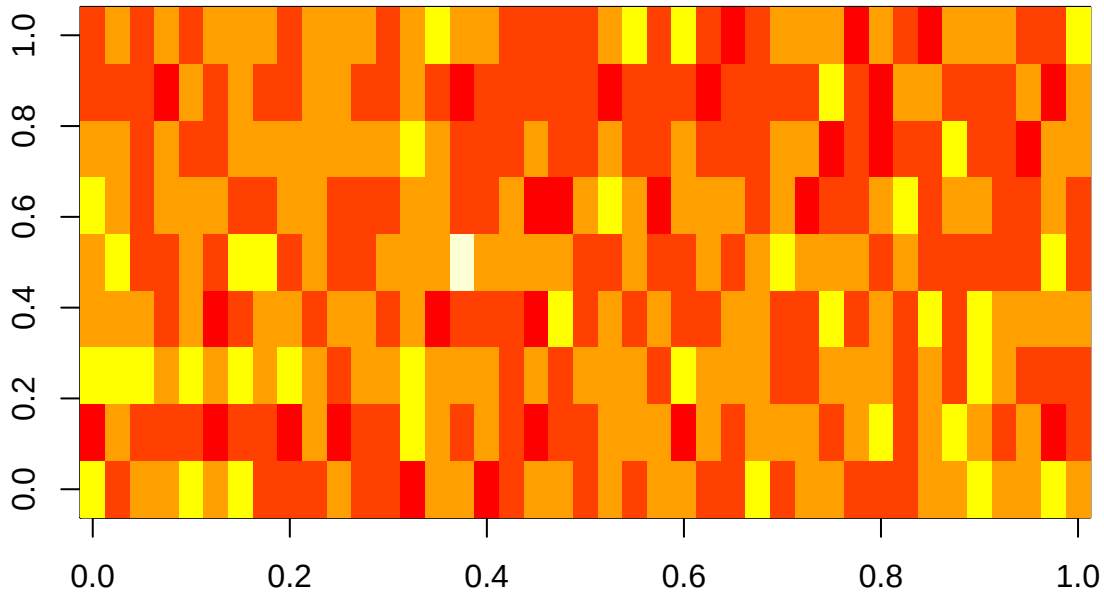
HeatWave W = 3 consecutive days above 95th quantile for that year

```
rm(list = ls())
setwd("c:/e/brucebcampbell-git/bayesian-learning-with-R/E3")
load("heatwaves.RData")

plot(y[1,1:365,1])
abline(h = quantile(y[1,1:365,1],.95))
```

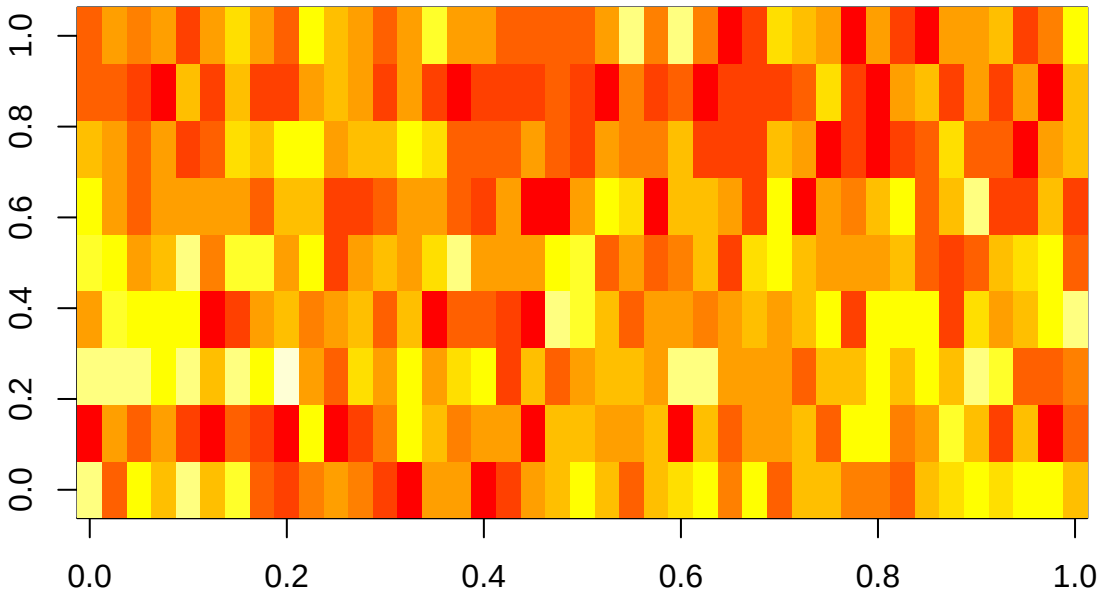



```
#########################################################################
#  HW  = 3 consecutive days above 95th quantile for that year
#########################################################################
X.num <- matrix(nrow =41, ncol = 9 )
X.sev <- matrix(nrow =41, ncol = 9 )

for(i in 1:41)
{
  for(k in 1:9)
  {
    data <- y[i,1:365,k]
```

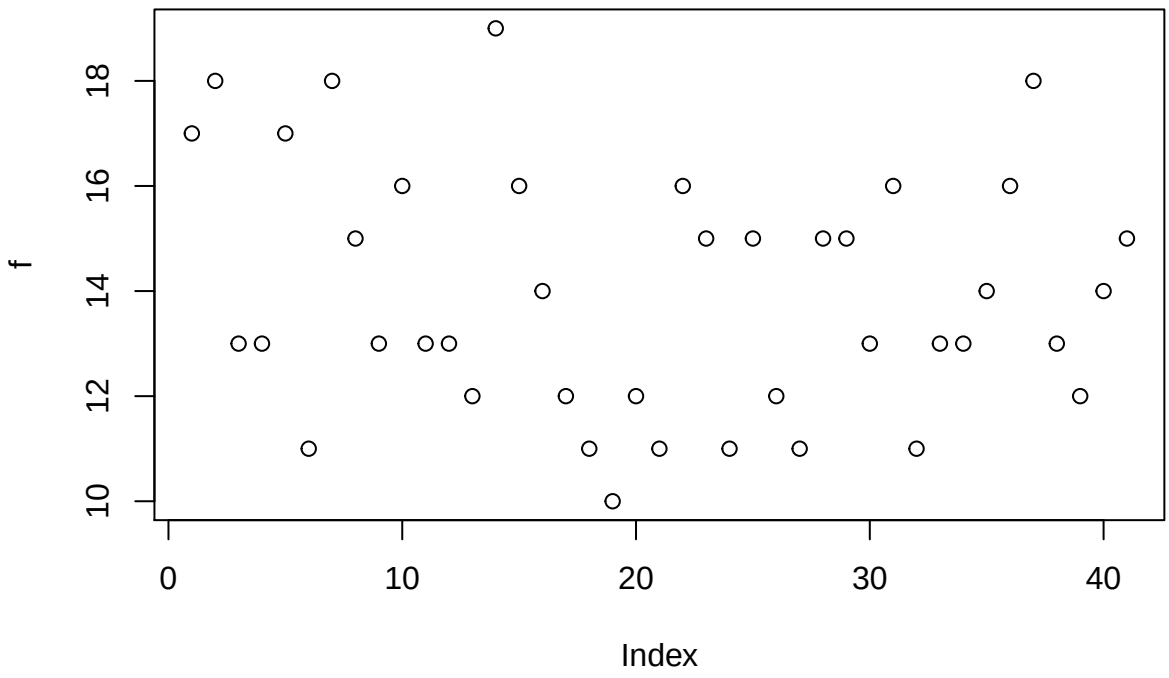
```
    data <- na.exclude(data)
    q_ik <-  quantile(data,.95)
    hot_ik <- data > q_ik
    run_lengths <- rle(hot_ik)
    df_rl <- data.frame(l = run_lengths$lengths,v = run_lengths$values)
    hot_groups <- df_rl[df_rl$v==TRUE,]
    heat_waves <- hot_groups[hot_groups$l>3,]
    num_hw <- nrow(heat_waves)
    sev_hw <- sum(heat_waves$l)

    X.num[i,k] <- num_hw
    X.sev[i,k] <- sev_hw
  }

}

image(X.num)
```



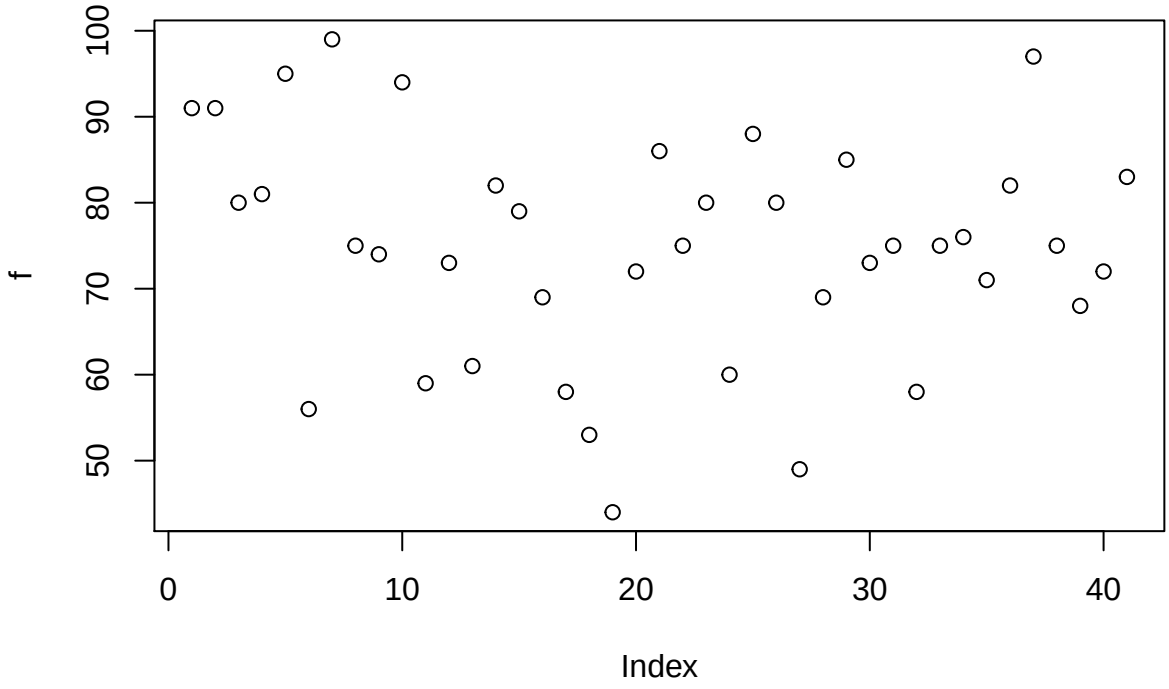

```
image(X.sev)
```



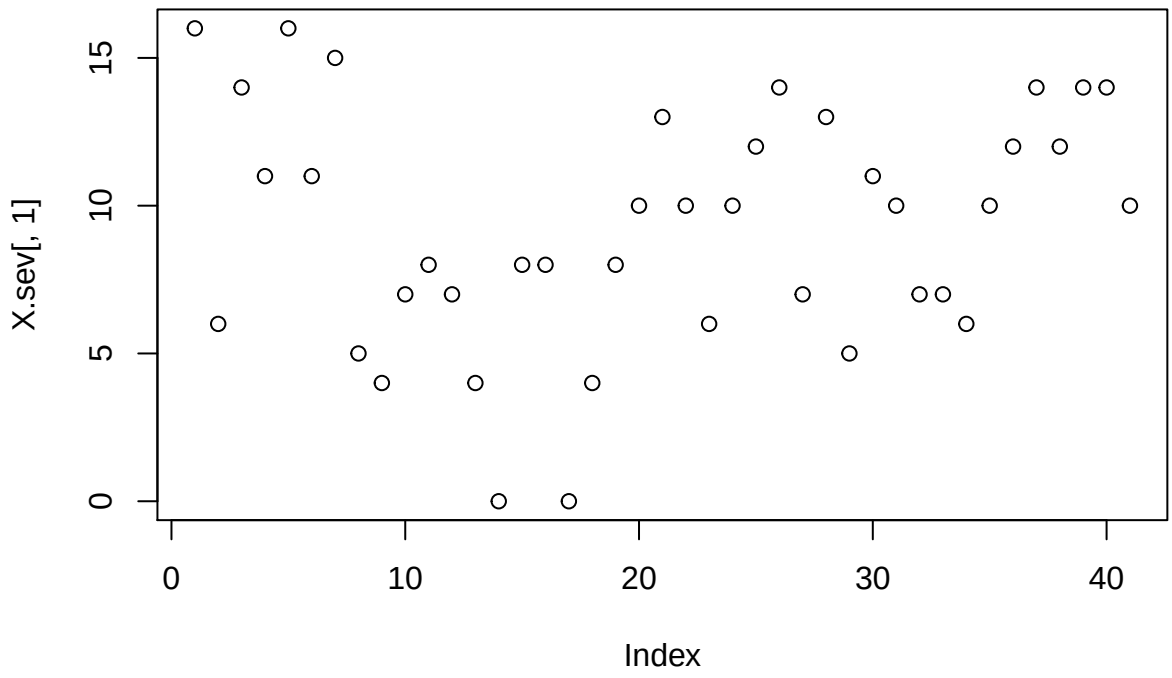

```
f<- rowSums(X.num)
plot(f)
```



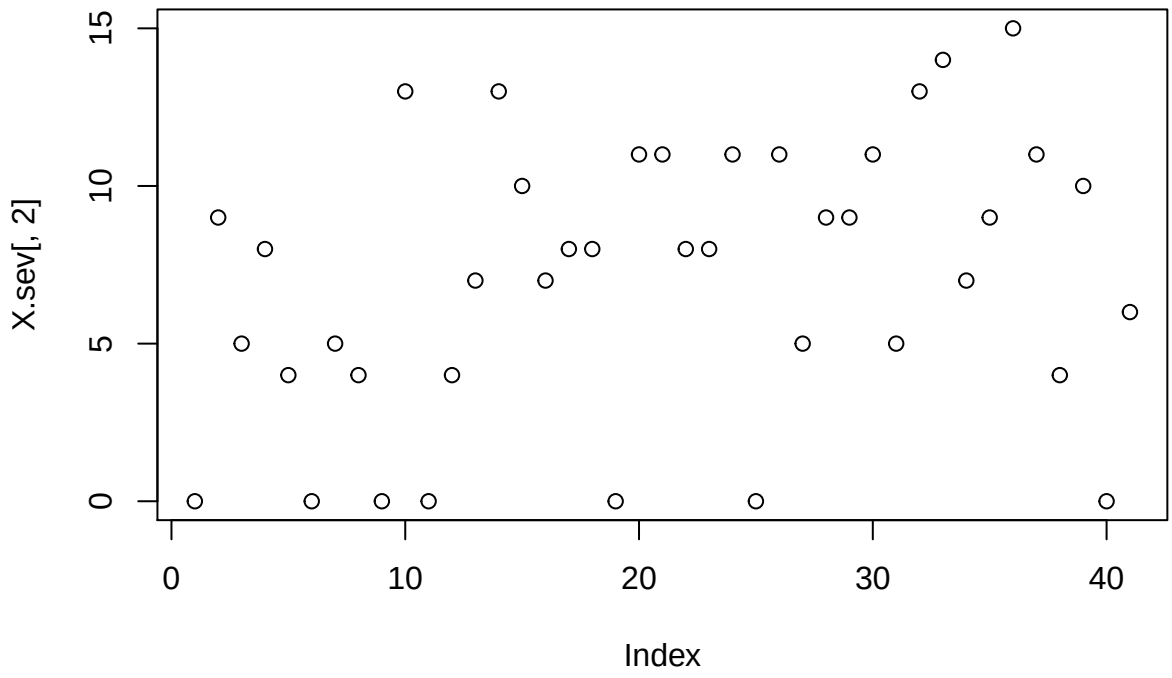

```
f<- rowSums(X.sev)
plot(f)
```



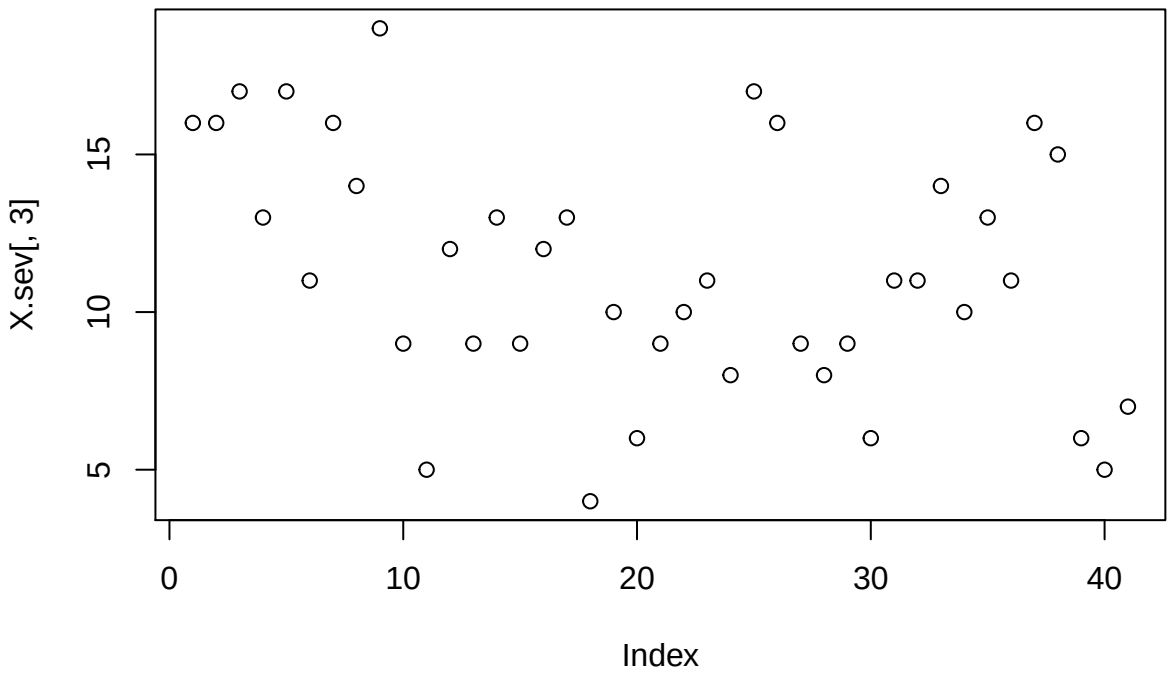

```
plot(X.sev[,1])
```



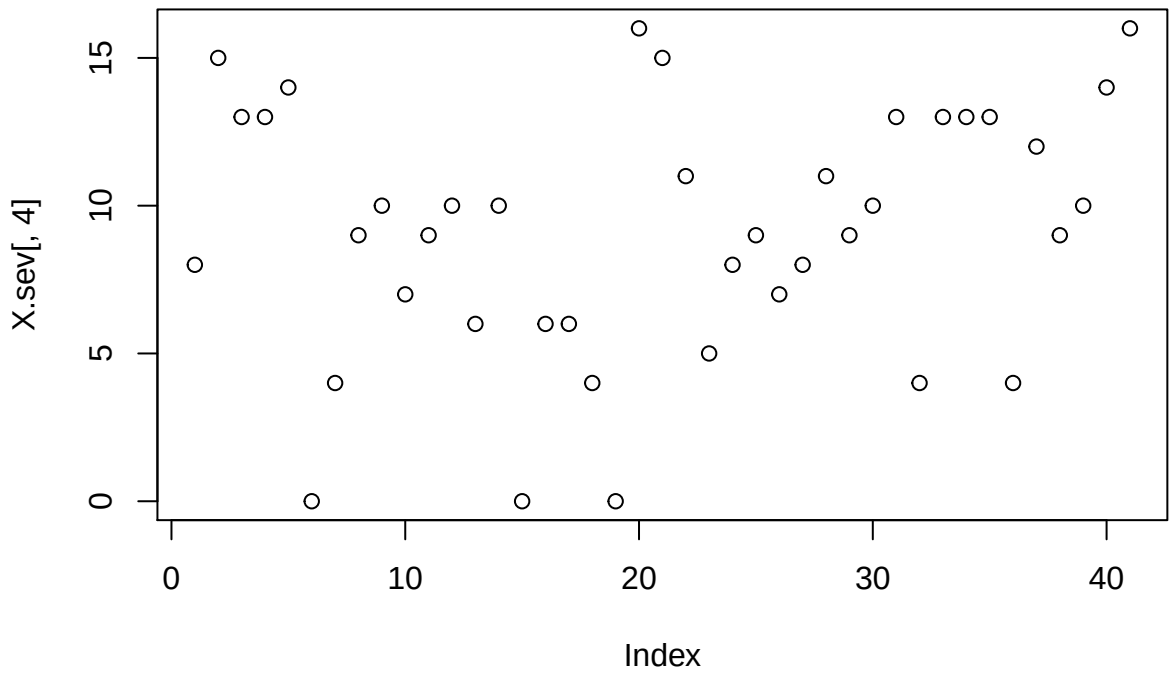

```
plot(X.sev[,2])
```



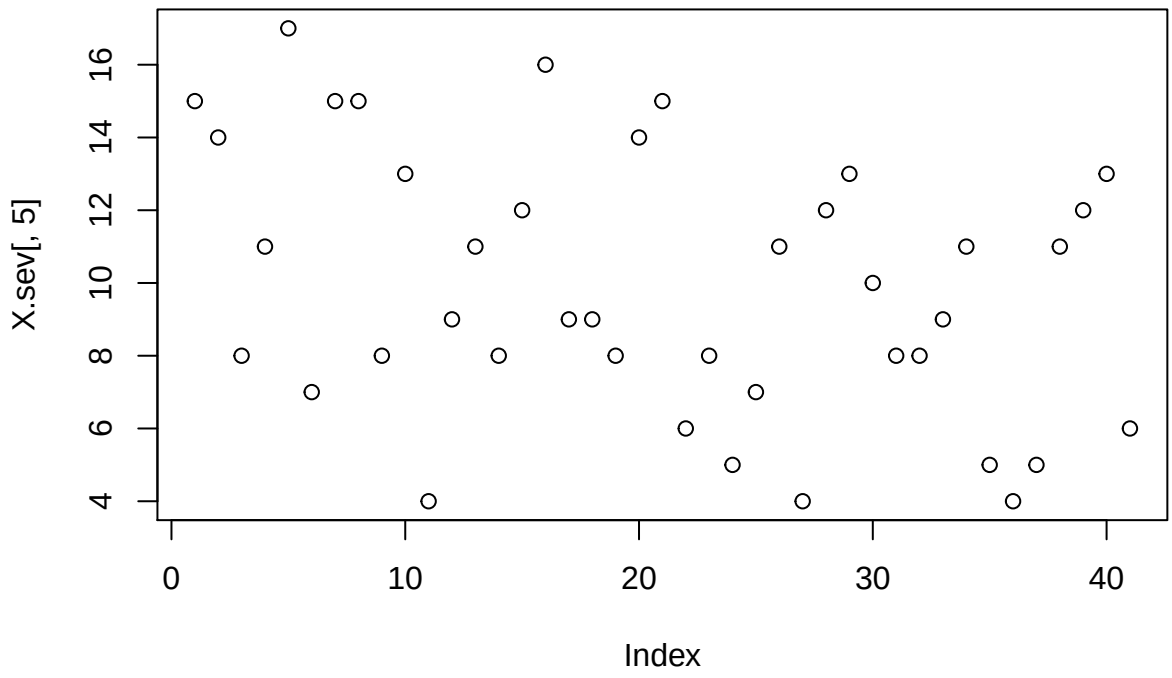

```
plot(X.sev[,3])
```



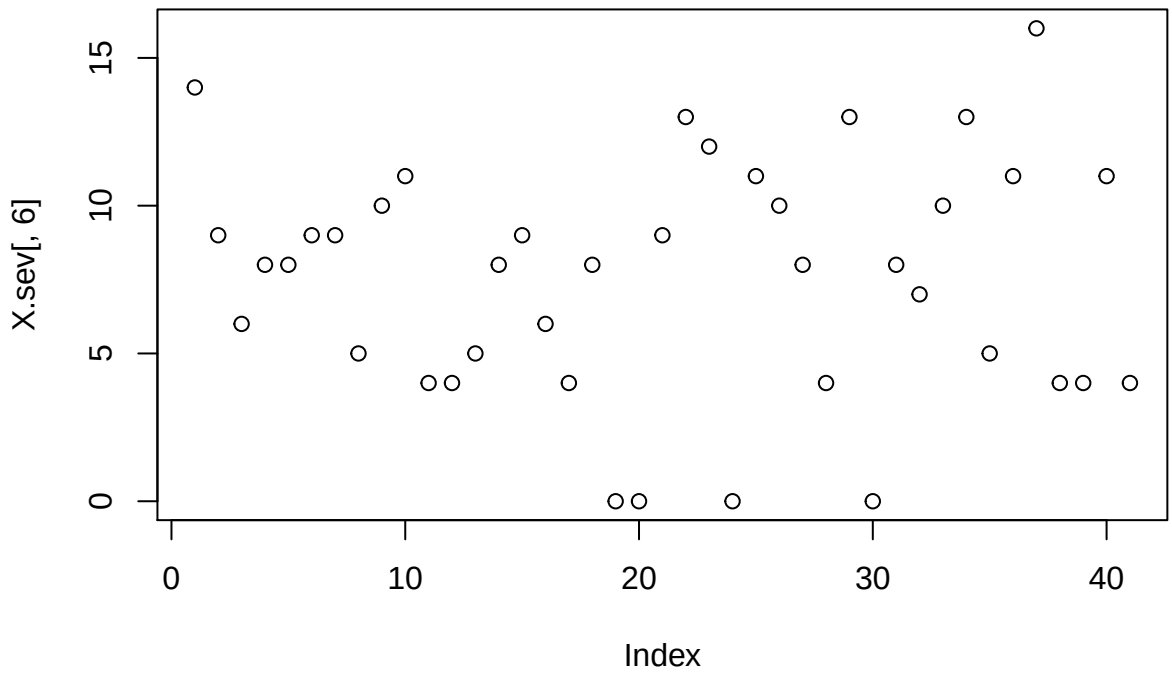

```
plot(X.sev[,4])
```



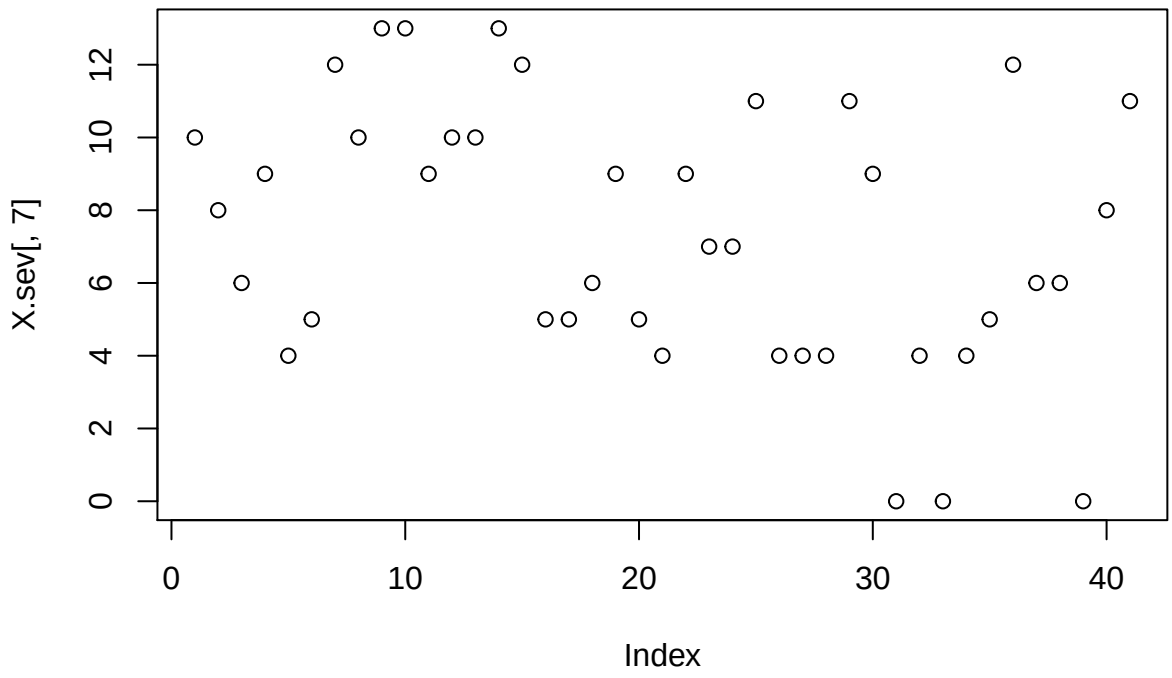

```
plot(X.sev[,5])
```



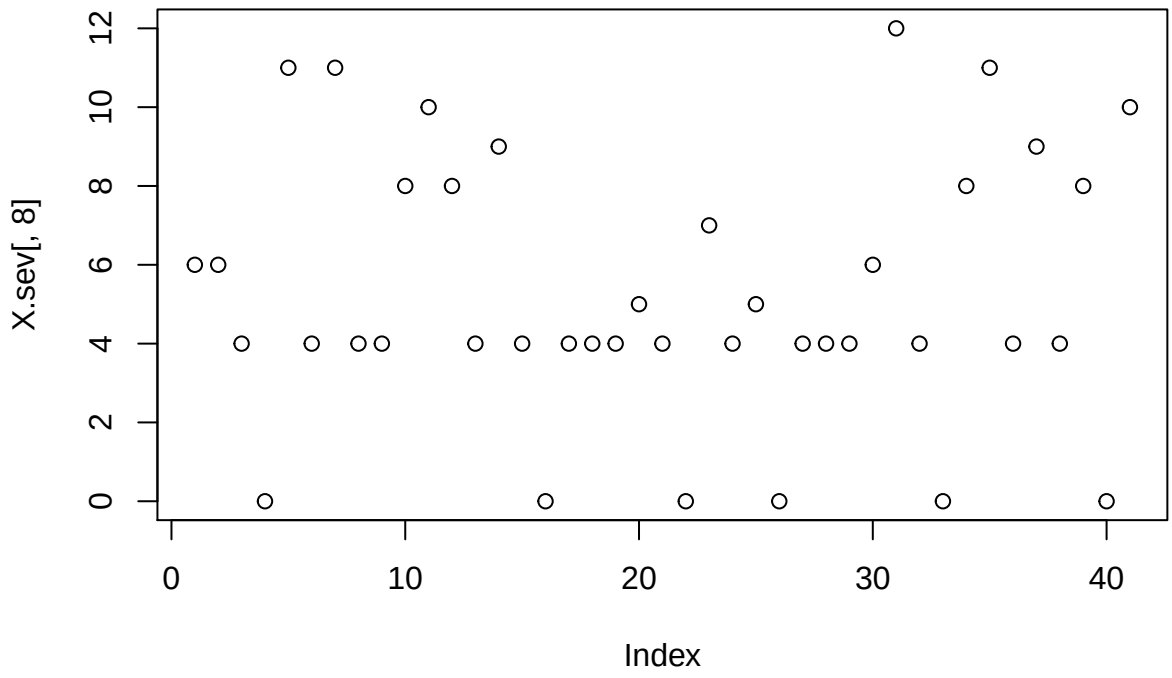

```
plot(X.sev[,6])
```



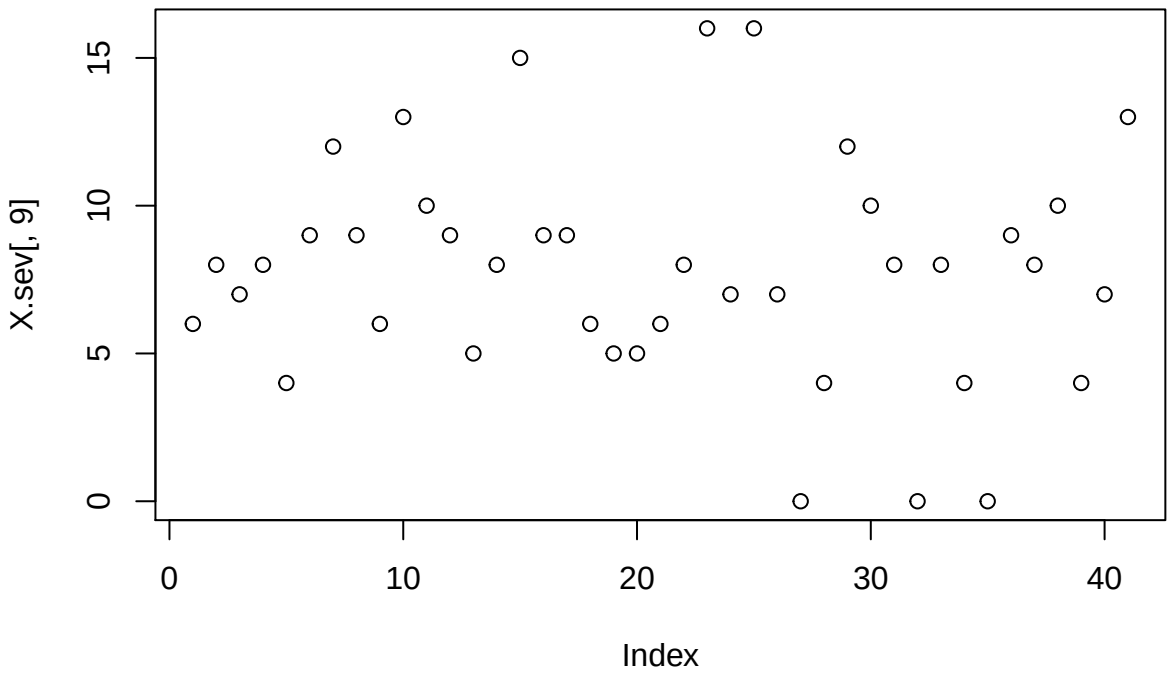

```
plot(X.sev[,7])
```




```
plot(X.sev[,8])
```



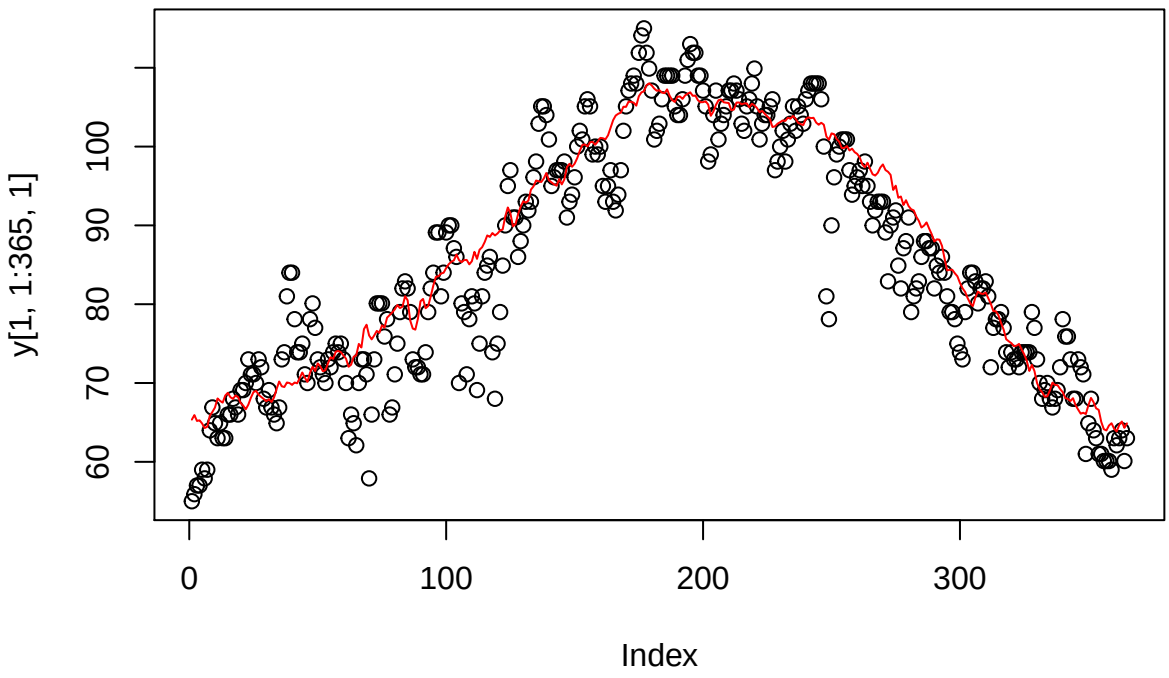

```
plot(X.sev[,9])
```



HeatWave = World Meteorological Organization definition

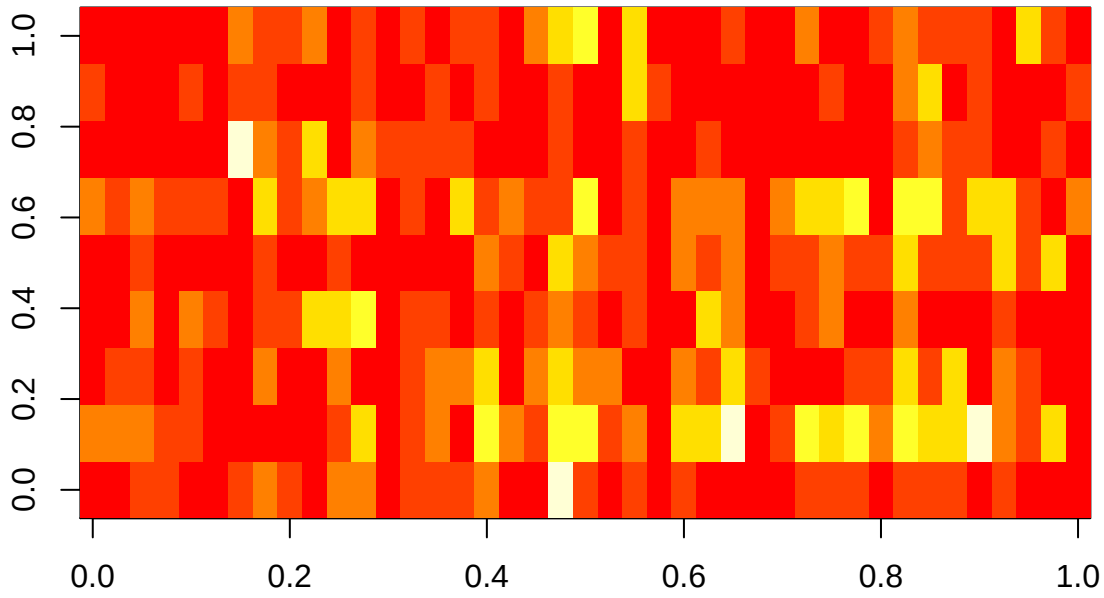
daily temperature for more than five consecutive days exceeds the average temperature by 9 degrees

```
#########################################################################
#  HW  = World Meteorological Organization definition
#  daily temperature for more than five consecutive days exceeds the average temperature by 9 degrees
#########################################################################
X.num <- matrix(nrow =41, ncol = 9 )
X.sev <- matrix(nrow =41, ncol = 9 )

#Daily Averages
DA <- matrix(nrow = 365,ncol = 9)
for(j in 1:365)
{
  for(k in 1:9)
  {
    data <- y[1:41,j,k]
    data <- na.exclude(data)
    DA[j,k] <- mean(data)
  }
}
plot(y[1,1:365,1]); lines(DA[,1],col='red')
```

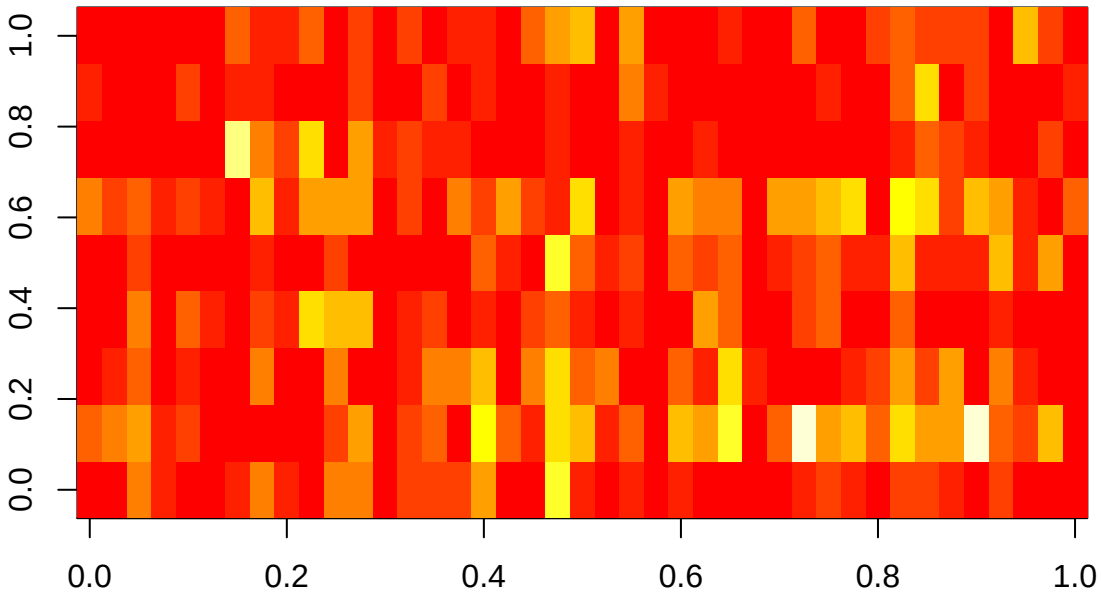



```
AboveDA.Idx <- y
for(i in 1:41)
{
  for(k in 1:9)
  {
    for(j in 1:365)
    {
      t <- y[i,j,k]
      A <- DA[j,k]
      isHot <- t >=A+9
      AboveDA.Idx[i,j,k] <- isHot
    }
  }
}

X.num <- matrix(nrow =41, ncol = 9 )
X.sev <- matrix(nrow =41, ncol = 9 )

for(i in 1:41)
{
  for(k in 1:9)
  {
    data <- AboveDA.Idx[i,1:365,k]
    data <- na.exclude(data)
    run_lengths <- rle(as.vector(data))
    df_rl <- data.frame(l = run_lengths$lengths,v = run_lengths$values)
```

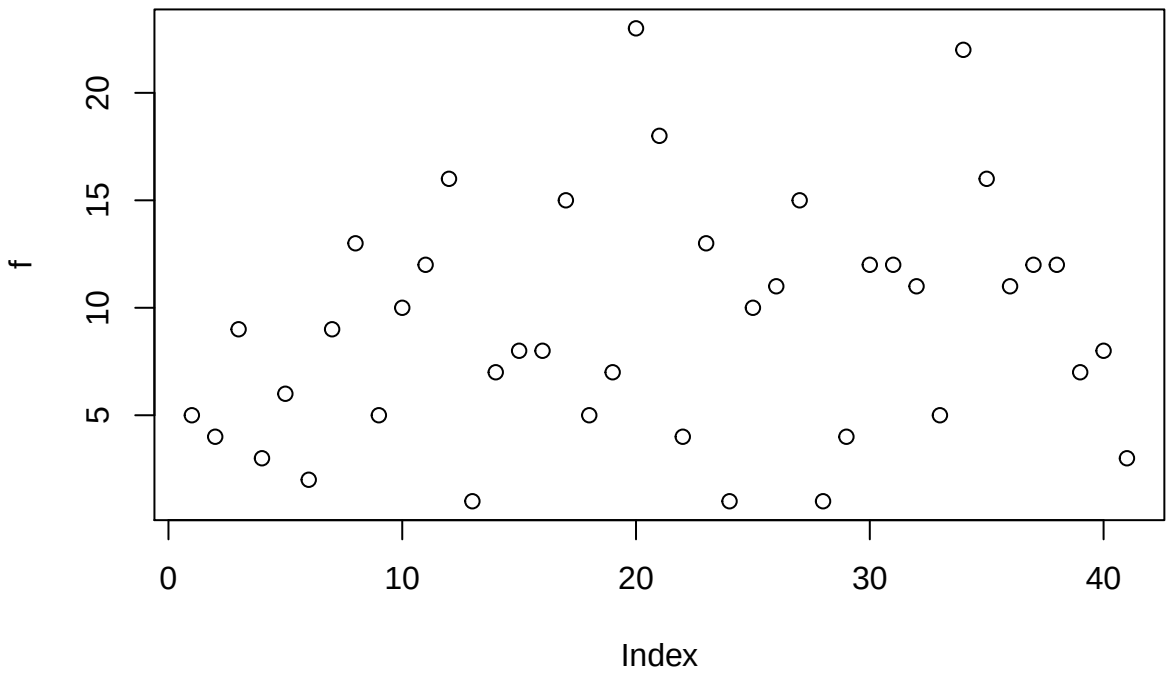
```
    hot_groups <- df_rl[df_rl$v==TRUE,]
    heat_waves <- hot_groups[hot_groups$l>5,]
    num_hw <- nrow(heat_waves)
    sev_hw <- sum(heat_waves$l)

    X.num[i,k] <- num_hw
    X.sev[i,k] <- sev_hw
  }

}

image(X.num)
```



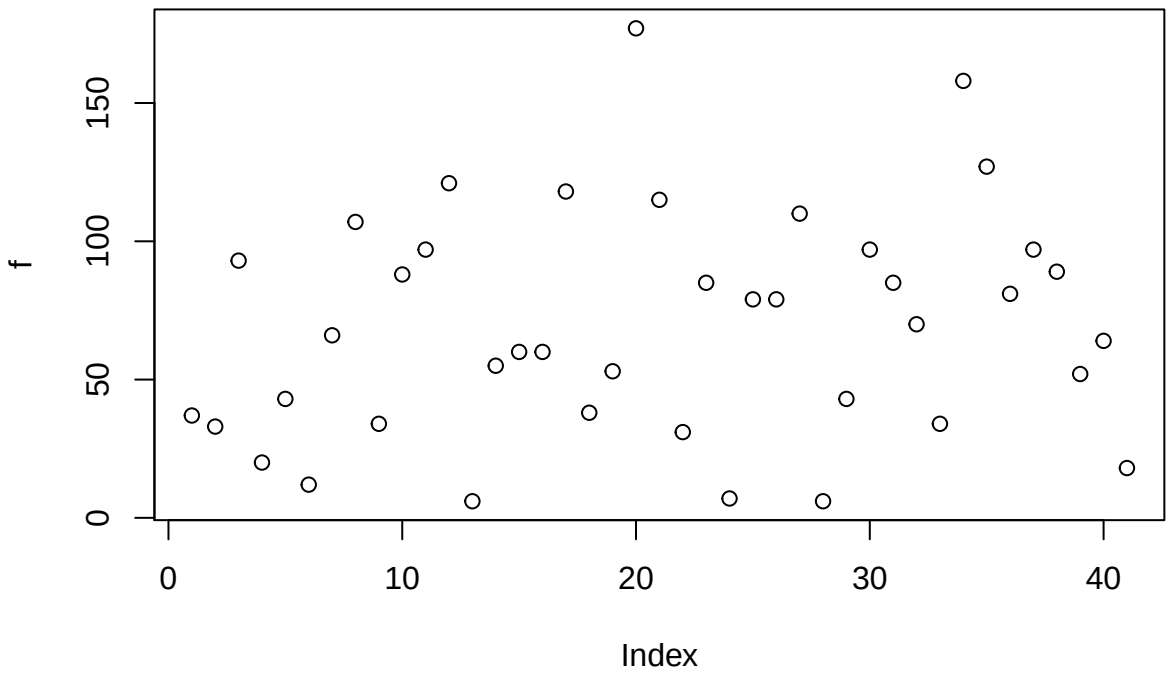

```
image(X.sev)
```



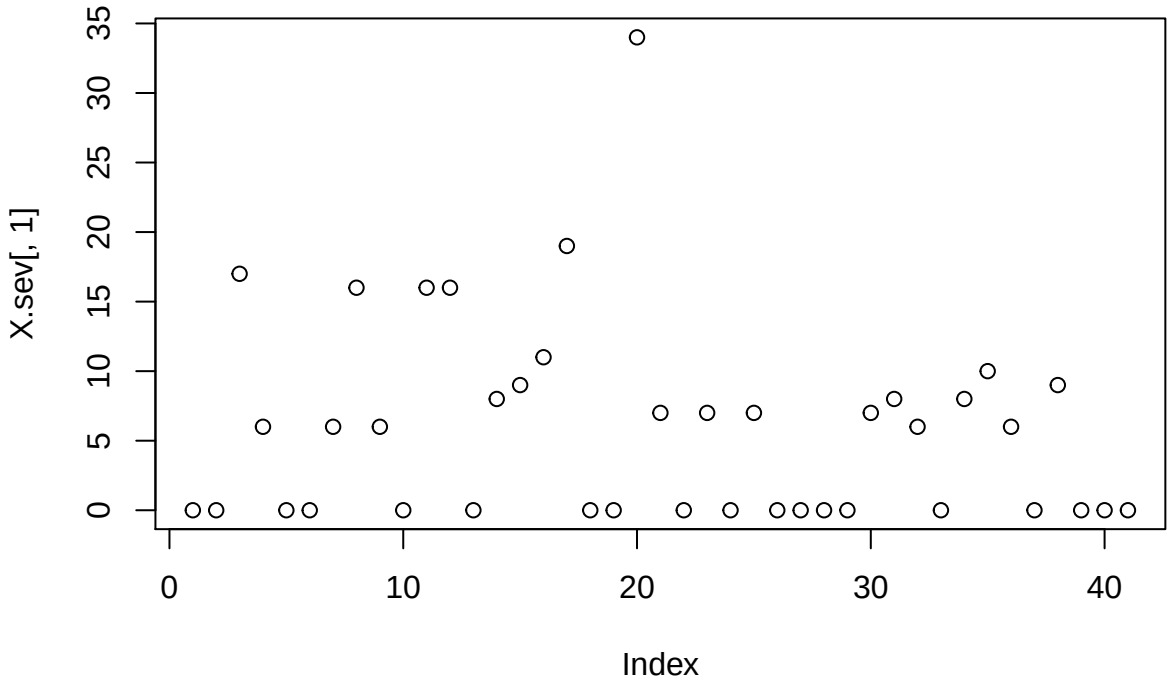

```
f<- rowSums(X.num)
plot(f)
```



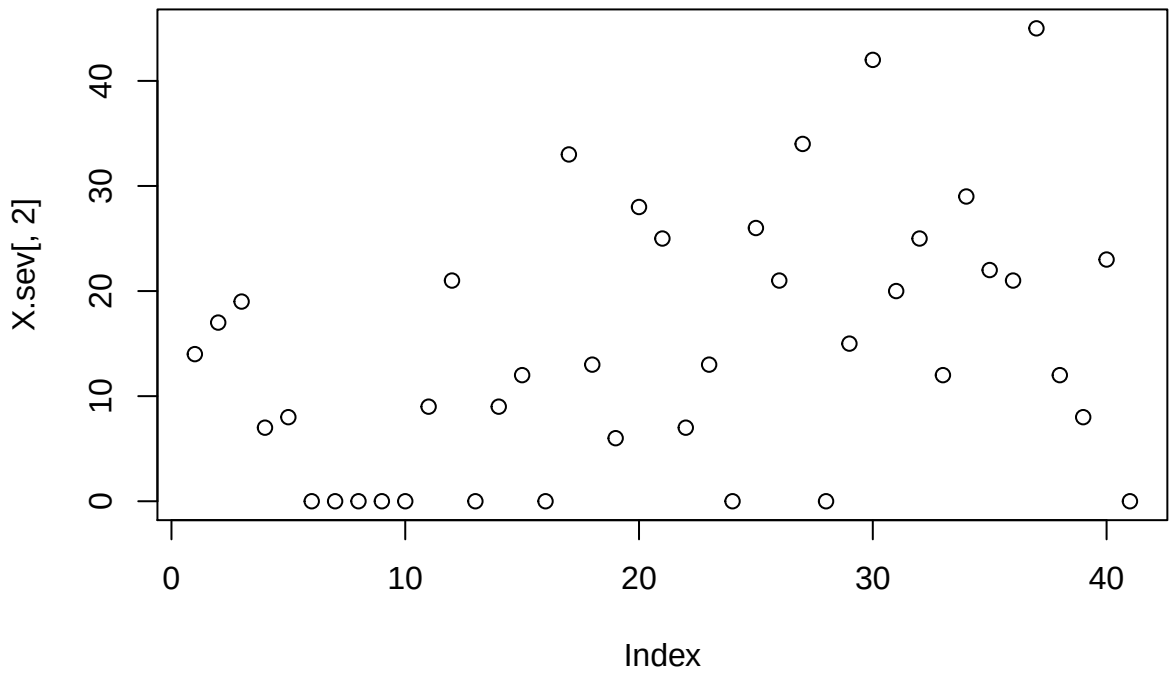

```
f<- rowSums(X.sev)
plot(f)
```



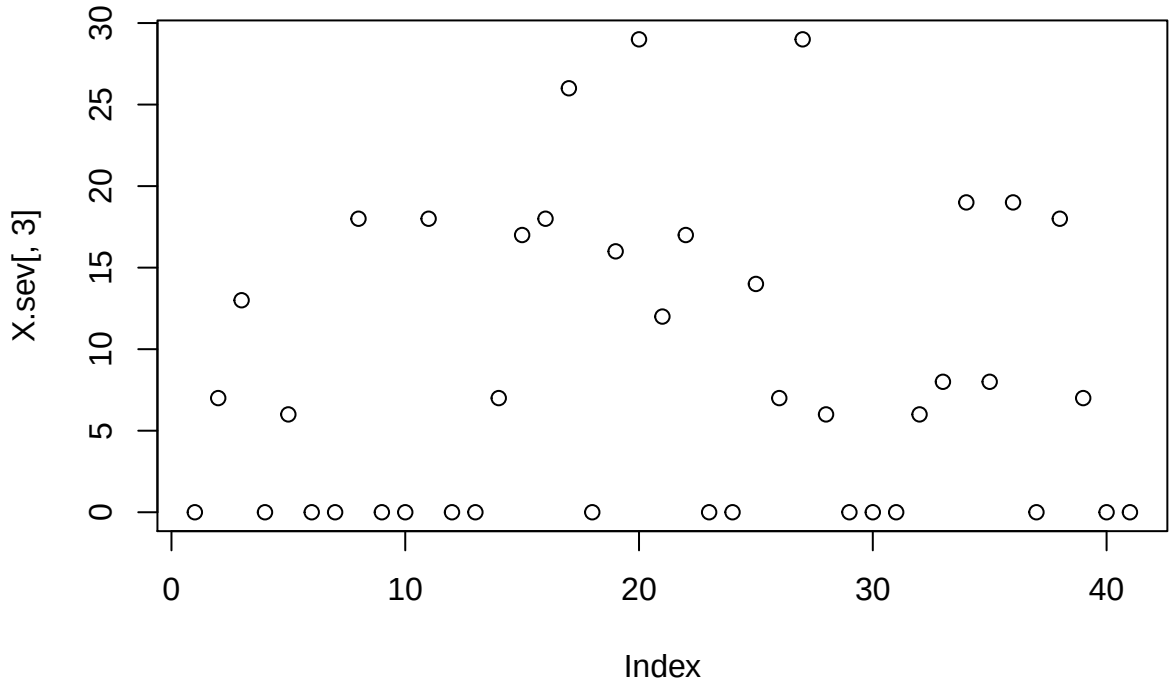

```
plot(X.sev[,1])
```



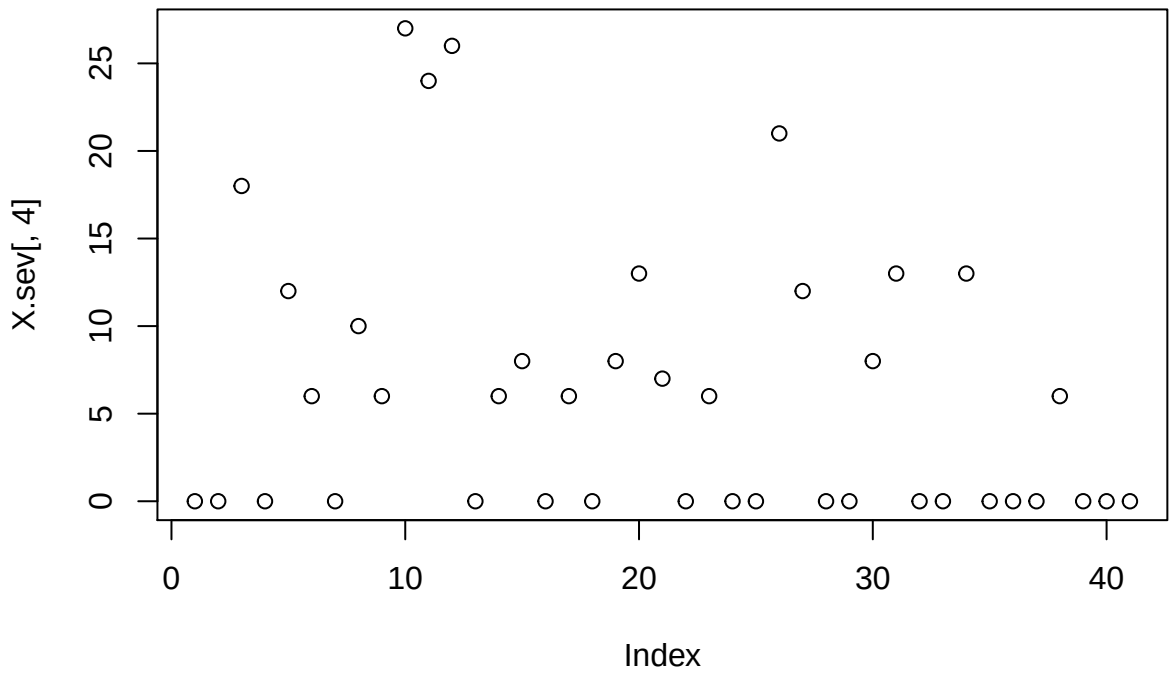

```
plot(X.sev[,2])
```



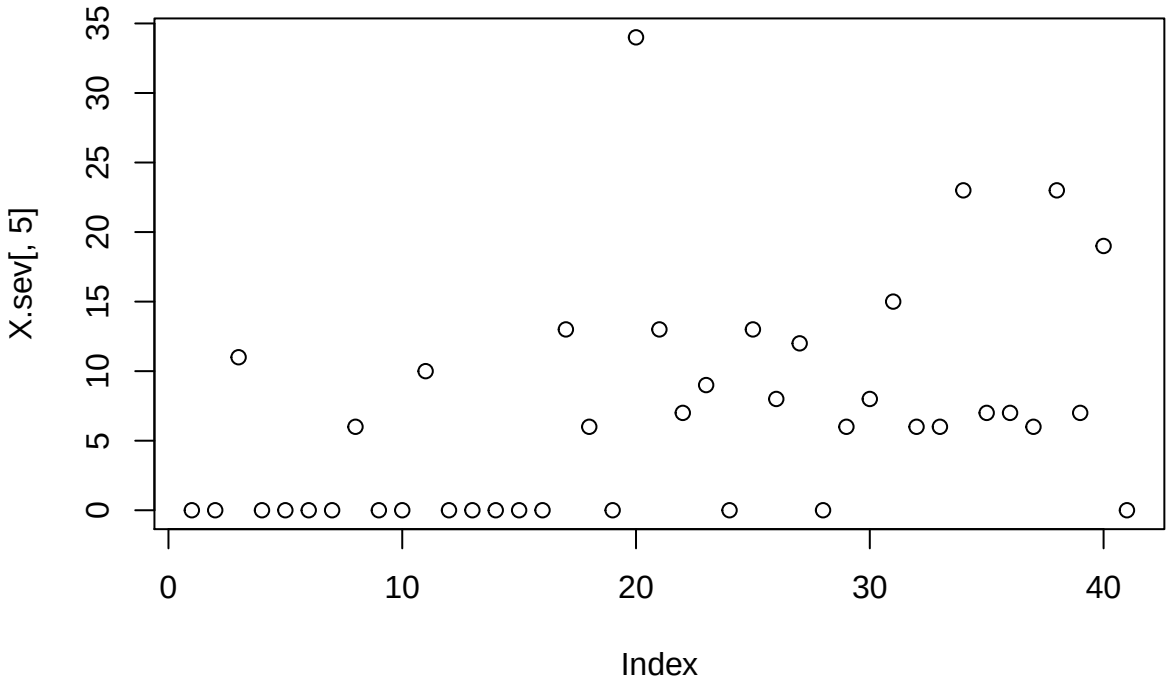

```
plot(X.sev[,3])
```



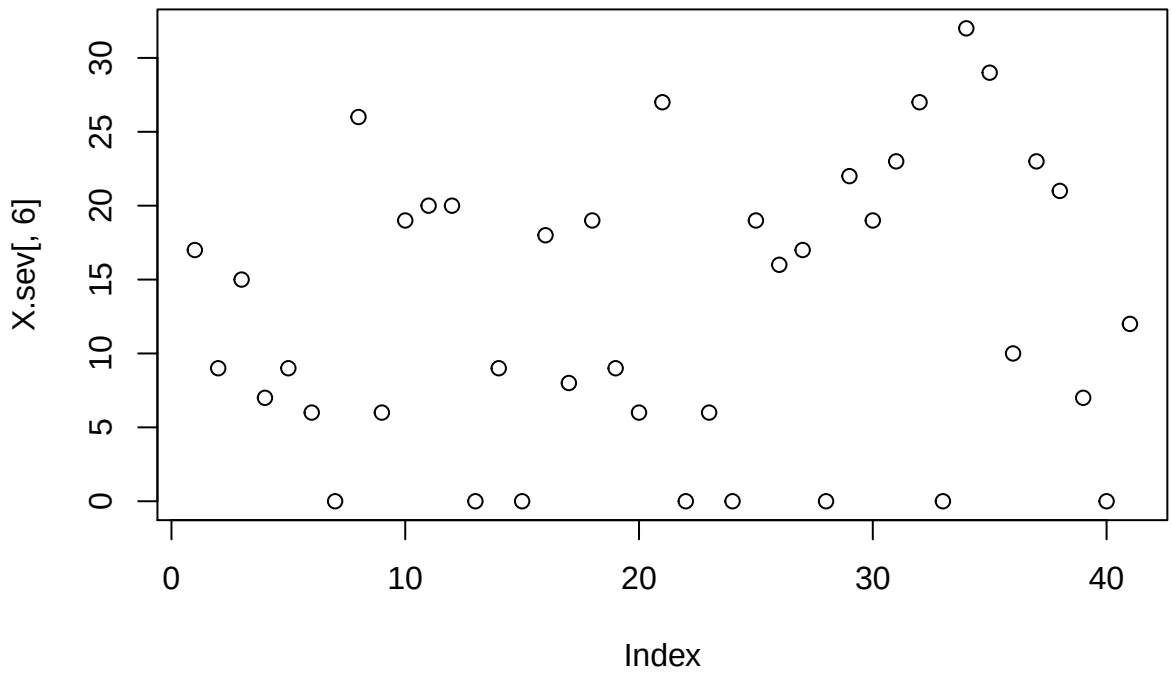

```
plot(X.sev[,4])
```



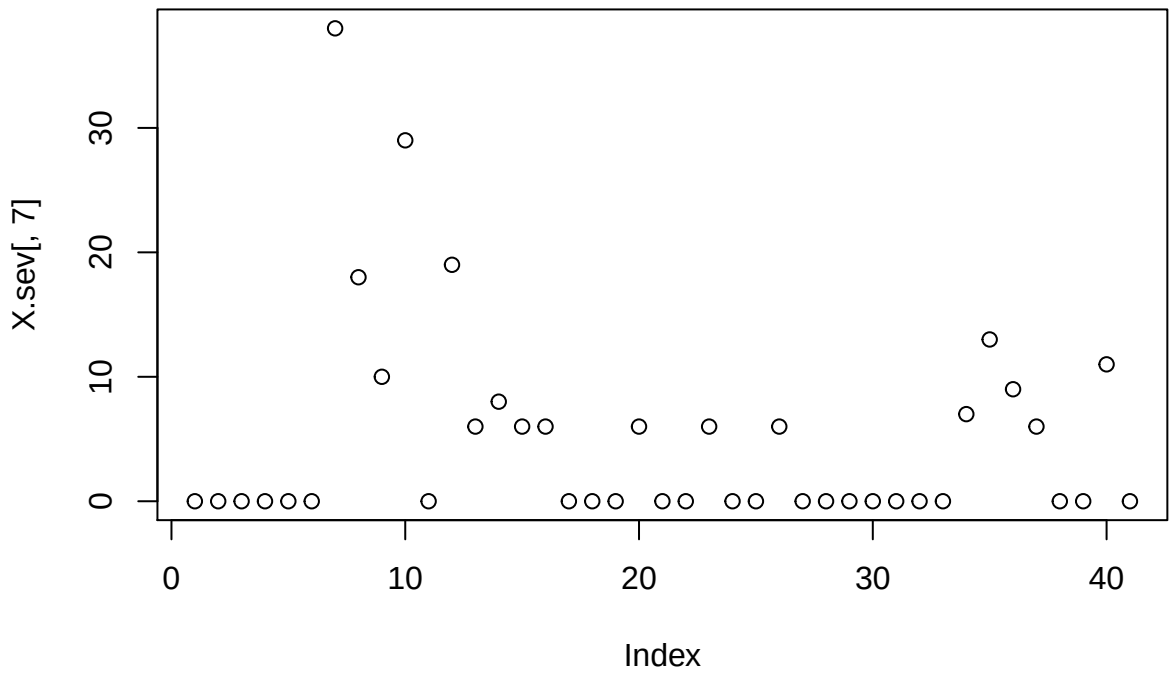

```
plot(X.sev[,5])
```



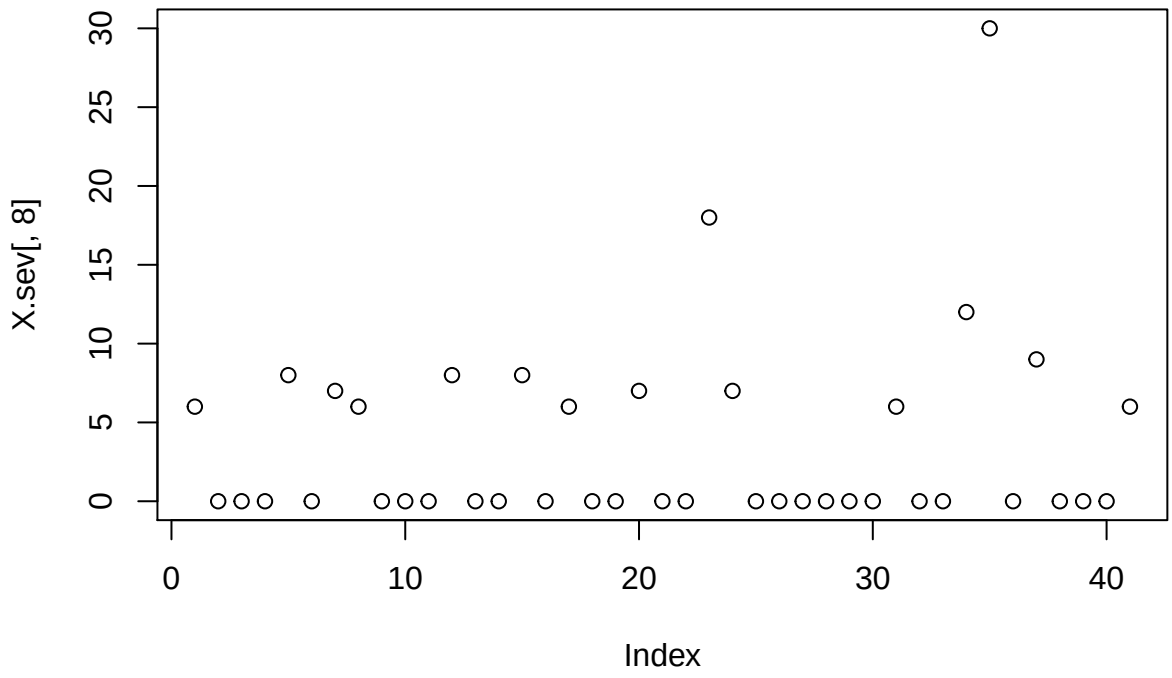

```
plot(X.sev[,6])
```



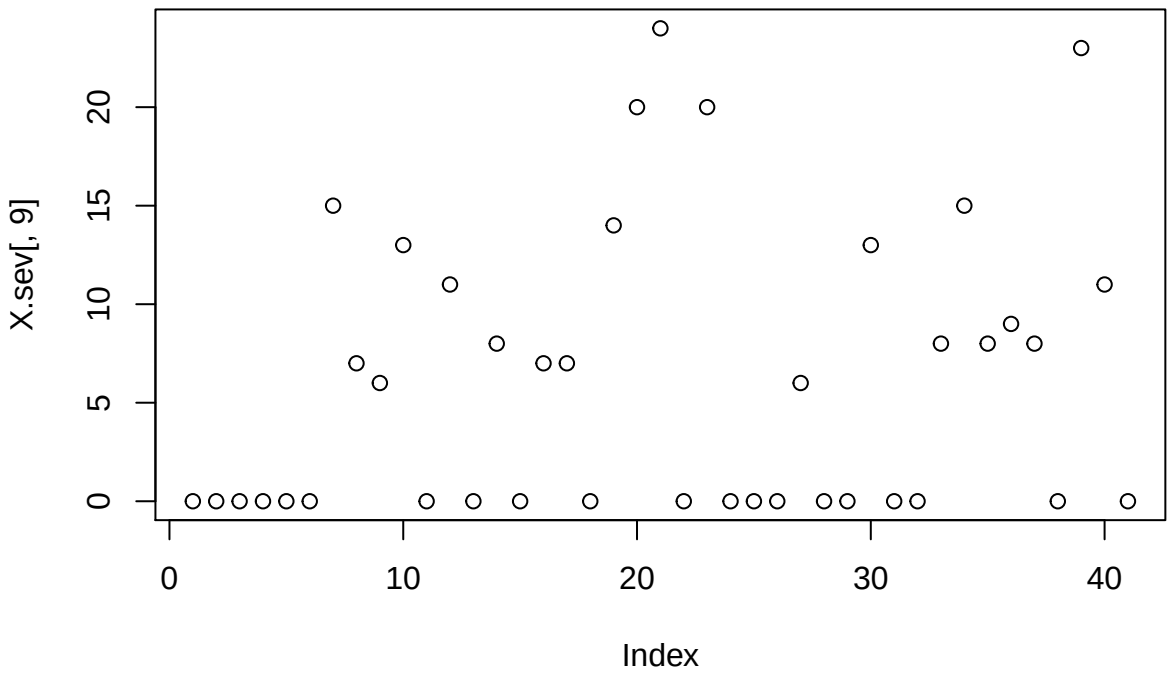

```
plot(X.sev[,7])
```




```
plot(X.sev[,8])
```



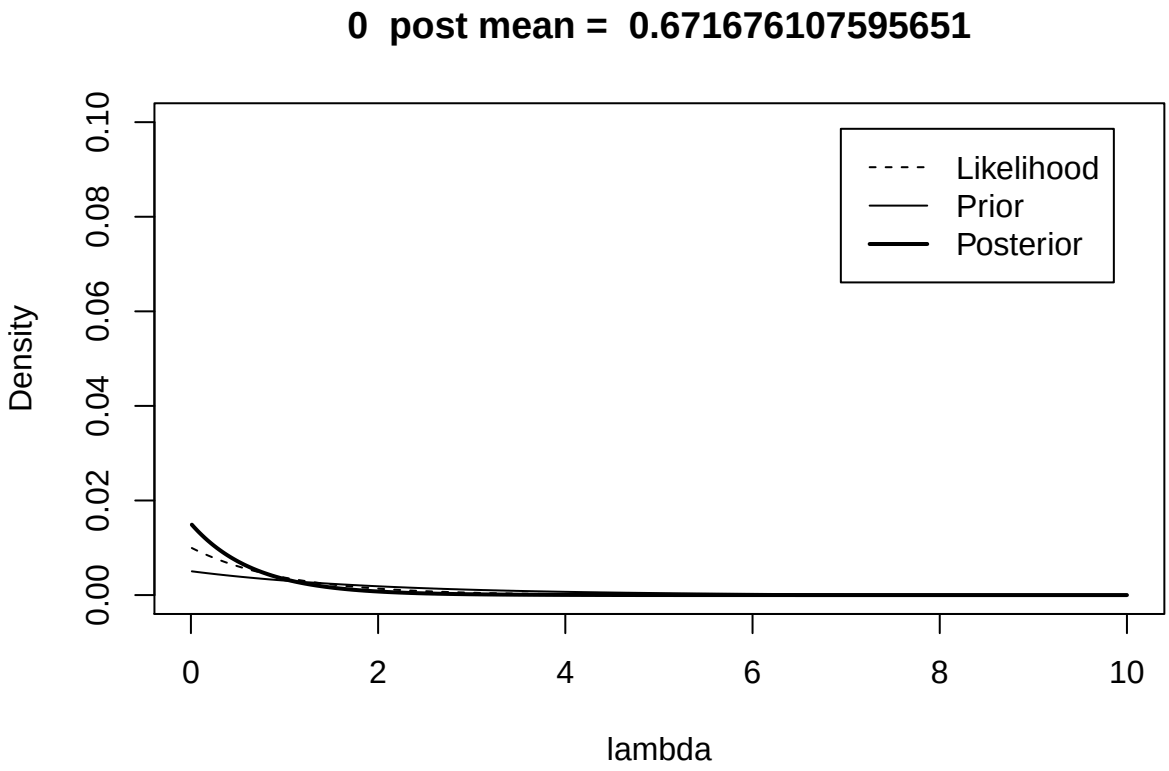

```
plot(X.sev[,9])
```

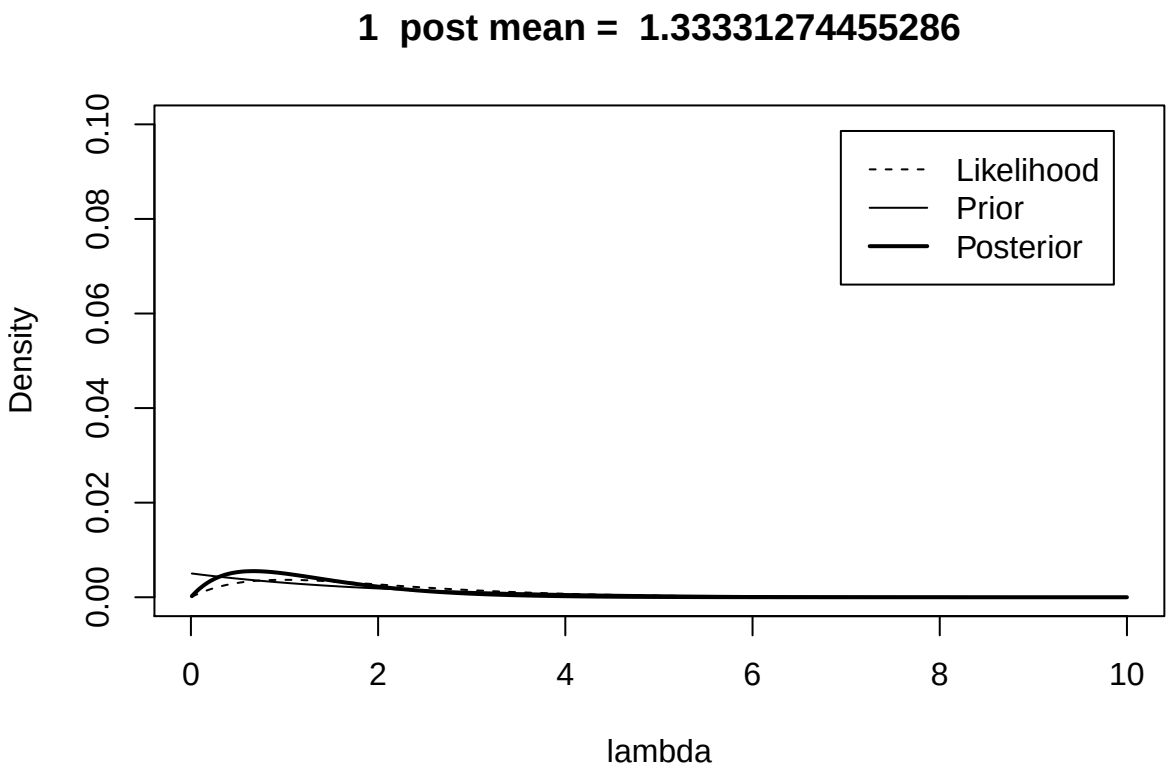


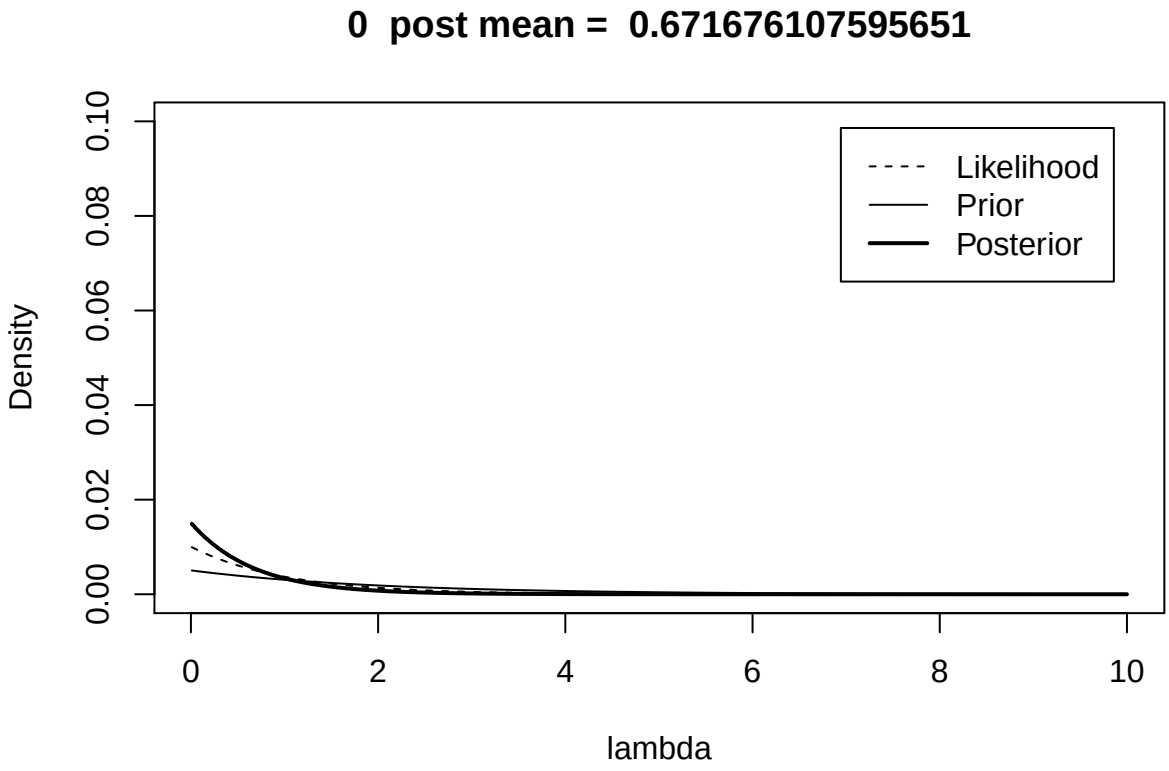
Plot the prior, likelihood, and posterior on a grid

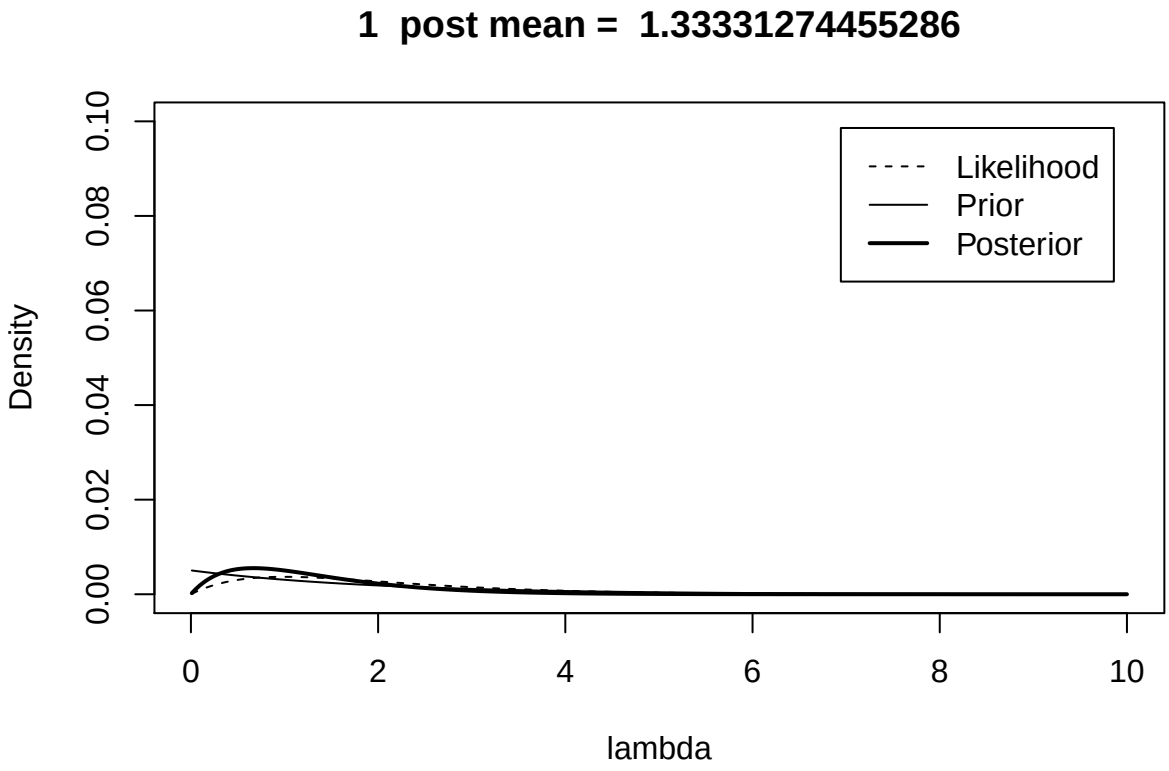
```
################################
#Plot the prior, likelihood, and posterior on a grid
k=1
for(i in 1:41)
{
  N      <- 1
  Y      <- X.num[i,k]
  a      <- 1
  b      <- 0.5
  grid   <- seq(0.01,10,.01)
  like   <- dpois(Y,N*grid)
  like   <- like/sum(like) #standardize
  prior  <- dgamma(grid,a,b)
  prior  <- prior/sum(prior) #standardize
  post   <- like*prior
  post   <- post/sum(post)
  ps.mean <-sum(post*grid)
  plot(grid,like,type="l",lty=2,
       xlab="lambda",ylab="Density",main=paste(Y," post mean = ",ps.mean,sep = ' '), ylim=c(0,.1))
   lines(grid,prior)
  lines(grid,post,lwd=2)
  legend("topright",c("Likelihood","Prior","Posterior"),lwd=c(1,1,2),lty=c(2,1,1),inset=0.05)
```

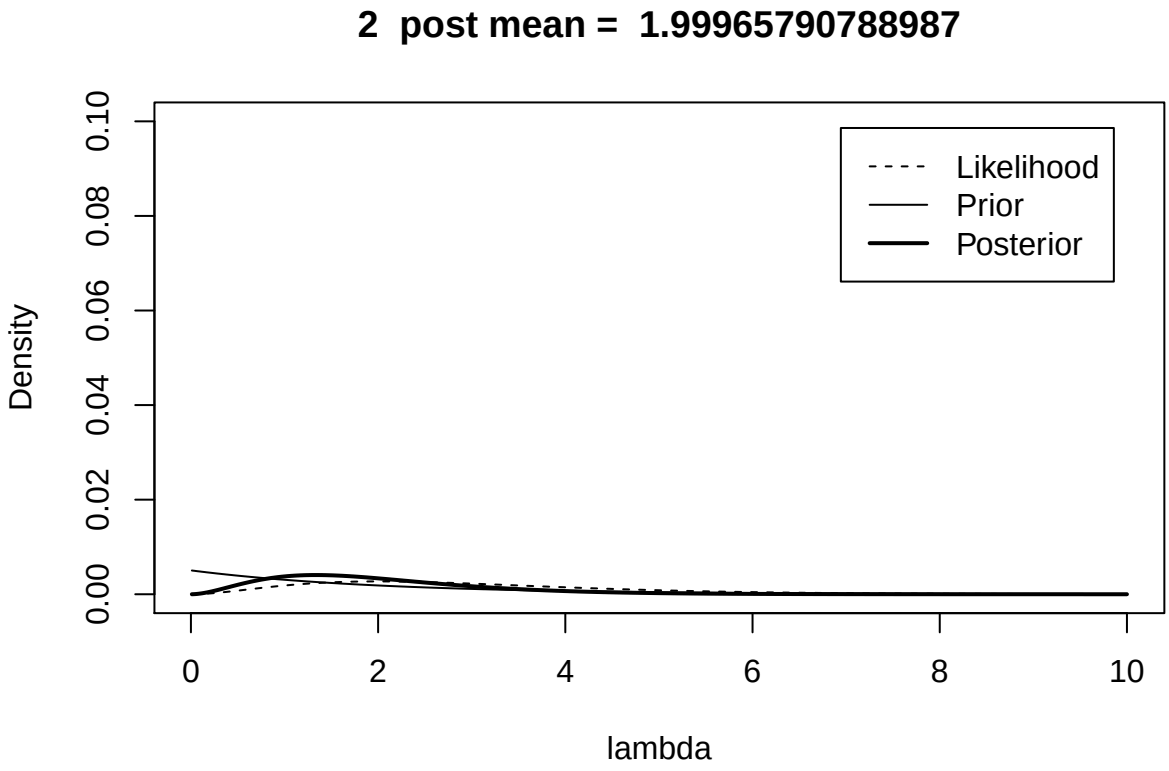
```
}
```

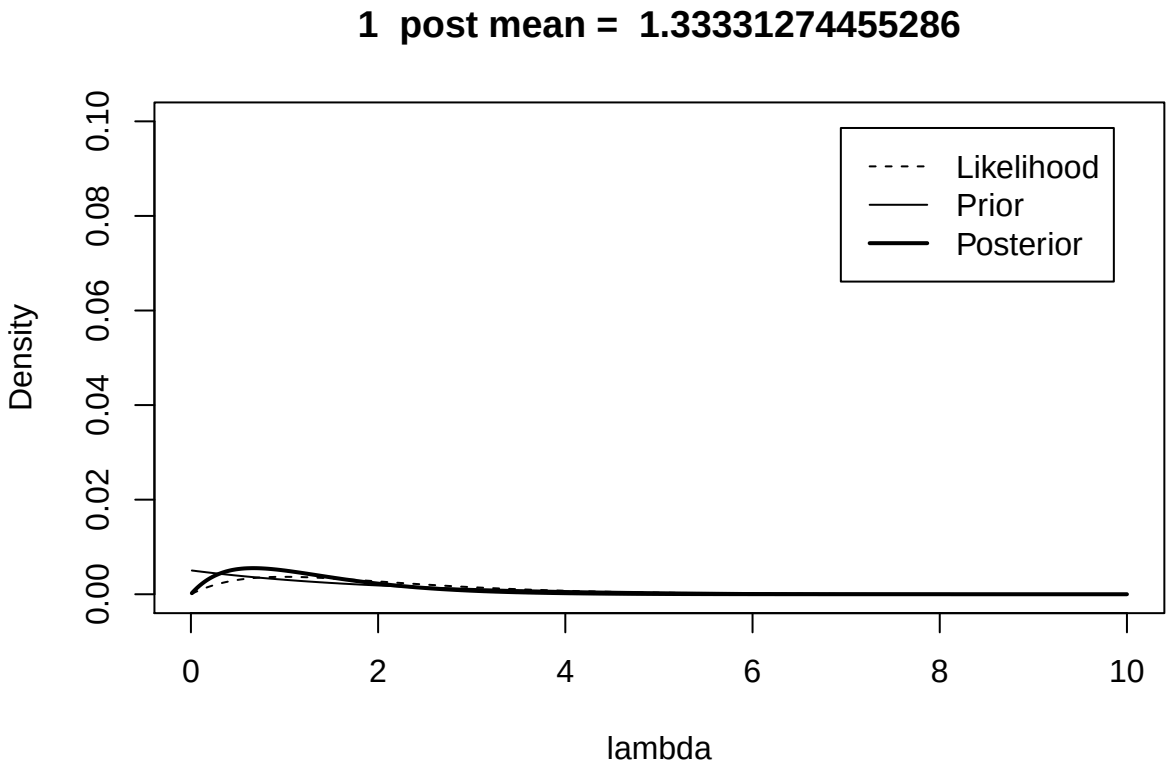

0 post mean = 0.671676107595651
Density
0.00
0.02
0.04
0.06
0.08
0.10
0
2
4
6
8
10
lambda
Likelihood
Prior
Posterior

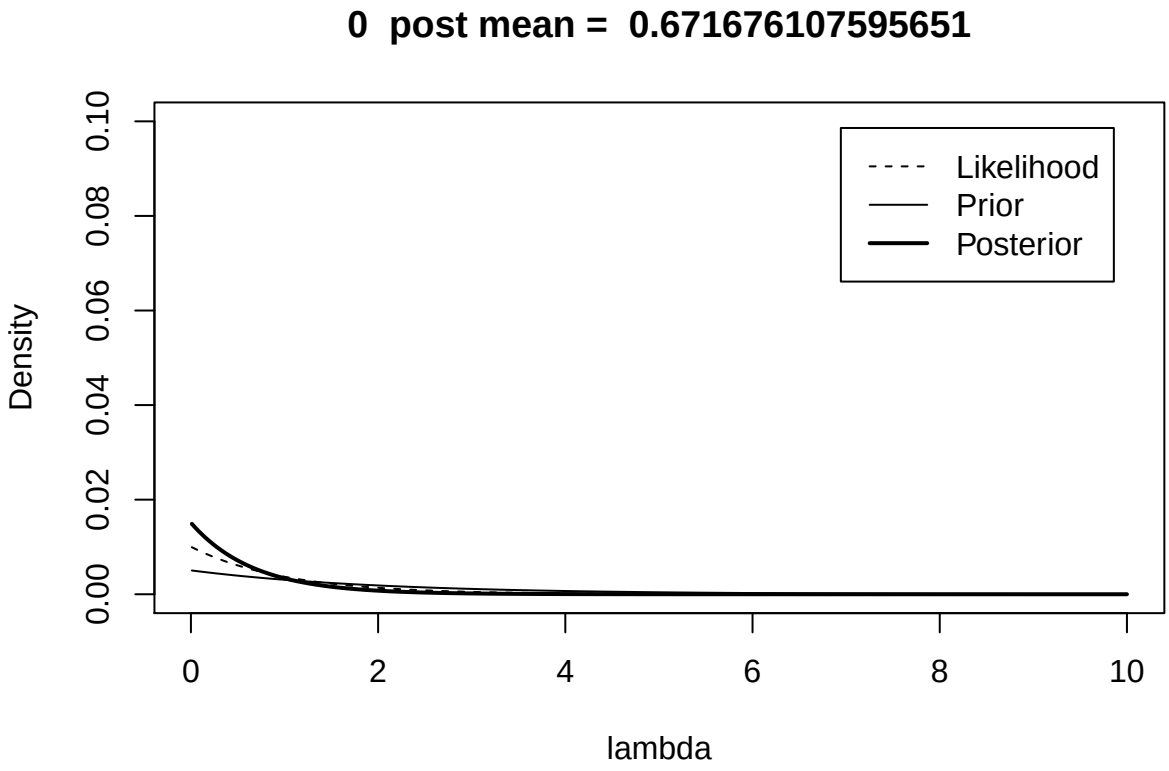

1 post mean = 1.33331274455286
Density
0.00
0.02
0.04
0.06
0.08
0.10
0
2
4
6
8
10
lambda
Likelihood
Prior
Posterior

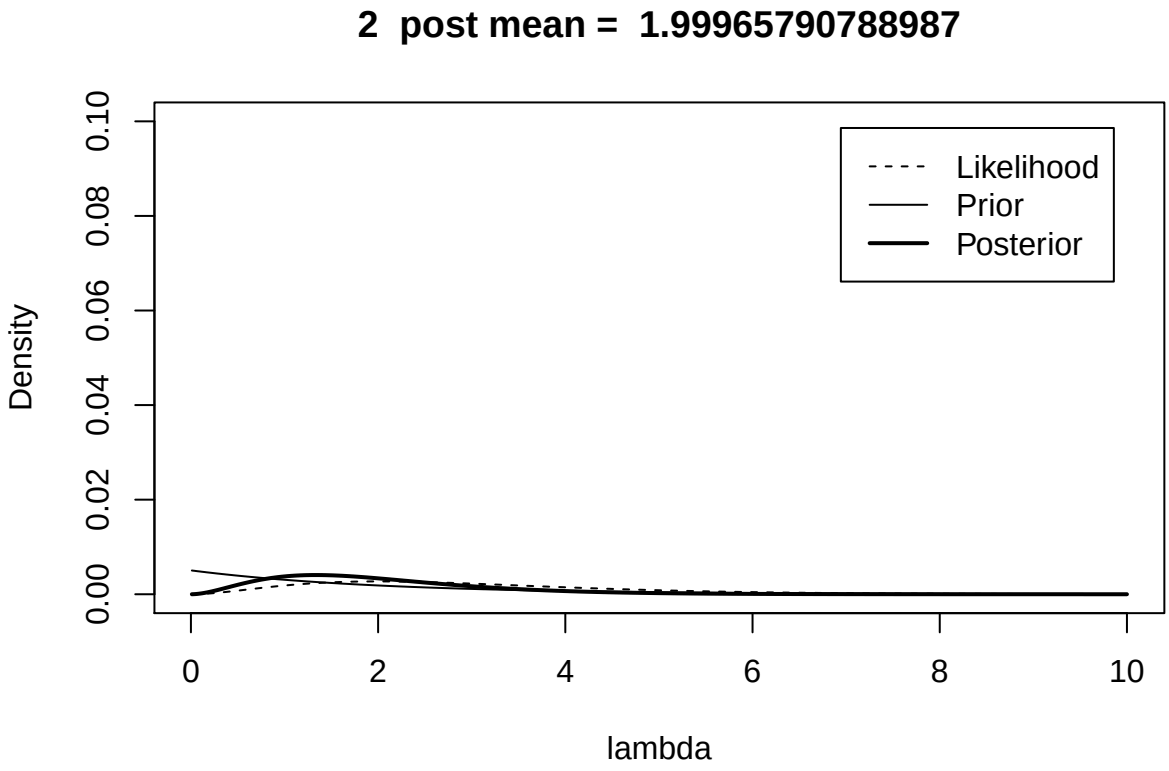

0 post mean = 0.671676107595651
Likelihood
Prior
Posterior
Density
0.00
0.02
0.04
0.06
0.08
0.10
0
2
4
6
8
10
lambda

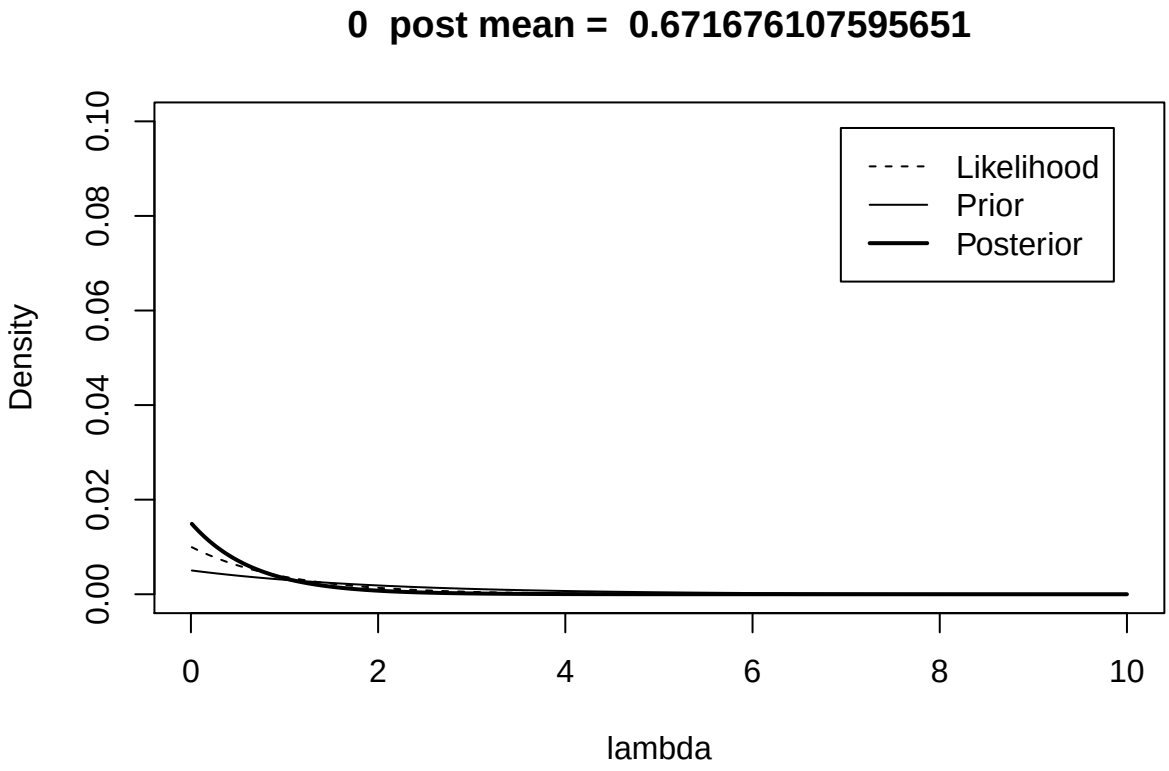

1 post mean = 1.33331274455286
Likelihood
Prior
Posterior
Density
0.00
0.02
0.04
0.06
0.08
0.10
0
2
4
6
8
10
lambda

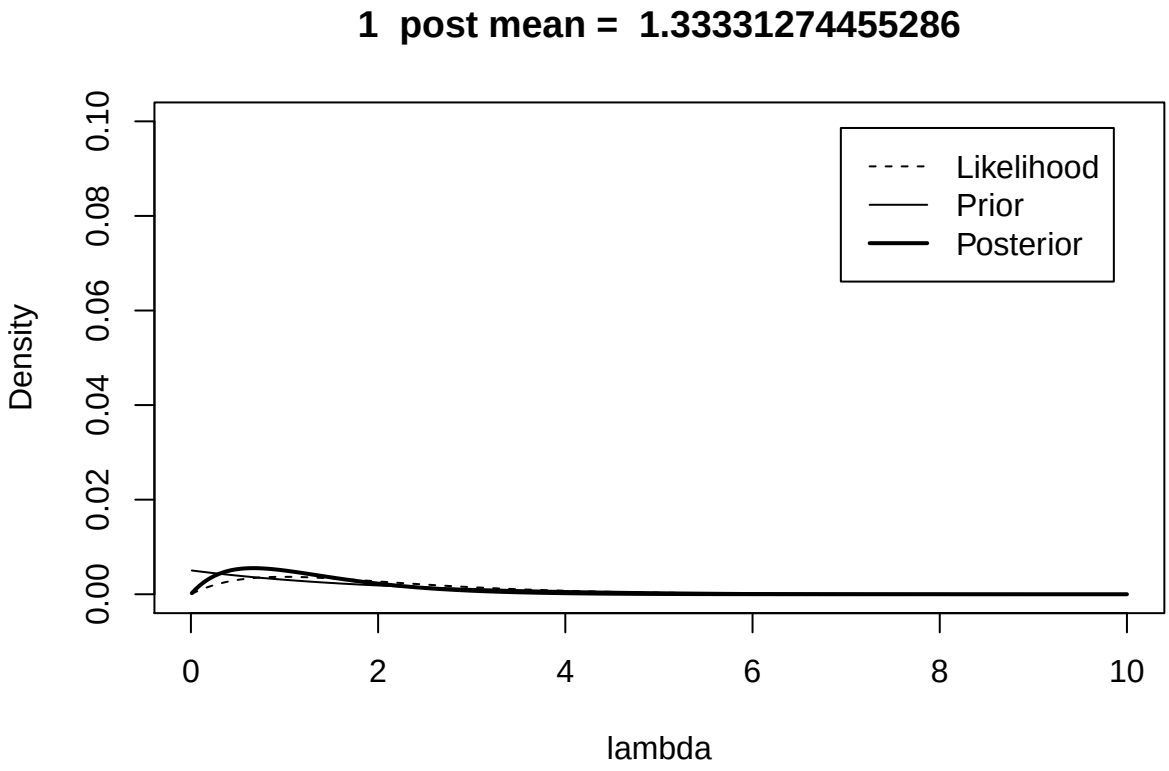

2 post mean = 1.99965790788987
Likelihood
Prior
Posterior
Density
0.00
0.02
0.04
0.06
0.08
0.10
0
2
4
6
8
10
lambda

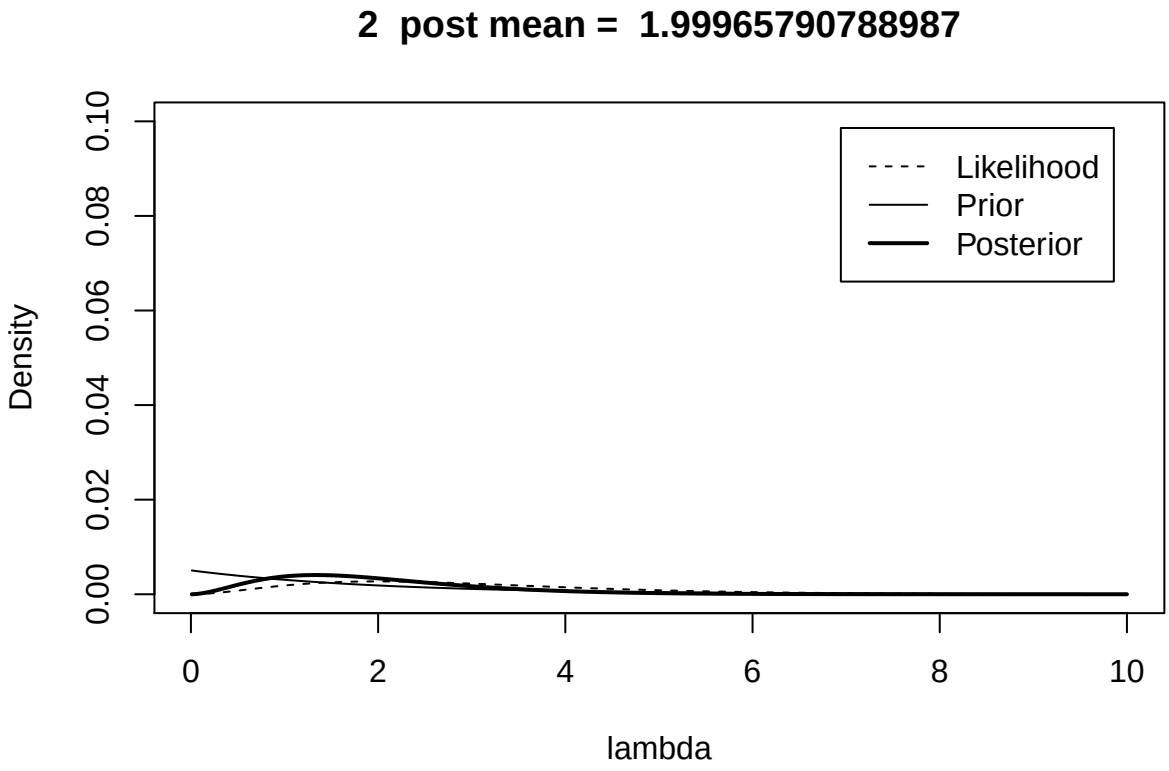

1 post mean = 1.33331274455286
Likelihood
Prior
Posterior
Density
0.00
0.02
0.04
0.06
0.08
0.10
0
2
4
6
8
10
lambda

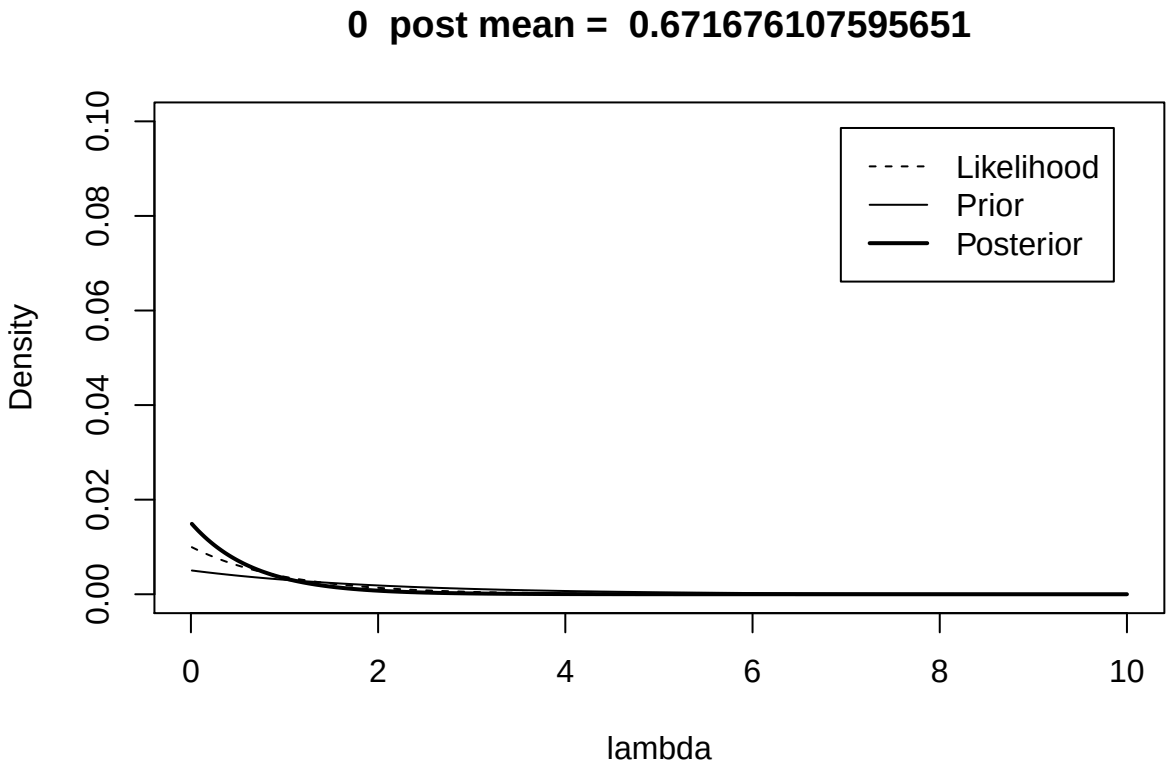

0 post mean = 0.671676107595651
Density
0.00
0.02
0.04
0.06
0.08
0.10
0
2
4
6
8
10
lambda
Likelihood
Prior
Posterior

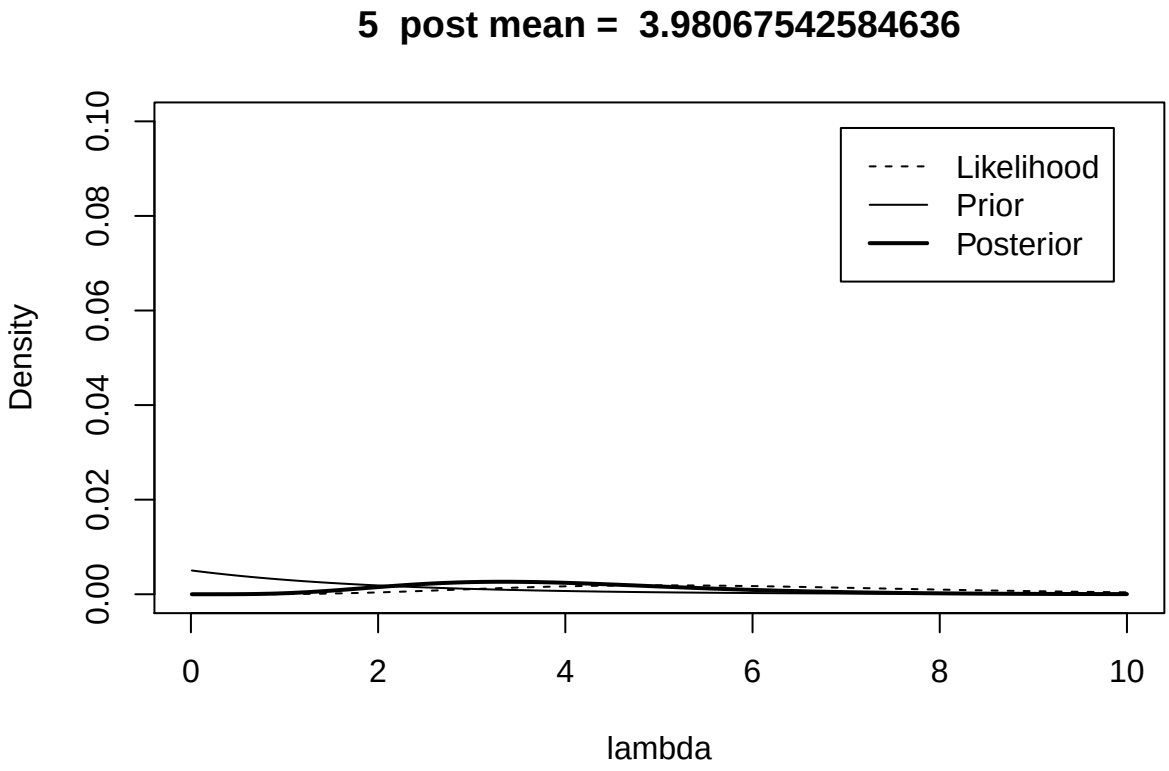

2 post mean = 1.99965790788987
Likelihood
Prior
Posterior
Density
0.00
0.02
0.04
0.06
0.08
0.10
0
2
4
6
8
10
lambda

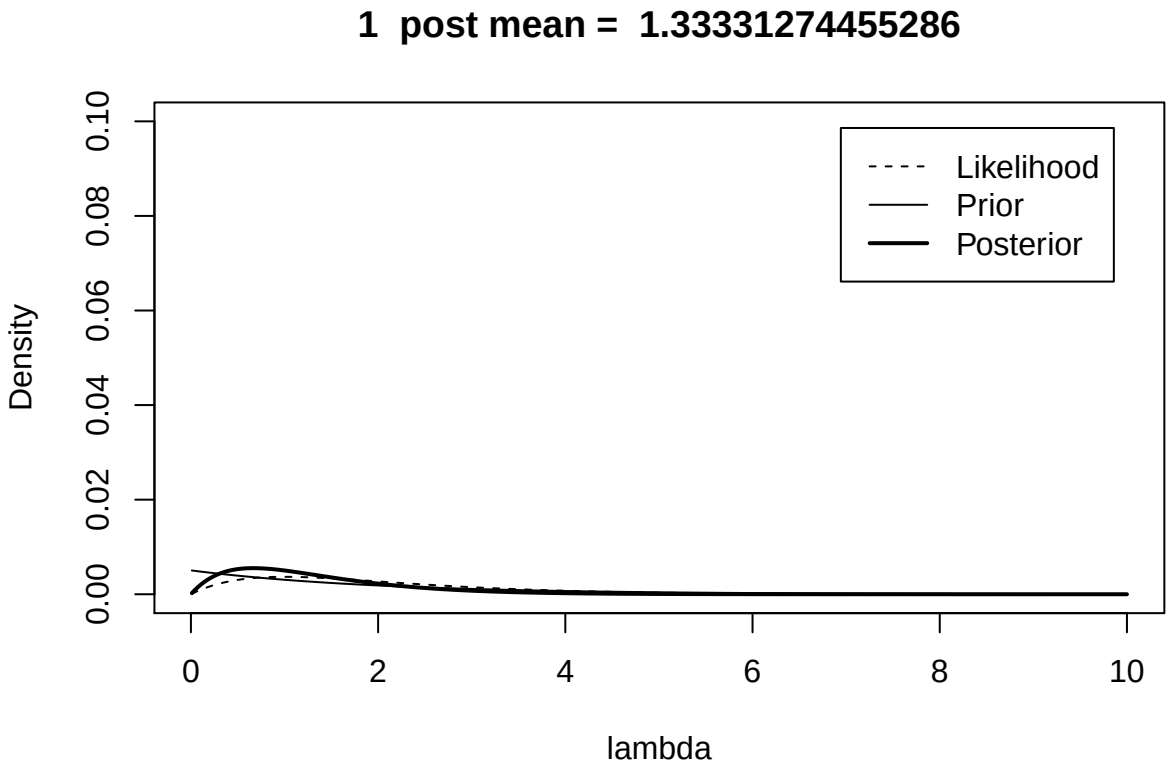

0 post mean = 0.671676107595651
Likelihood
Prior
Posterior
Density
0.00
0.02
0.04
0.06
0.08
0.10
0
2
4
6
8
10
lambda

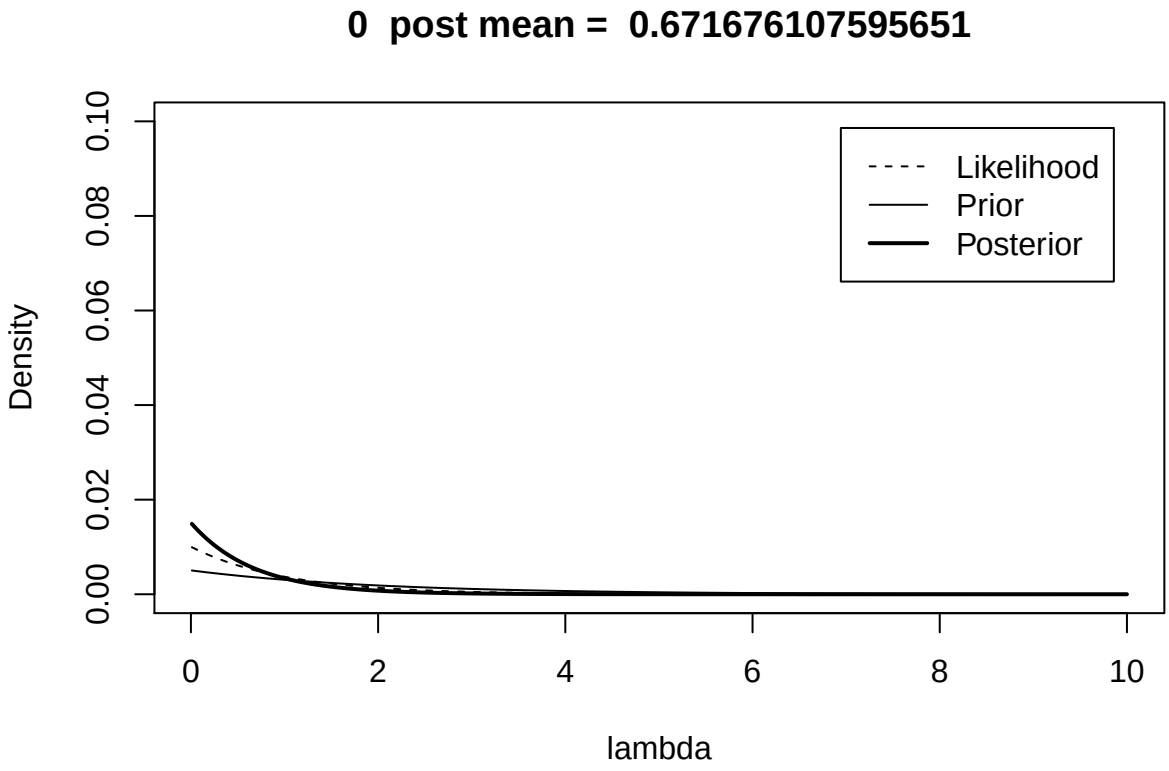

1 post mean = 1.33331274455286
Likelihood
Prior
Posterior
Density
0.00
0.02
0.04
0.06
0.08
0.10
0
2
4
6
8
10
lambda

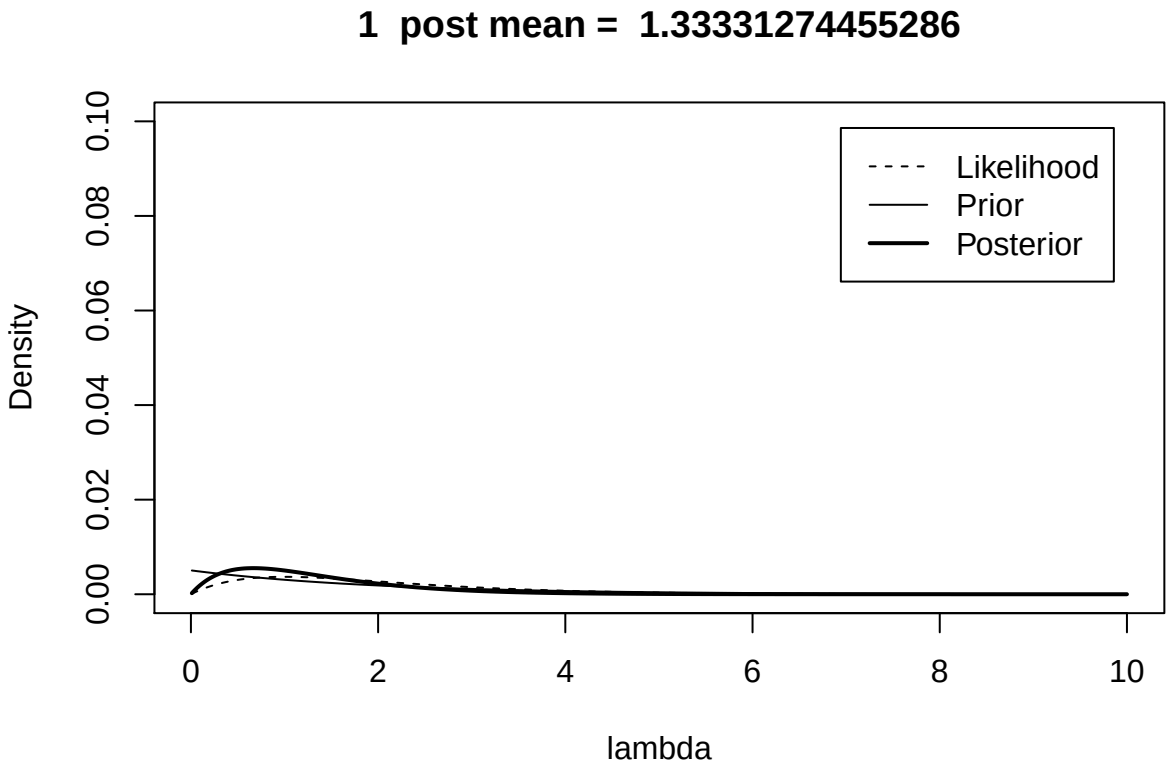

2 post mean = 1.99965790788987
Likelihood
Prior
Posterior
Density
0.00
0.02
0.04
0.06
0.08
0.10
0
2
4
6
8
10
lambda

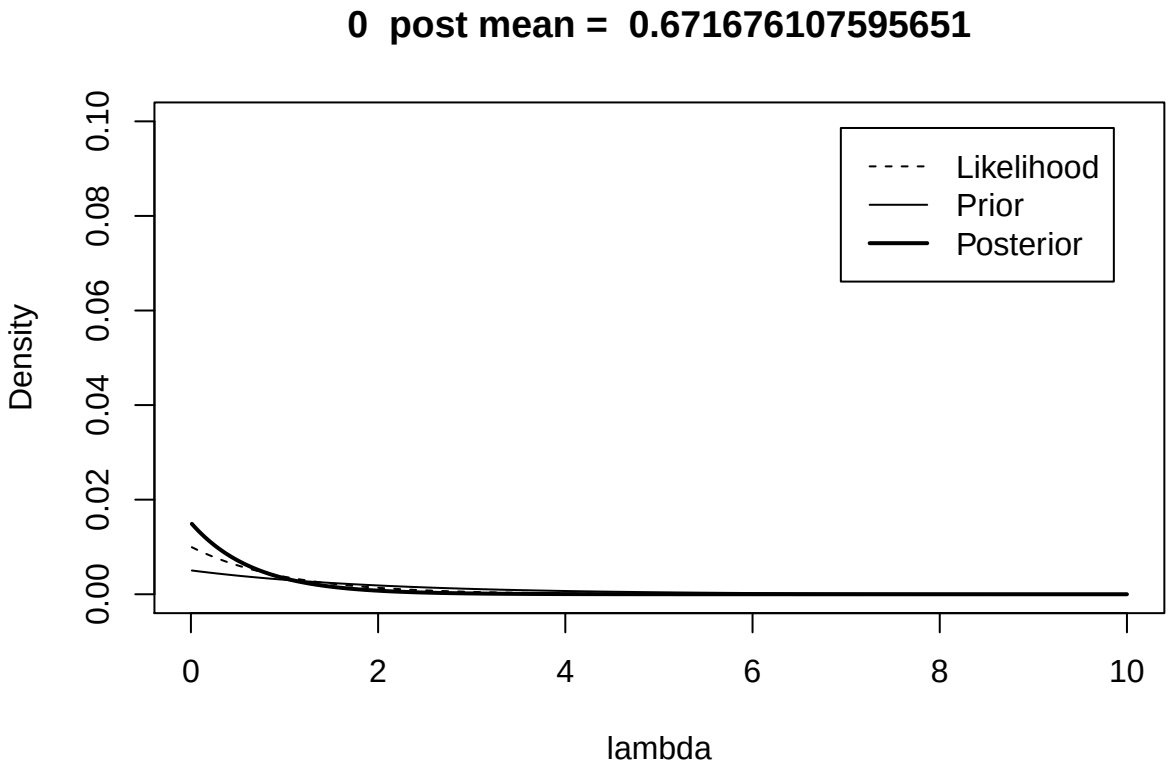

0 post mean = 0.671676107595651
Likelihood
Prior
Posterior
Density
0.00
0.02
0.04
0.06
0.08
0.10
0
2
4
6
8
10
lambda

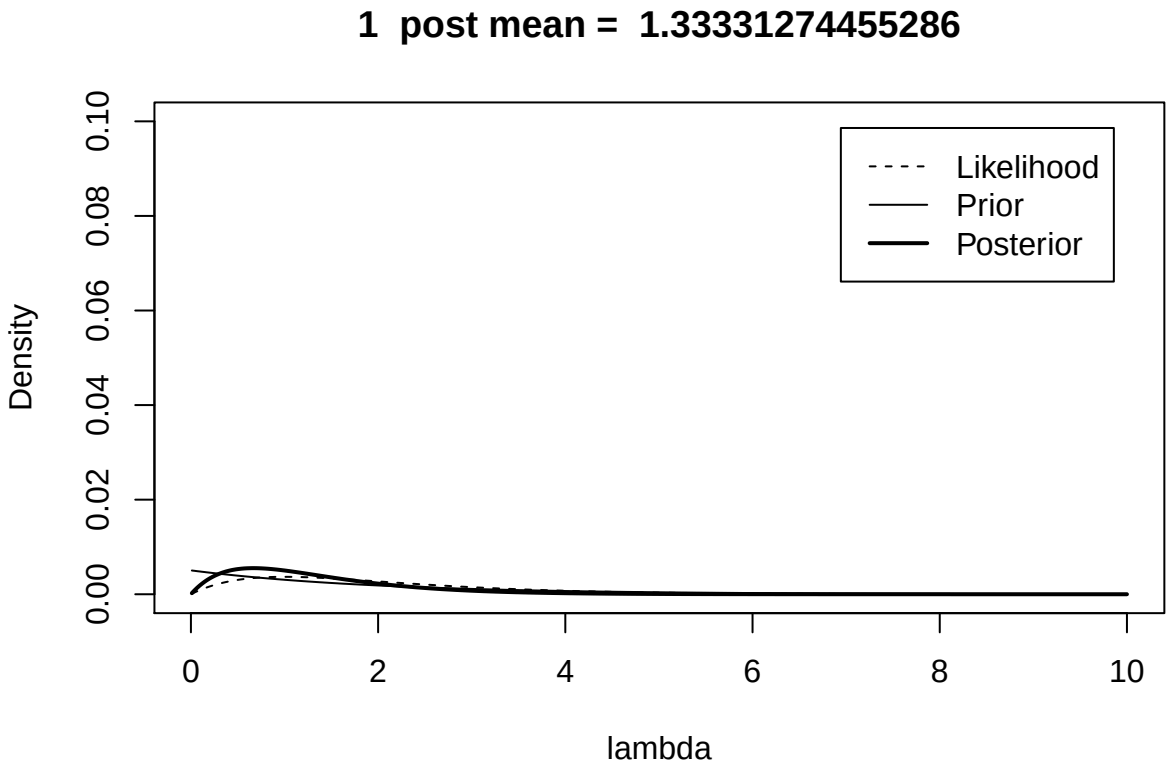

5 post mean = 3.98067542584636
Likelihood
Prior
Posterior
Density
0.00
0.02
0.04
0.06
0.08
0.10
0
2
4
6
8
10
lambda

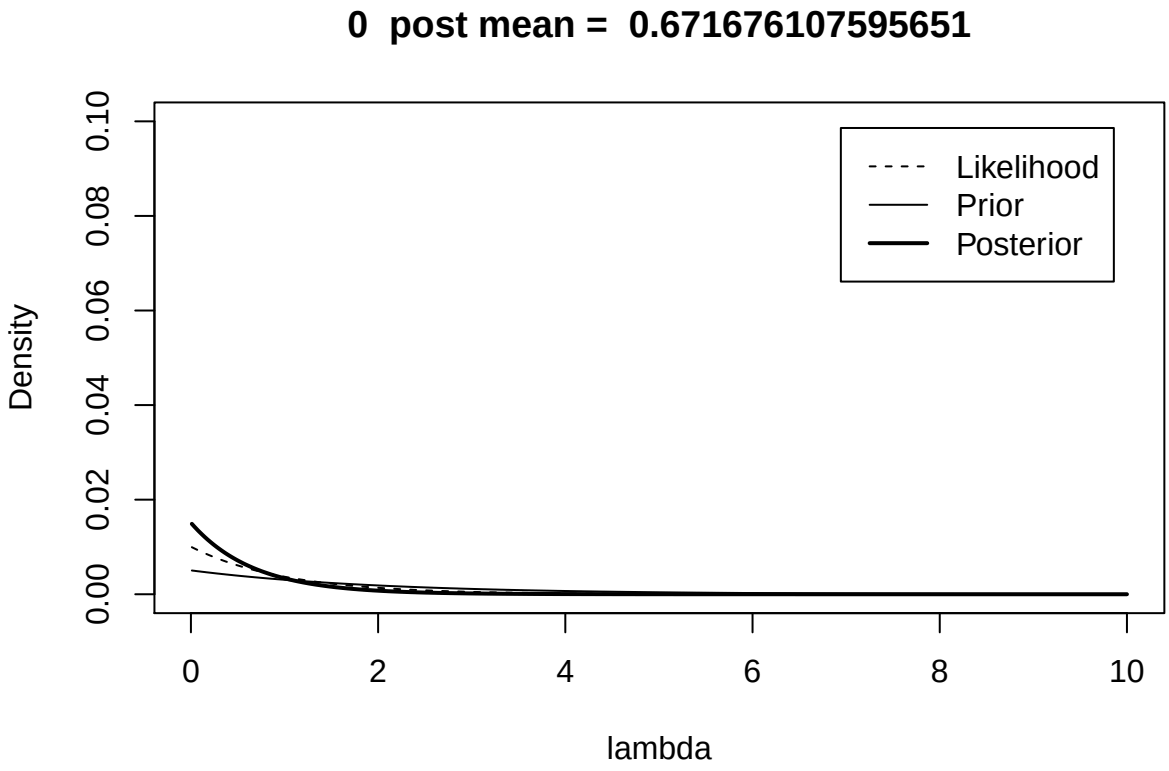

1 post mean = 1.33331274455286
Likelihood
Prior
Posterior
Density
0.00
0.02
0.04
0.06
0.08
0.10
0
2
4
6
8
10
lambda

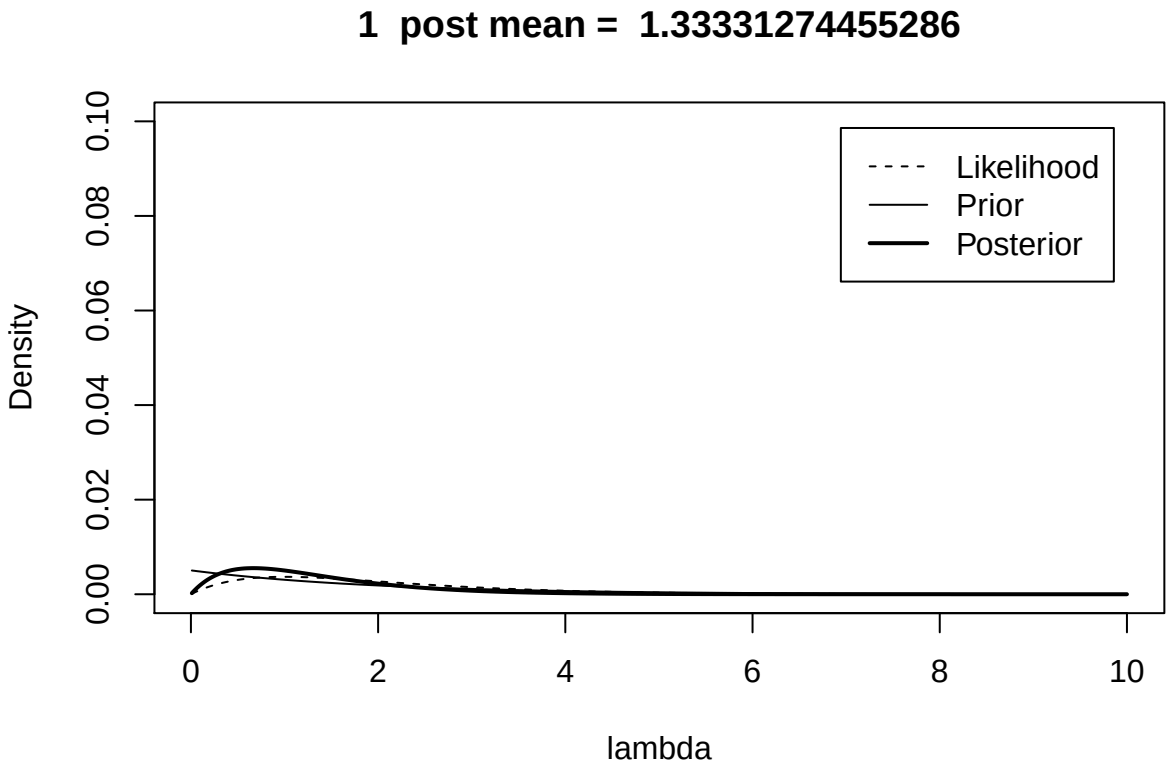

0 post mean = 0.671676107595651
Likelihood
Prior
Posterior
Density
0.00
0.02
0.04
0.06
0.08
0.10
0
2
4
6
8
10
lambda

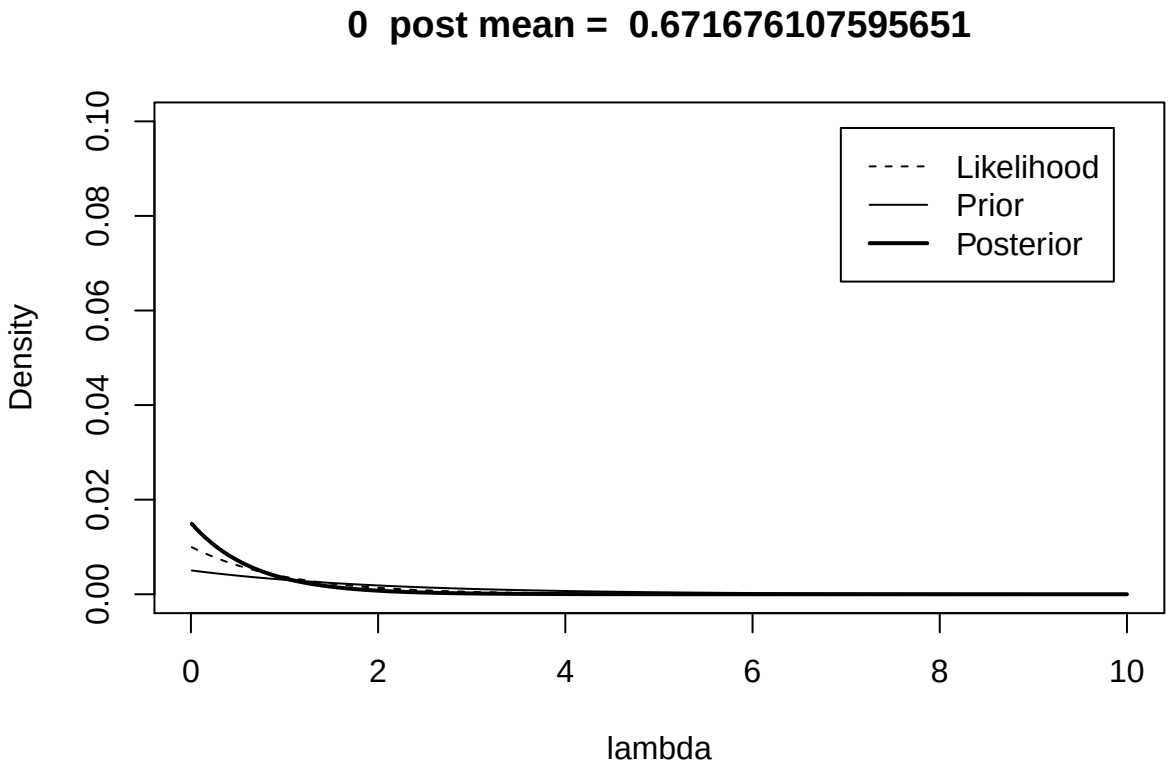

1 post mean = 1.33331274455286
Likelihood
Prior
Posterior
Density
0.00
0.02
0.04
0.06
0.08
0.10
0
2
4
6
8
10
lambda

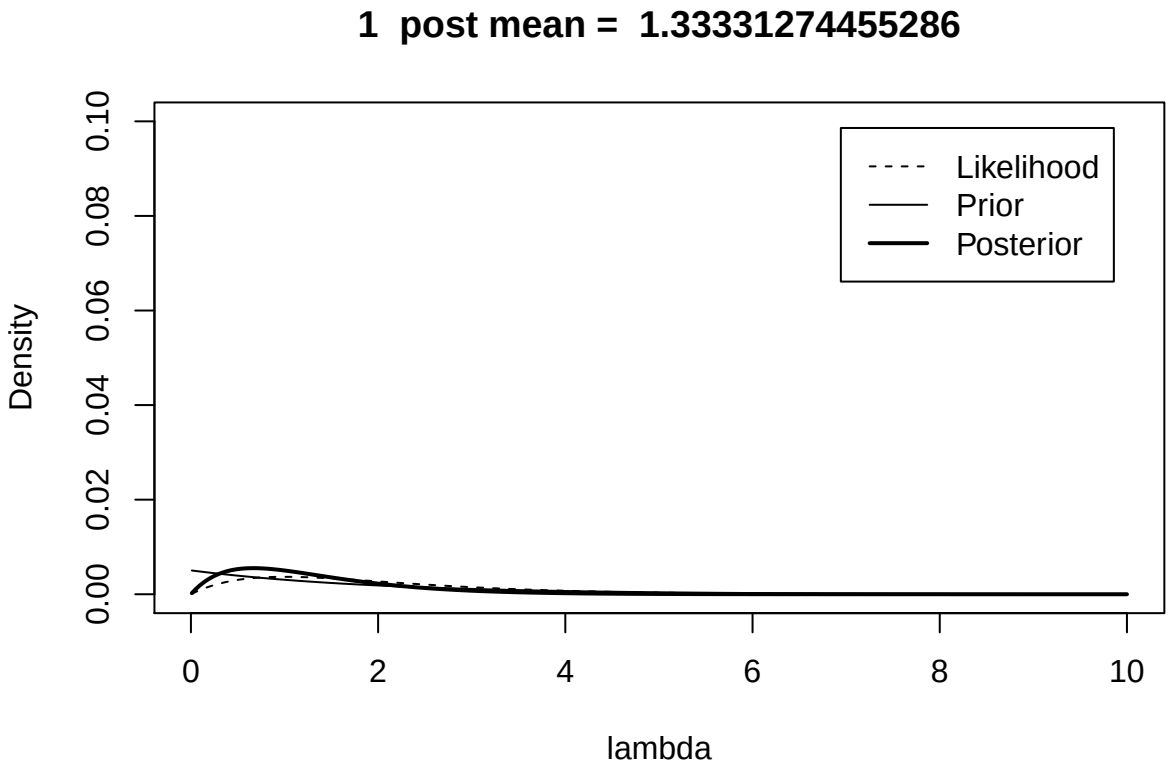

0 post mean = 0.671676107595651
Likelihood
Prior
Posterior
Density
0.00
0.02
0.04
0.06
0.08
0.10
0
2
4
6
8
10
lambda

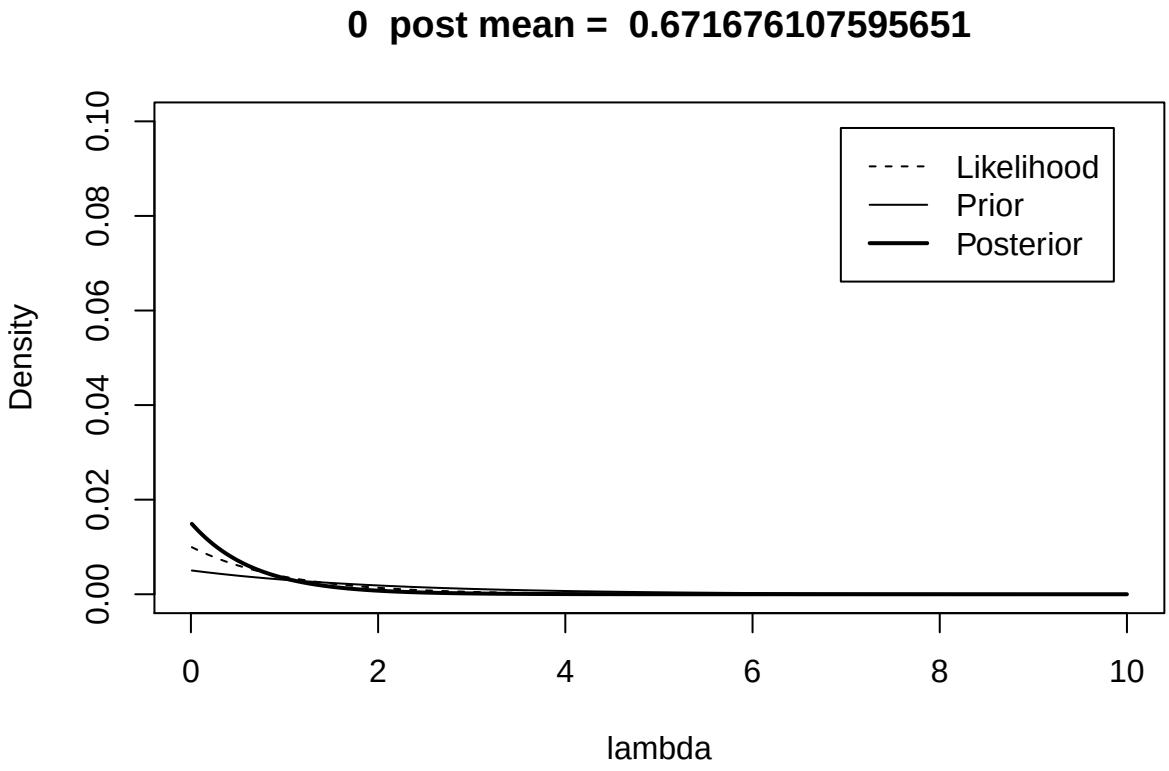

1 post mean = 1.33331274455286
Likelihood
Prior
Posterior
Density
0.00
0.02
0.04
0.06
0.08
0.10
0
2
4
6
8
10
lambda

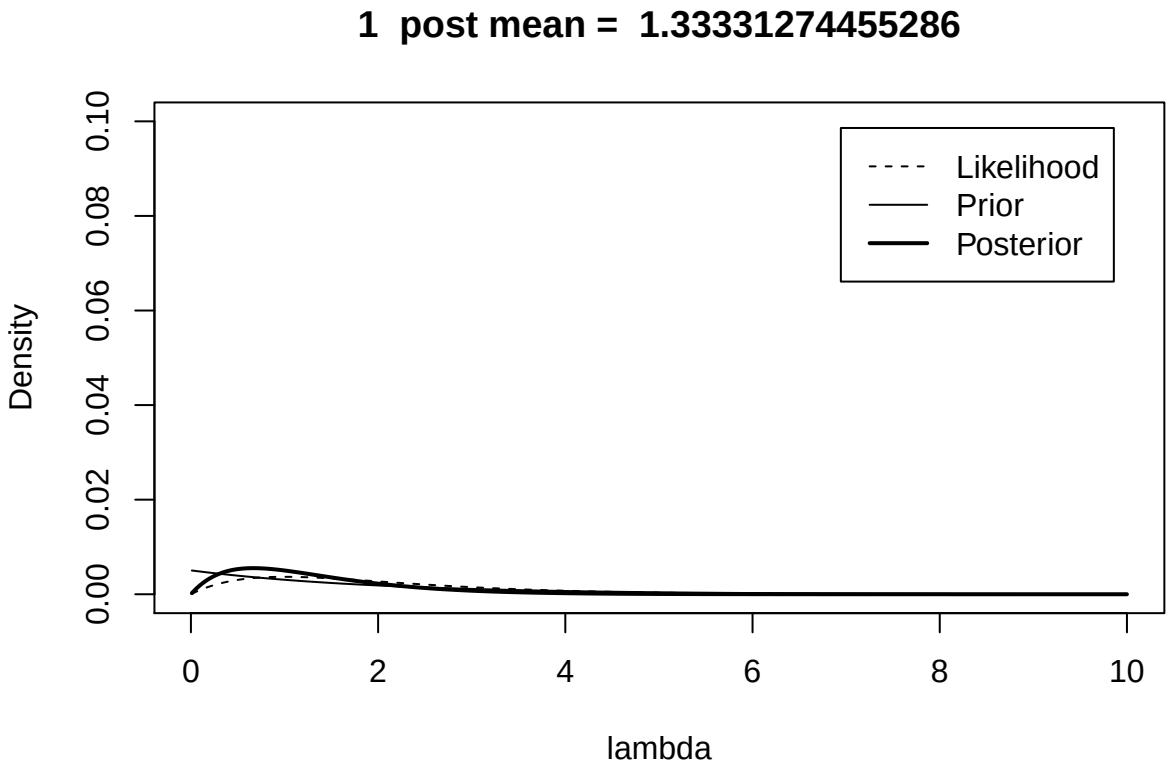

0 post mean = 0.671676107595651
Likelihood
Prior
Posterior
Density
0.00
0.02
0.04
0.06
0.08
0.10
0
2
4
6
8
10
lambda

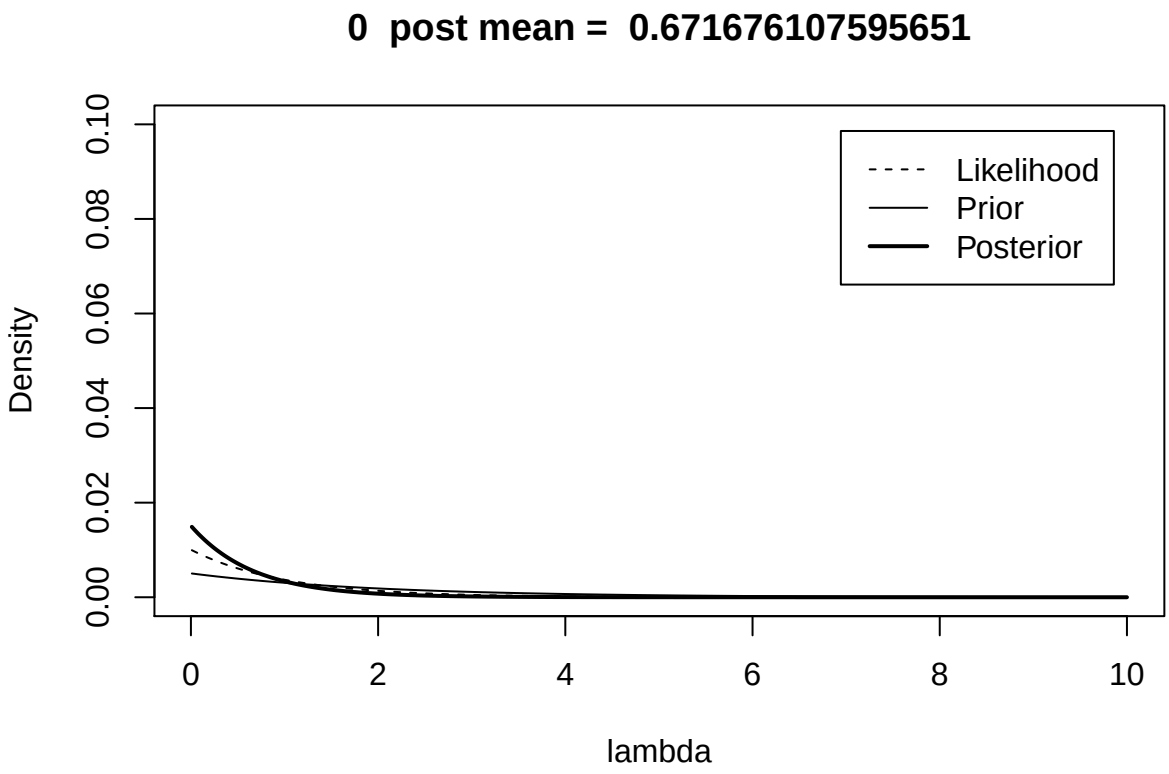

1 post mean = 1.33331274455286
Likelihood
Prior
Posterior
Density
0.00
0.02
0.04
0.06
0.08
0.10
0
2
4
6
8
10
lambda


0 post mean = 0.671676107595651
Likelihood
Prior
Posterior
Density
0.00
0.02
0.04
0.06
0.08
0.10
0
2
4
6
8
10
lambda


1 post mean = 1.33331274455286
Likelihood
Prior
Posterior
Density
0.00
0.02
0.04
0.06
0.08
0.10
0
2
4
6
8
10
lambda


0 post mean = 0.671676107595651
Likelihood
Prior
Posterior
Density
0.00
0.02
0.04
0.06
0.08
0.10
0
2
4
6
8
10
lambda


1 post mean = 1.33331274455286
Density
0.00
0.02
0.04
0.06
0.08
0.10
0
2
4
6
8
10
lambda
Likelihood
Prior
Posterior

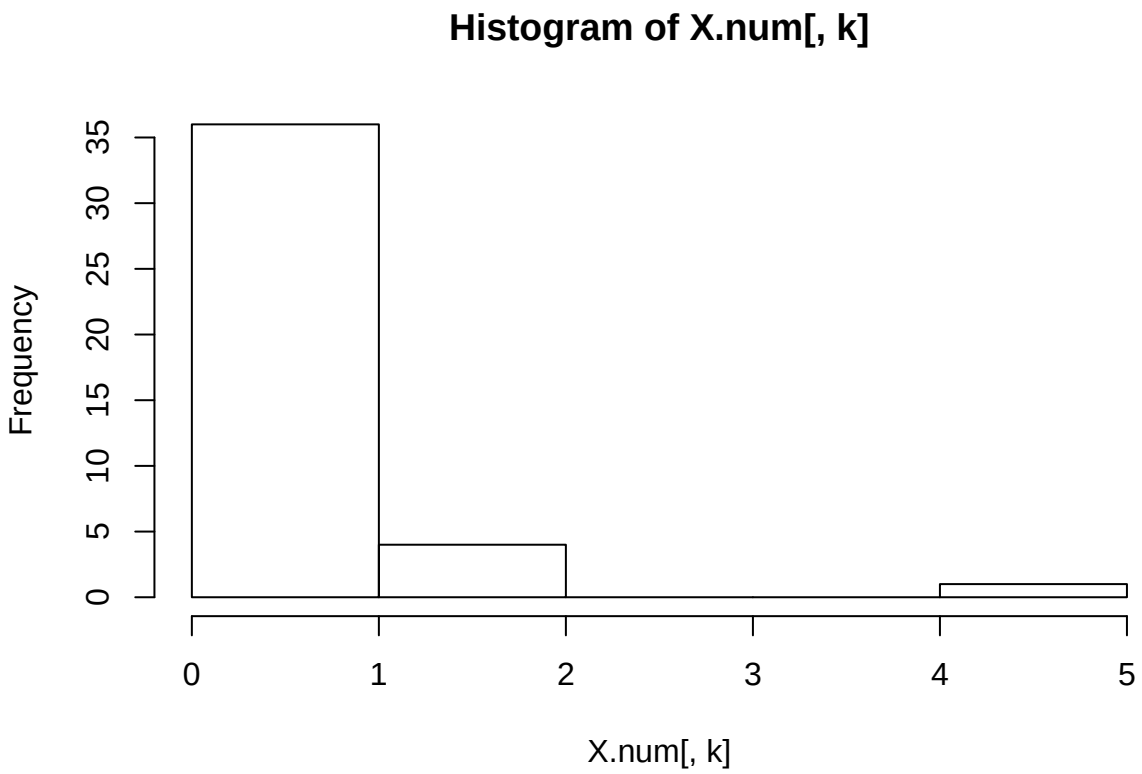


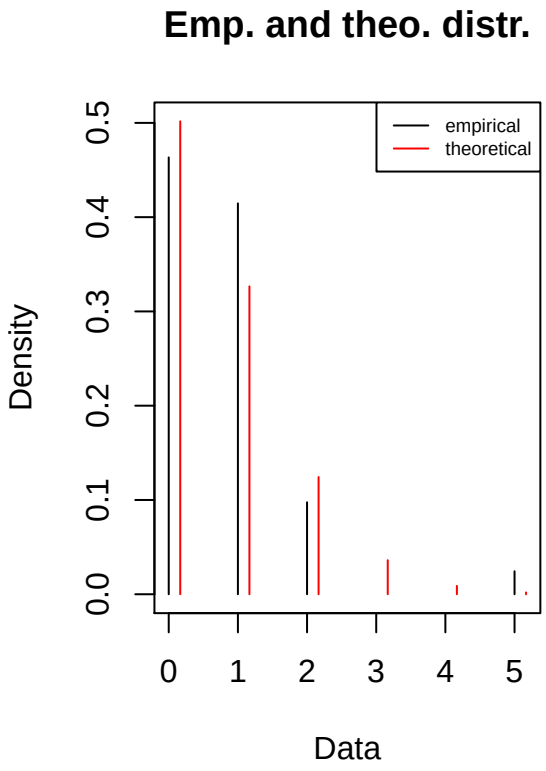
Find MLE of fit to pois and negbinom, and fit glm's with time as predictor

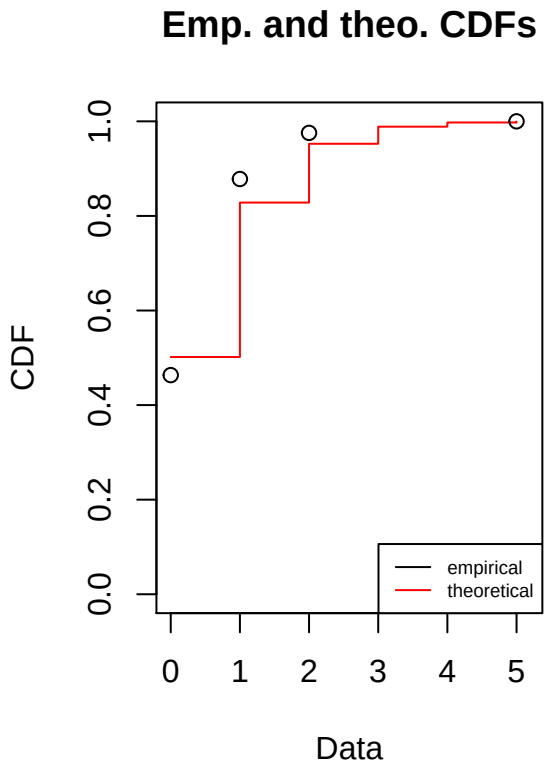
```
################################### Find MLE of fit to pois and negbinom, and fit glm's with time as pred
mle.nb.params <- matrix(nrow = 9,ncol = 2)
mle.pois.params <- matrix(nrow = 9,ncol = 1)

mle.nb.glm.params <- matrix(nrow = 9,ncol = 8)
mle.pois.glm.params <- matrix(nrow = 9,ncol = 8)
for(k in 1:9)
{
  hist(X.num[,k])
  library(fitdistrplus)
  library(gamlss)
  fit_nb <- fitdist(X.num[,k], 'nbinom', start = list(mu = 3, size = 0.1))
  mle.nb.params[k,1] <- fit_nb$estimate[1]
  mle.nb.params[k,2] <- fit_nb$estimate[2]
  plot(fit_nb)
  #gofstat(fit_nb)

  fit_pois <- fitdist(X.num[,k], 'pois',method = 'mle')
  mle.pois.params[k,1] <- fit_pois$estimate[1]
  plot(fit_pois)
```

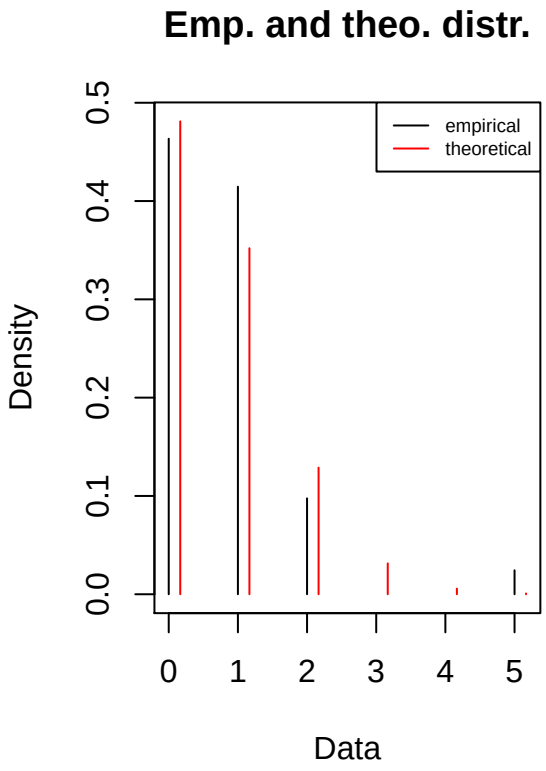
```
  #gofstat(fit_pois)

  df <- data.frame(t=seq(1:41),Y=X.num[,k])
  model.pois <- glm( Y~ t, family=poisson, df)
  sp <- summary.glm(model.pois)

  model.nb <- glm.nb(Y~t,data=df)
  snb <- summary.glm(model.nb)

  mle.nb.glm.params[k,1:4] <- snb$coefficients[1,]
  mle.nb.glm.params[k,5:8] <- snb$coefficients[2,]

  mle.pois.glm.params[k,1:4] <- sp$coefficients[1,]
  mle.pois.glm.params[k,5:8] <- sp$coefficients[2,]
}
```

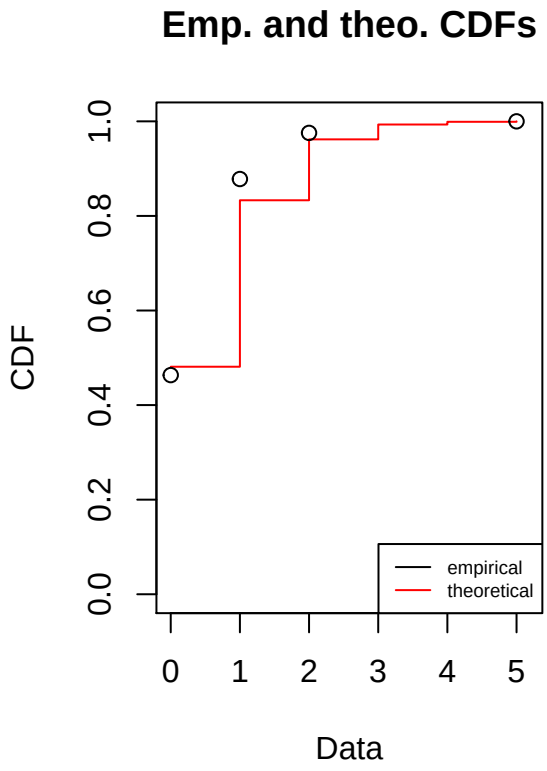
```
## Loading required package: MASS

## Loading required package: survival

## Loading required package: splines

## Loading required package: gamlss.data

## Loading required package: gamlss.dist

## Loading required package: nlme

## Loading required package: parallel

##  **********   GAMLSS Version 5.0-6  **********

## For more on GAMLSS look at http://www.gamlss.org/

## Type gamlssNews() to see new features/changes/bug fixes.
```

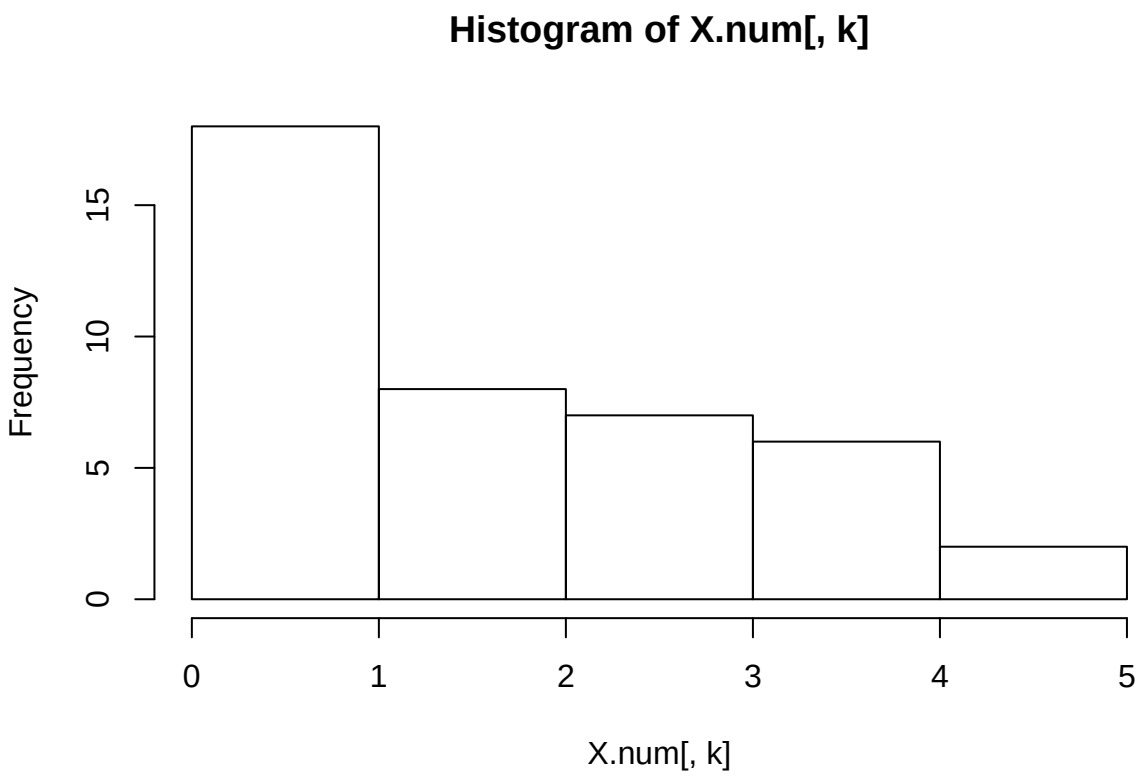

Histogram of X.num[, k]
Frequency
0
5
10
15
20
25
30
35
0
1
2
3
4
5
X.num[, k]

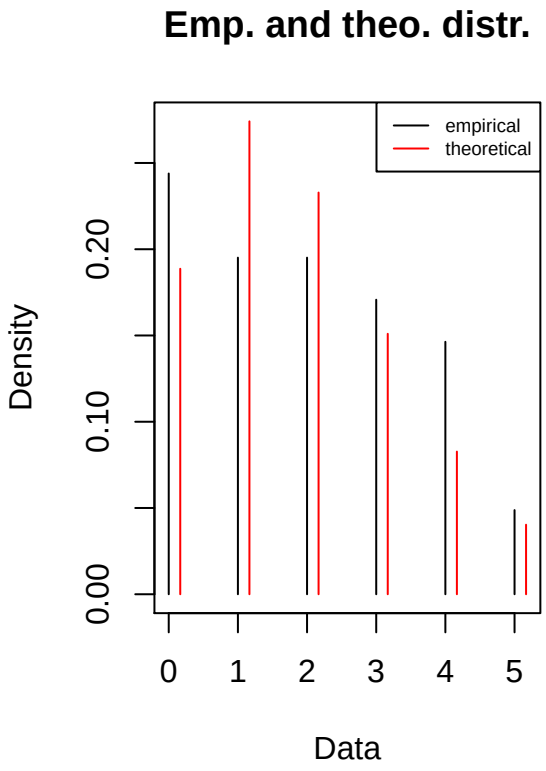

Emp. and theo. distr.
Density
0.0
0.1
0.2
0.3
0.4
0.5
0
1
2
3
4
5
Data
empirical
theoretical

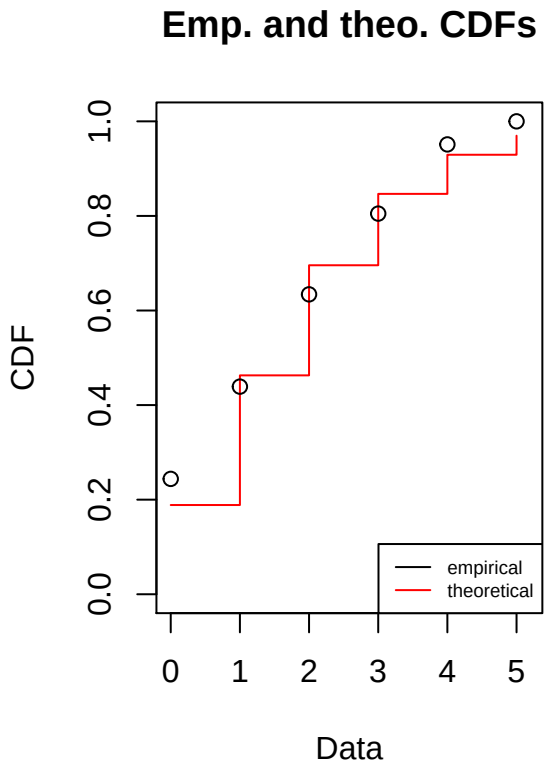

Emp. and theo. CDFs
CDF
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5
Data
empirical
theoretical

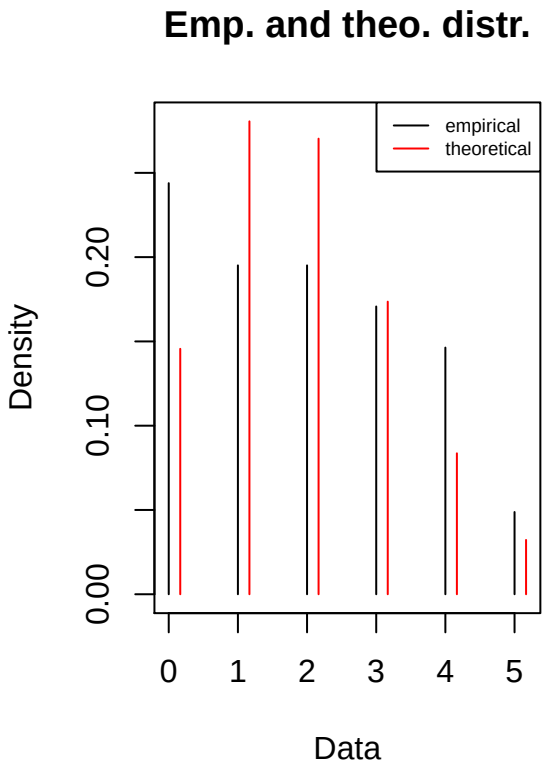

Emp. and theo. distr.
Density
0.0
0.1
0.2
0.3
0.4
0.5
empirical
theoretical
0
1
2
3
4
5
Data

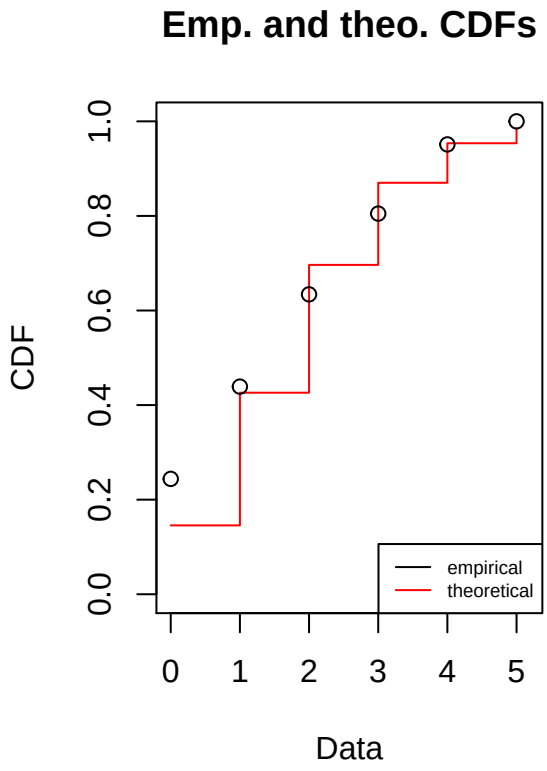

Emp. and theo. CDFs
CDF
0.0
0.2
0.4
0.6
0.8
1.0
empirical
theoretical
0
1
2
3
4
5
Data

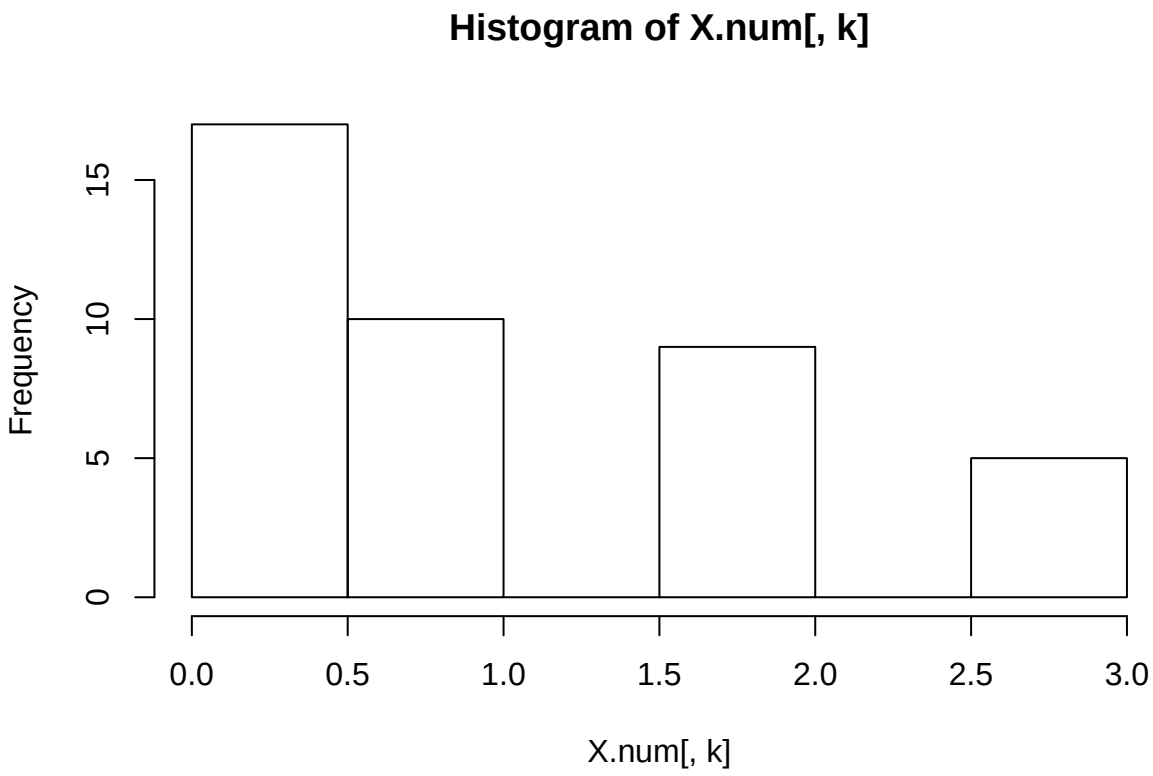

Histogram of X.num[, k]
Frequency
0
5
10
15
0
1
2
3
4
5
X.num[, k]

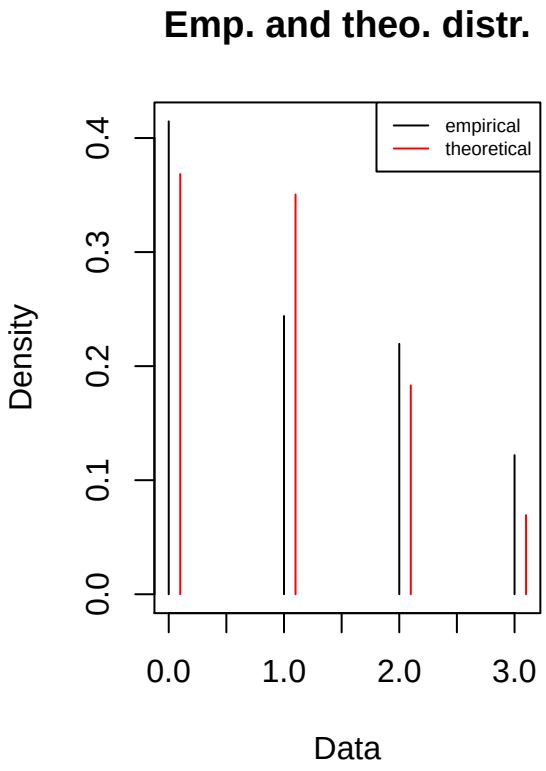

Emp. and theo. distr.
empirical
theoretical
Density
0.00
0.10
0.20
0
1
2
3
4
5
Data

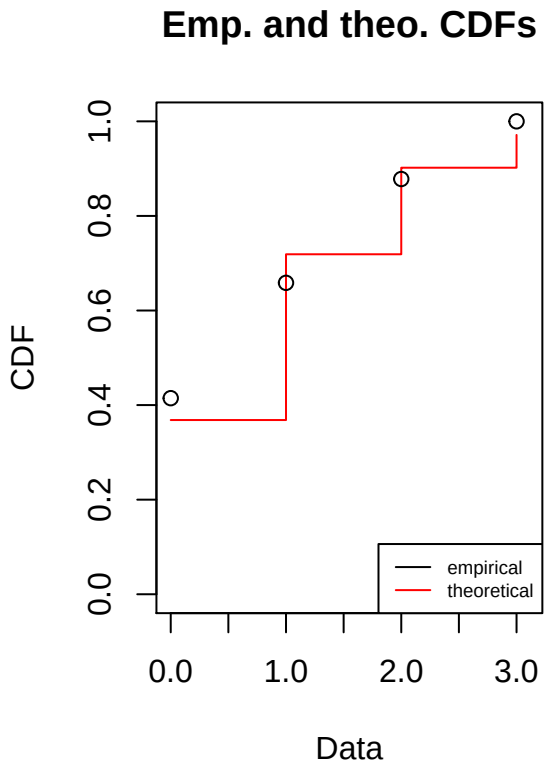

Emp. and theo. CDFs
CDF
0.0
0.2
0.4
0.6
0.8
1.0
empirical
theoretical
0
1
2
3
4
5
Data

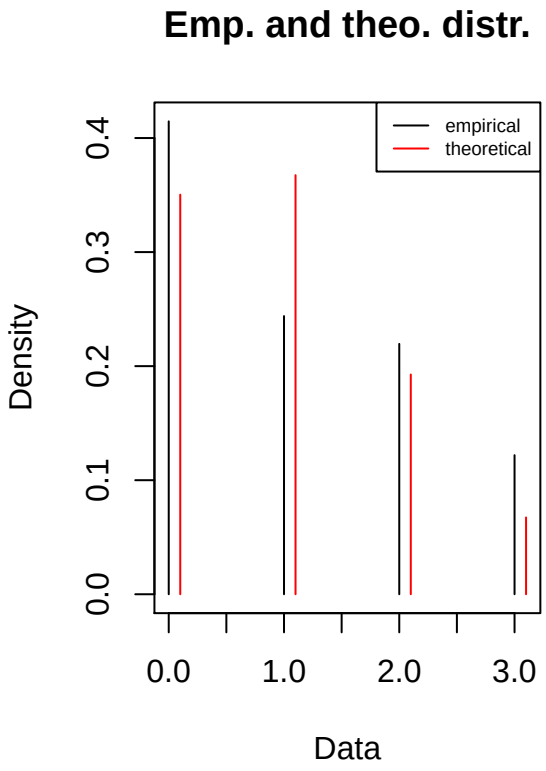

Emp. and theo. distr.
empirical
theoretical
Density
0.00
0.10
0.20
0
1
2
3
4
5
Data

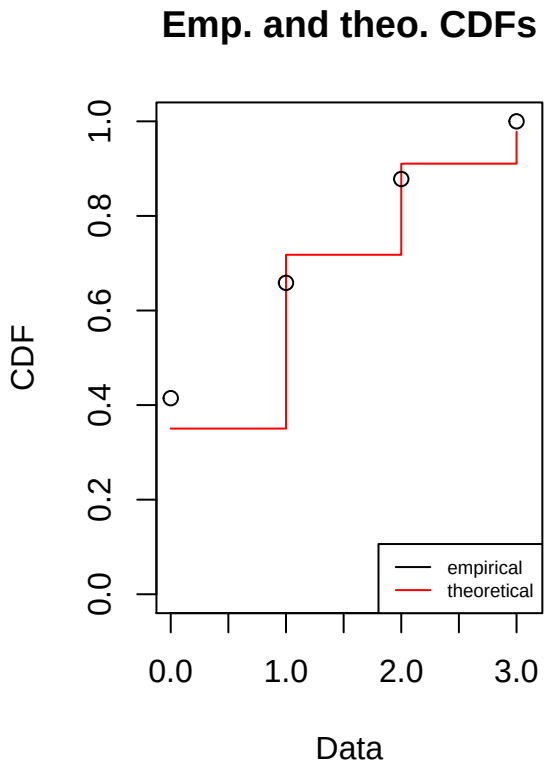

Emp. and theo. CDFs
CDF
0.0
0.2
0.4
0.6
0.8
1.0
empirical
theoretical
0
1
2
3
4
5
Data

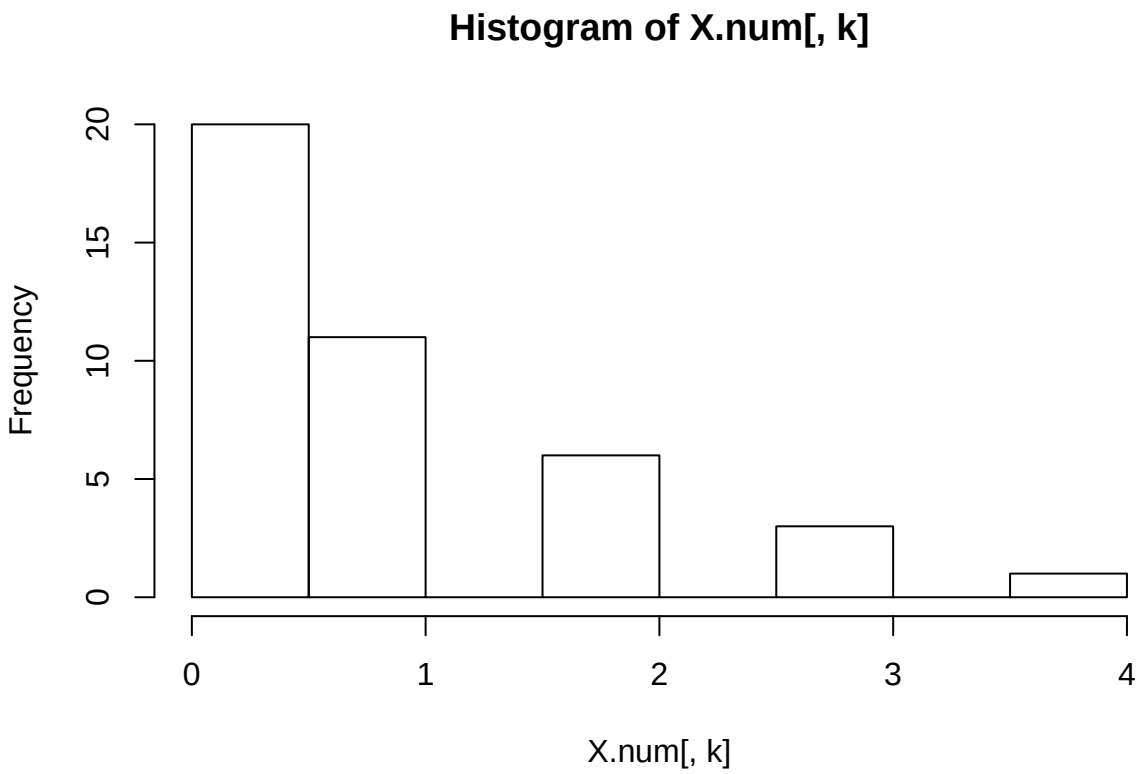

Histogram of X.num[, k]
Frequency
0
5
10
15
0.0
0.5
1.0
1.5
2.0
2.5
3.0
X.num[, k]

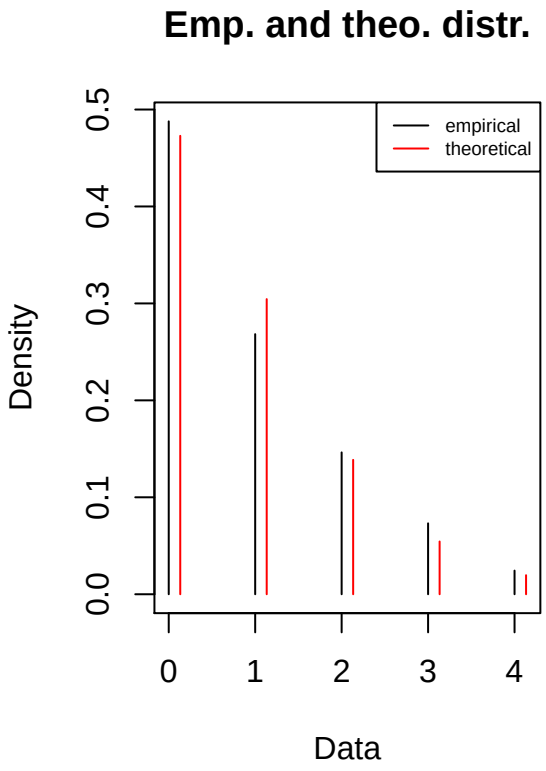

Emp. and theo. distr.
Density
0.0
0.1
0.2
0.3
0.4
empirical
theoretical
0.0
1.0
2.0
3.0
Data

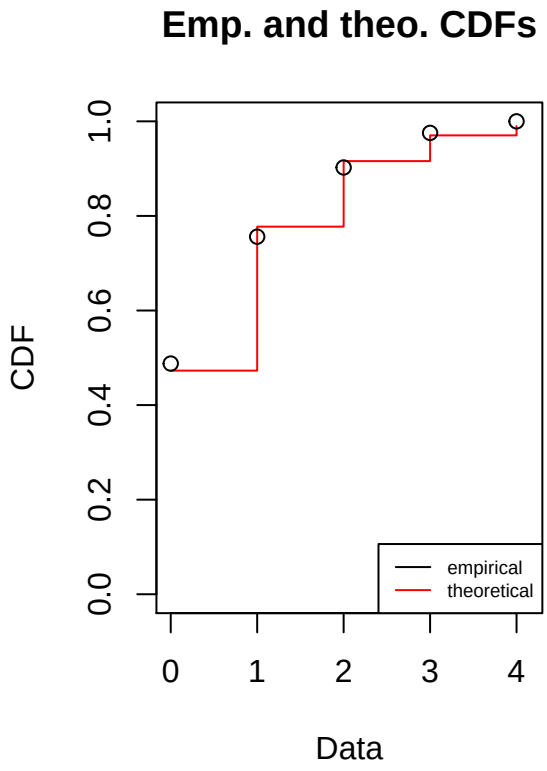

Emp. and theo. CDFs
CDF
0.0
0.2
0.4
0.6
0.8
1.0
empirical
theoretical
0.0
1.0
2.0
3.0
Data

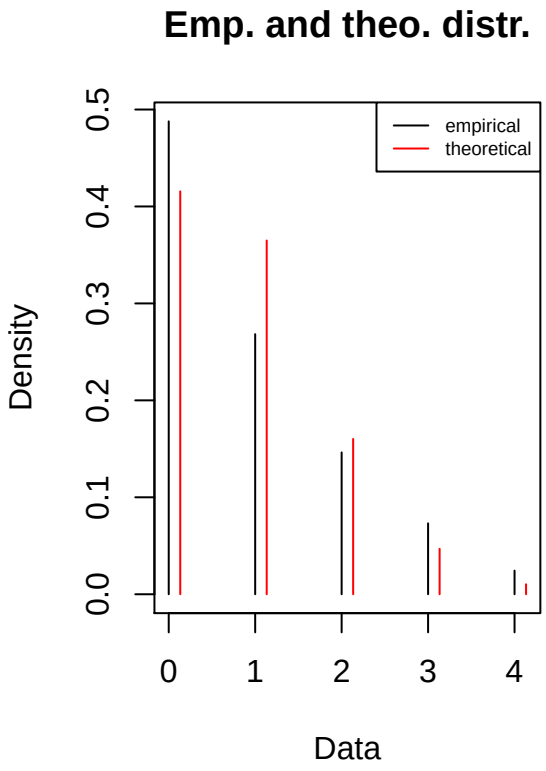

Emp. and theo. distr.
Density
0.0
0.1
0.2
0.3
0.4
empirical
theoretical
0.0
1.0
2.0
3.0
Data

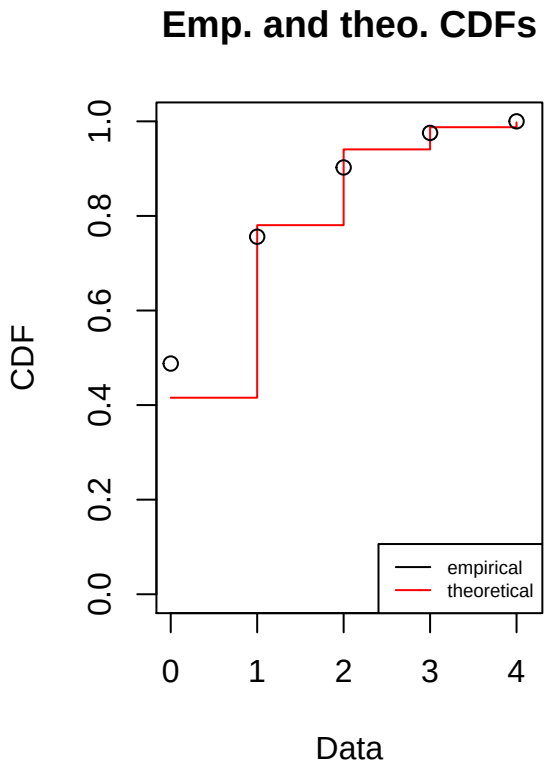

Emp. and theo. CDFs
CDF
0.0
0.2
0.4
0.6
0.8
1.0
empirical
theoretical
0.0
1.0
2.0
3.0
Data

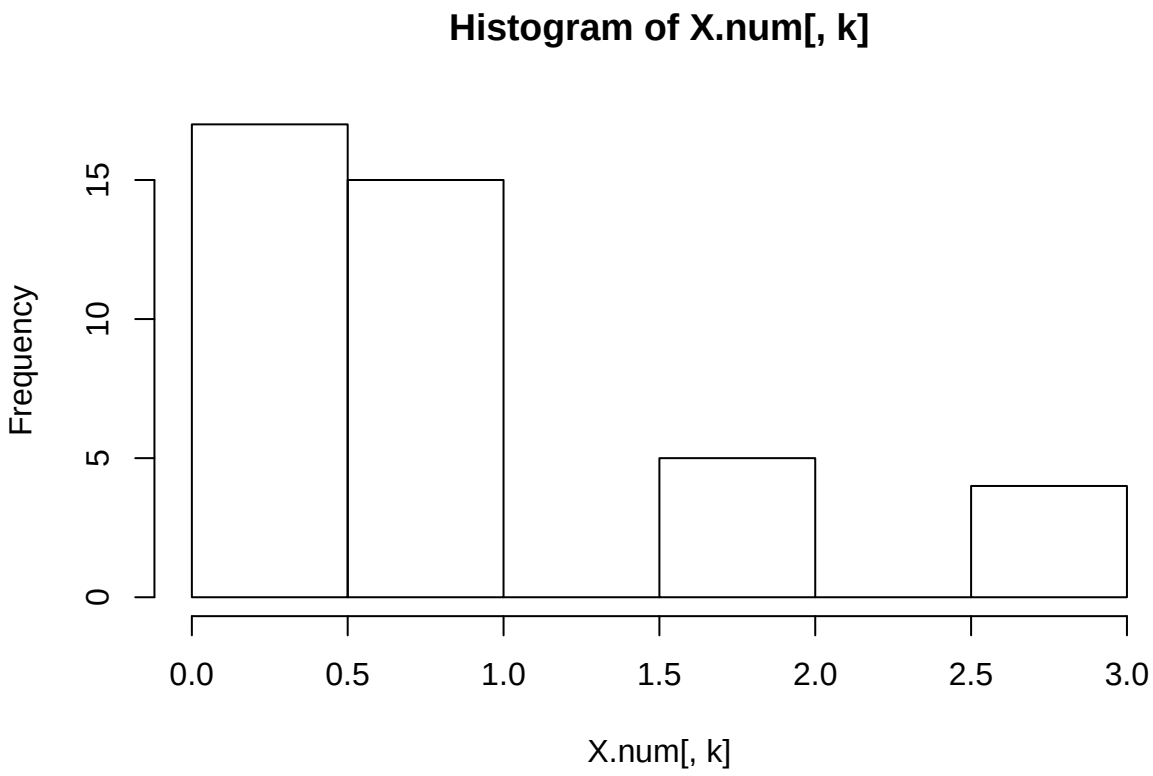

Histogram of X.num[, k]
Frequency
0
5
10
15
20
0
1
2
3
4
X.num[, k]

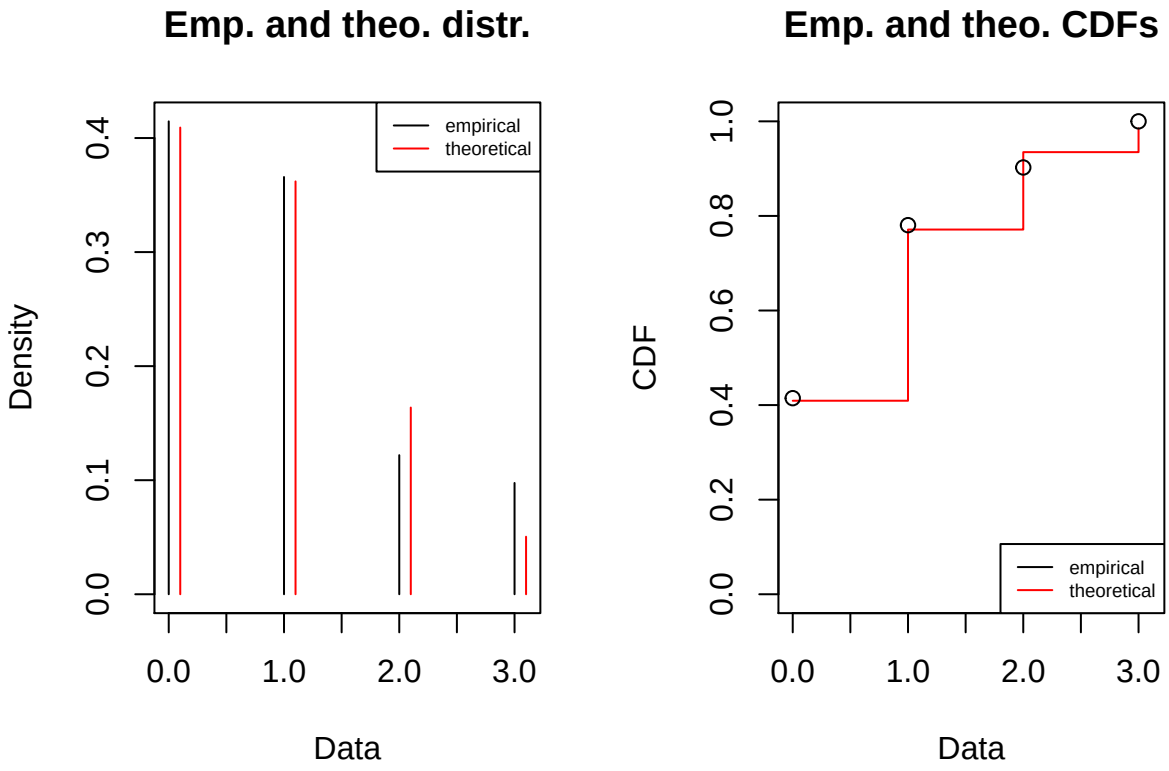

Emp. and theo. distr.
Density
0.0
0.1
0.2
0.3
0.4
0.5
0
1
2
3
4
Data
empirical
theoretical


Emp. and theo. CDFs
CDF
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
Data
empirical
theoretical

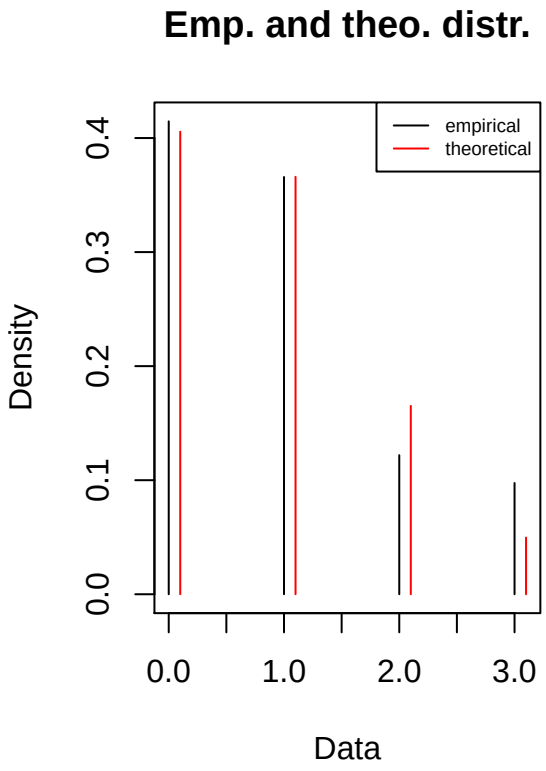

Emp. and theo. distr.
Density
0.0
0.1
0.2
0.3
0.4
0.5
0
1
2
3
4
Data
empirical
theoretical

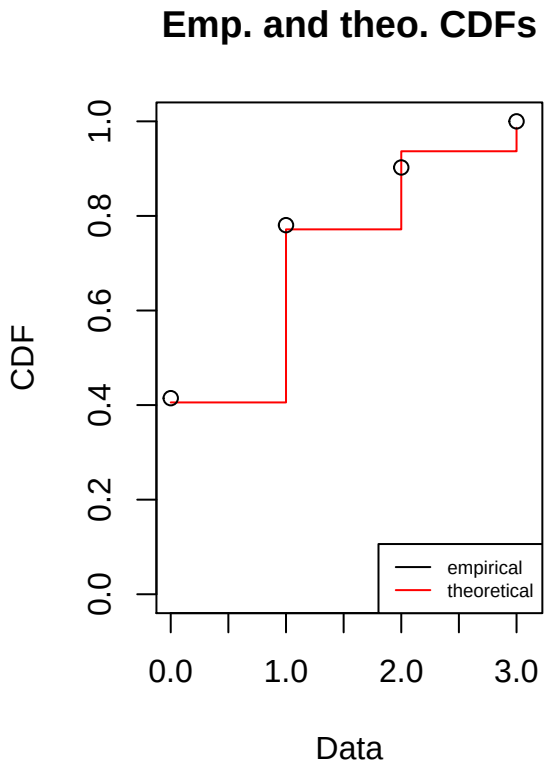

Emp. and theo. CDFs
CDF
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
Data
empirical
theoretical

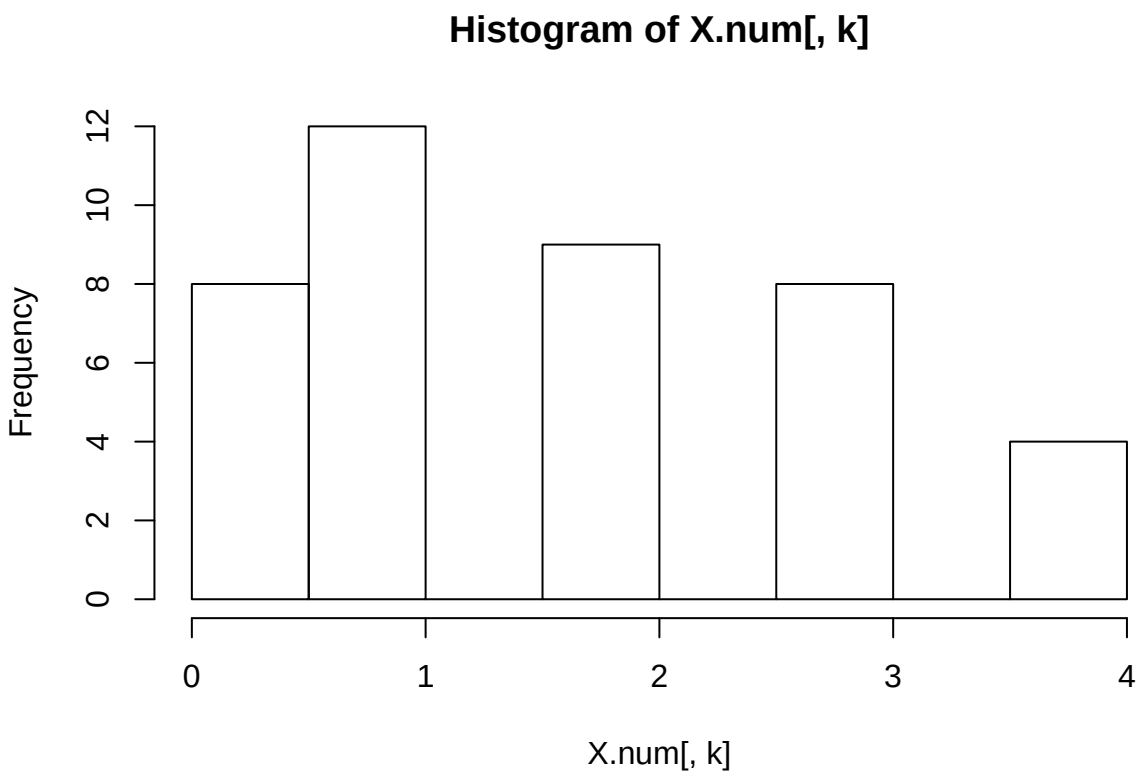

Histogram of X.num[, k]
Frequency
0
5
10
15
0.0
0.5
1.0
1.5
2.0
2.5
3.0
X.num[, k]

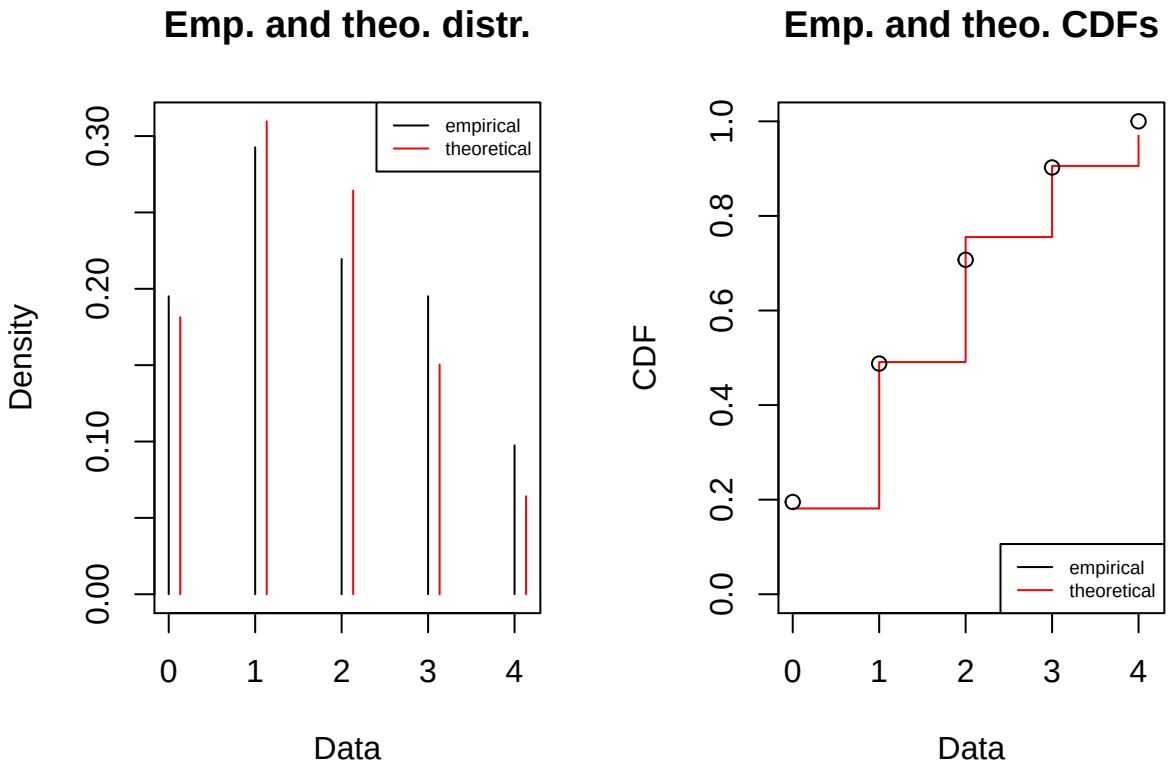



```
## Warning in theta.ml(Y, mu, sum(w), w, limit = control$maxit, trace =
## control$trace > : iteration limit reached

## Warning in theta.ml(Y, mu, sum(w), w, limit = control$maxit, trace =
## control$trace > : iteration limit reached
```

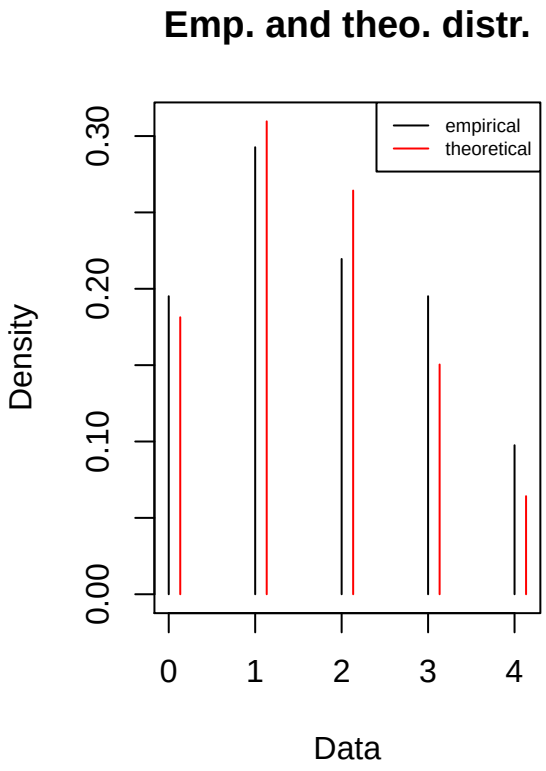

Emp. and theo. distr.
Density
0.0
0.1
0.2
0.3
0.4
empirical
theoretical
0.0
1.0
2.0
3.0
Data

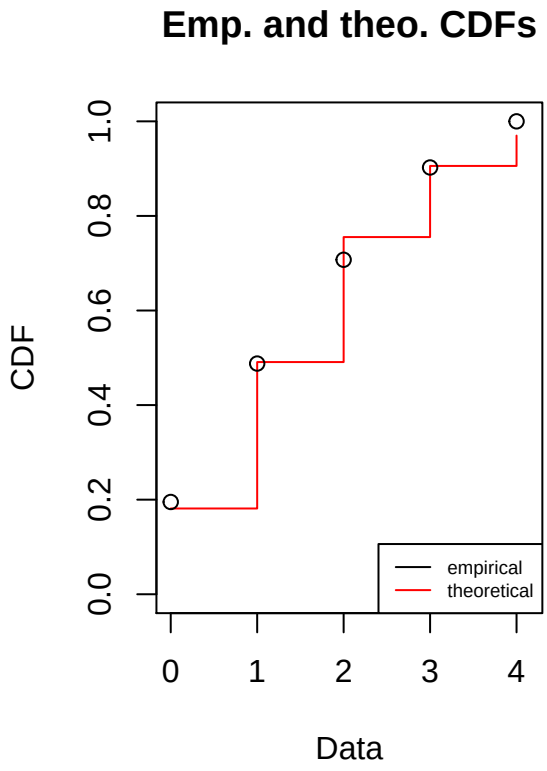

Emp. and theo. CDFs
CDF
0.0
0.2
0.4
0.6
0.8
1.0
empirical
theoretical
0.0
1.0
2.0
3.0
Data

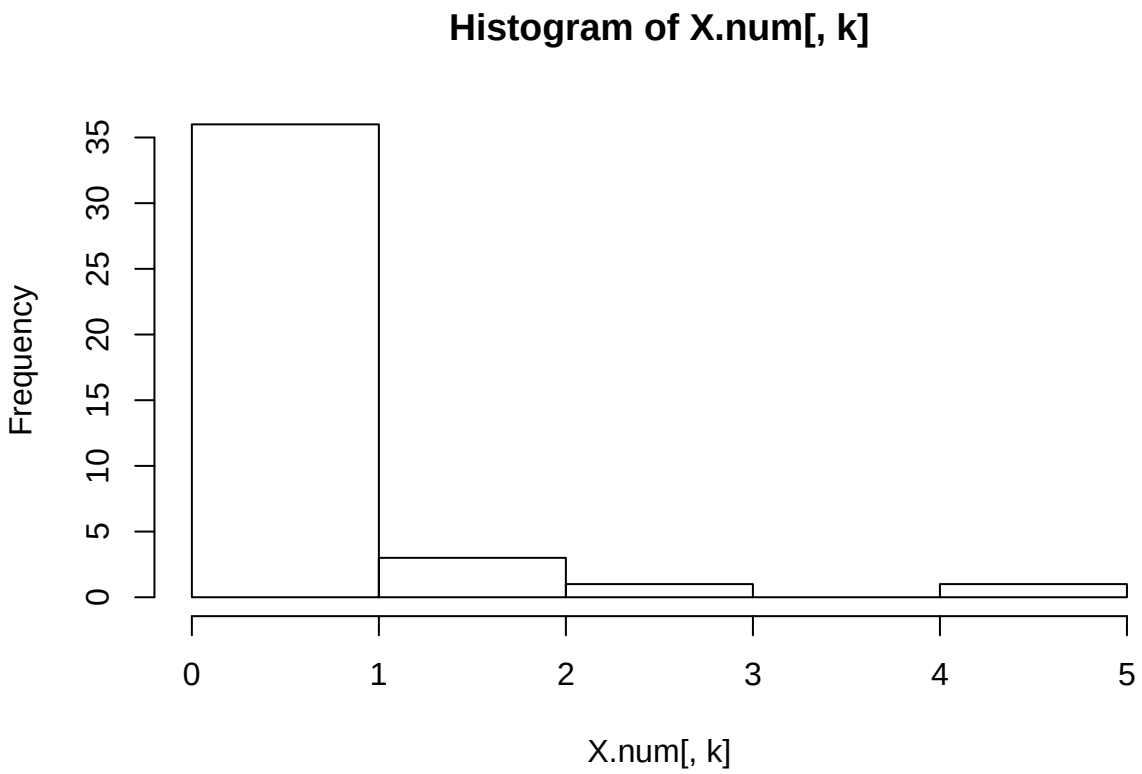

Histogram of X.num[, k]
Frequency
0
2
4
6
8
10
12
0
1
2
3
4
X.num[, k]

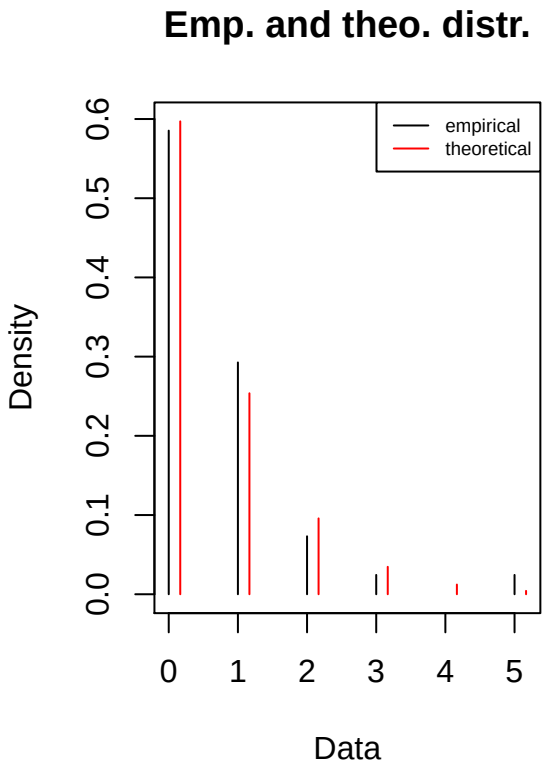


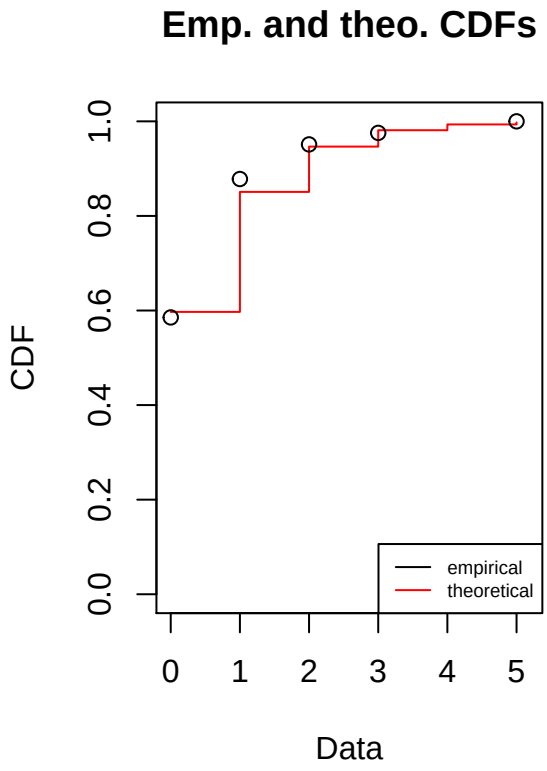

```
## Warning in theta.ml(Y, mu, sum(w), w, limit = control$maxit, trace =
## control$trace > : iteration limit reached

## Warning in theta.ml(Y, mu, sum(w), w, limit = control$maxit, trace =
## control$trace > : iteration limit reached
```

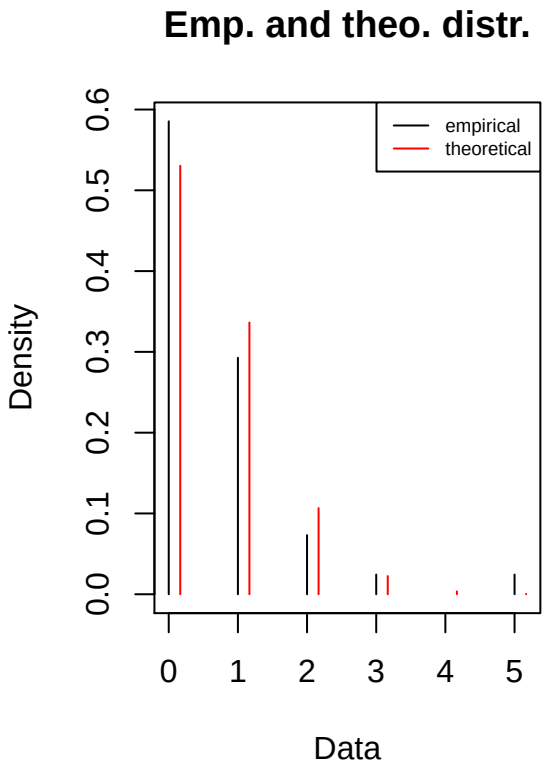

Emp. and theo. distr.
Density
0.00
0.10
0.20
0.30
empirical
theoretical
0
1
2
3
4
Data

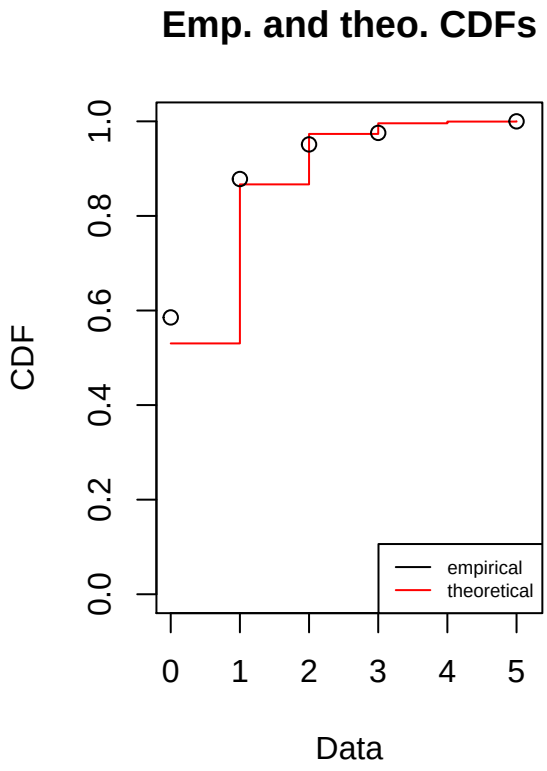

Emp. and theo. CDFs
CDF
0.0
0.2
0.4
0.6
0.8
1.0
empirical
theoretical
0
1
2
3
4
Data

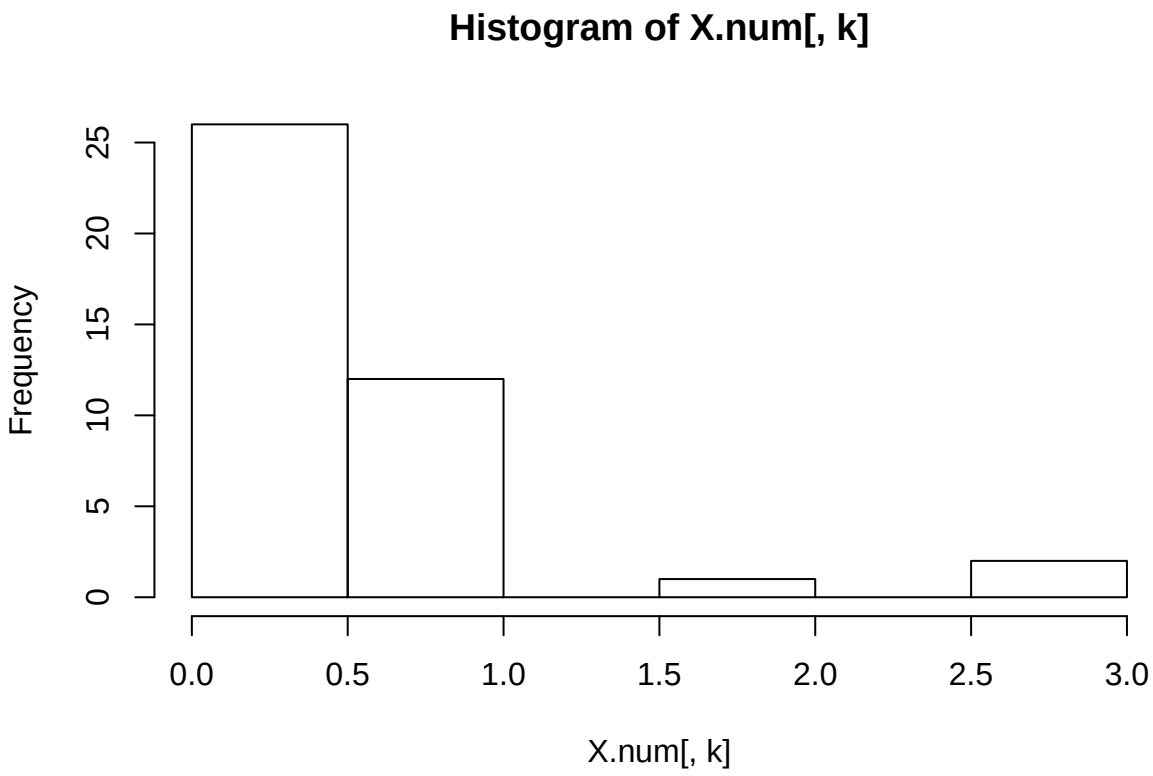

Histogram of X.num[, k]
Frequency
0
5
10
15
20
25
30
35
0
1
2
3
4
5
X.num[, k]

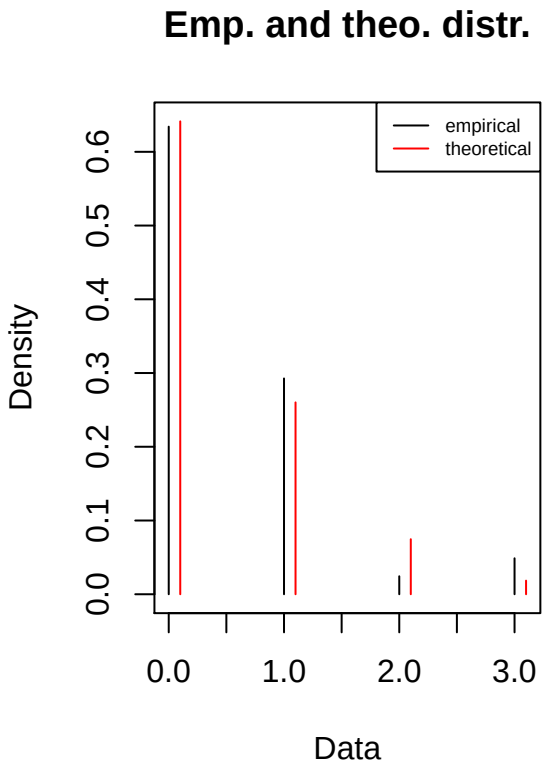

Emp. and theo. distr.
Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
empirical
theoretical
0
1
2
3
4
5
Data

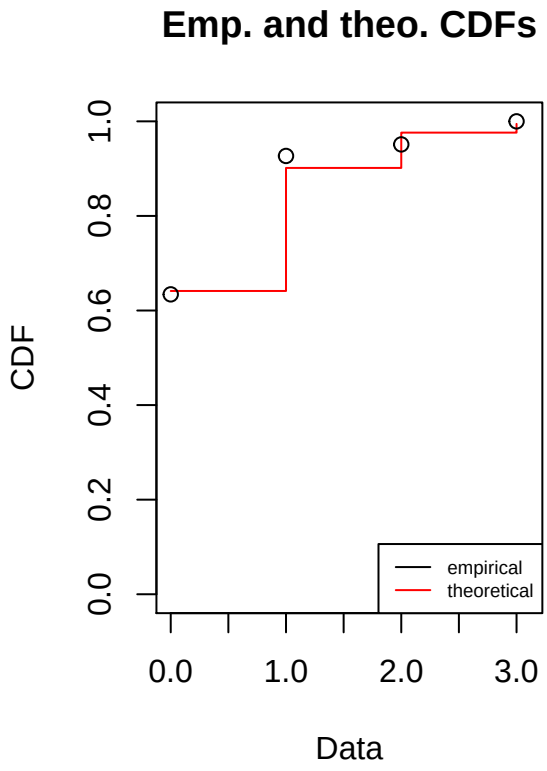

Emp. and theo. CDFs
CDF
0.0
0.2
0.4
0.6
0.8
1.0
empirical
theoretical
0
1
2
3
4
5
Data

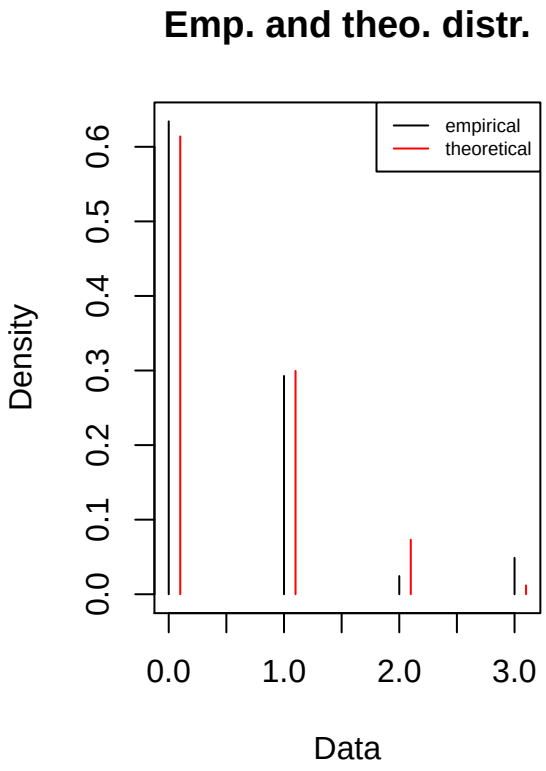

Emp. and theo. distr.
Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
empirical
theoretical
0
1
2
3
4
5
Data

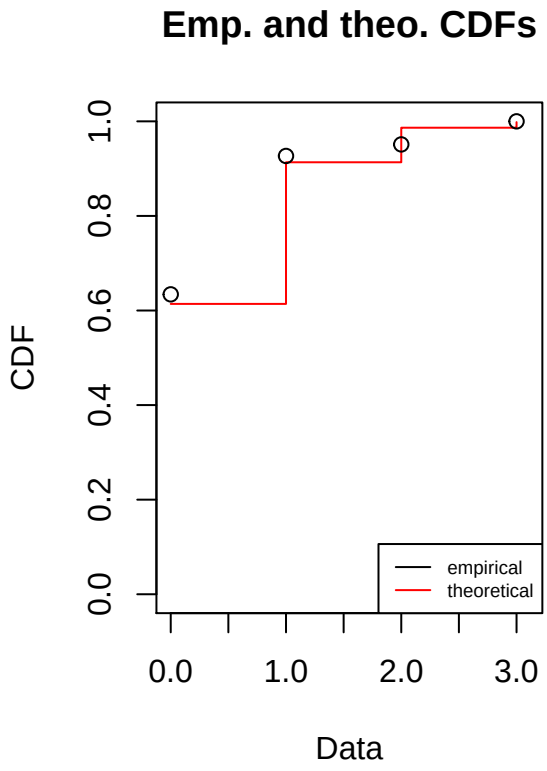

Emp. and theo. CDFs
CDF
0.0
0.2
0.4
0.6
0.8
1.0
empirical
theoretical
0
1
2
3
4
5
Data

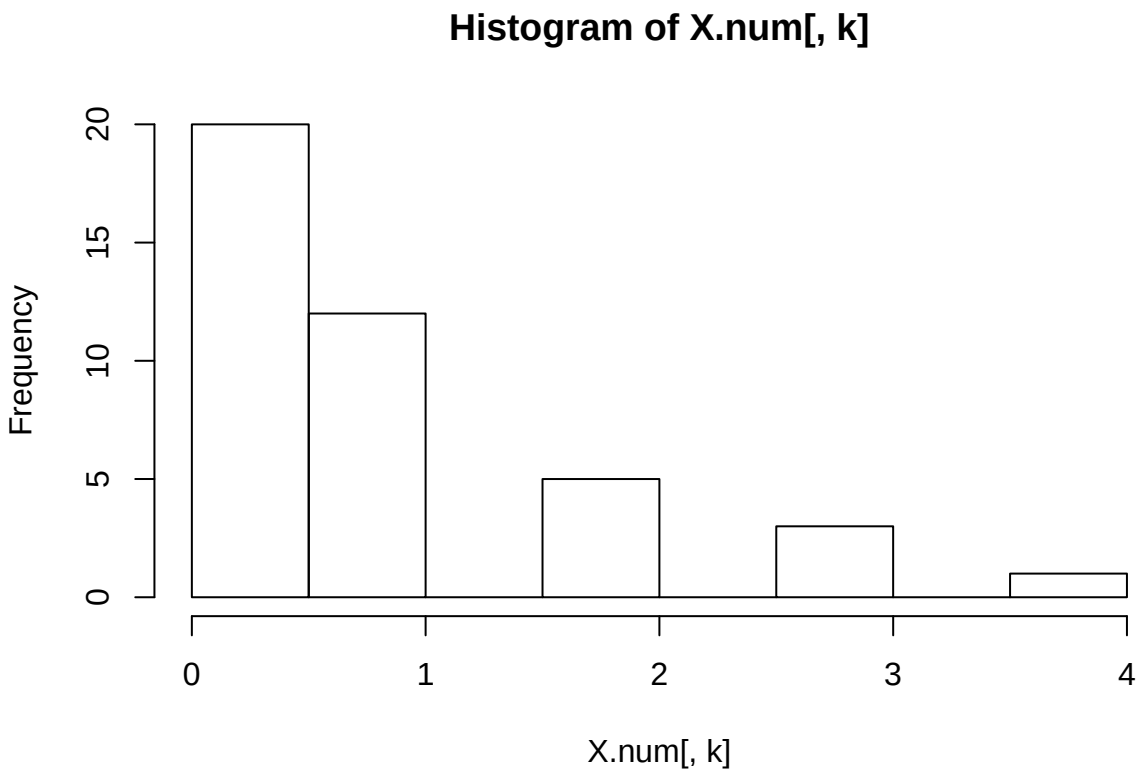

Histogram of X.num[, k]
Frequency
0
5
10
15
20
25
0.0
0.5
1.0
1.5
2.0
2.5
3.0
X.num[, k]

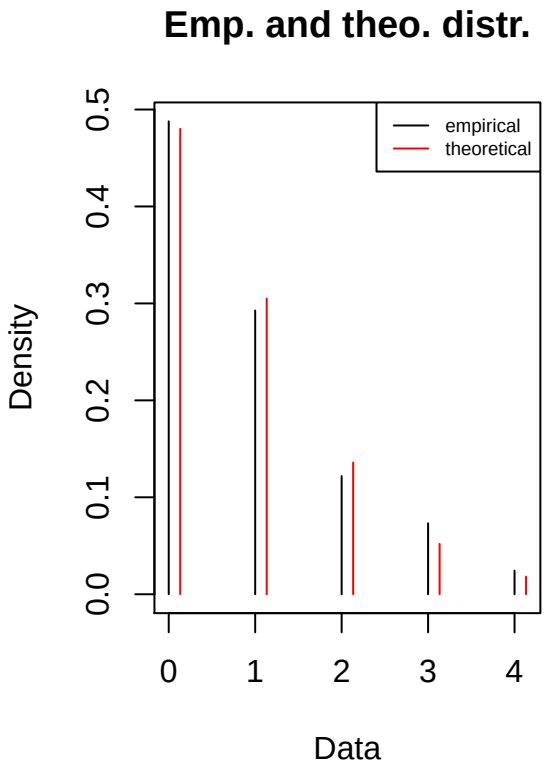

Emp. and theo. distr.
empirical
theoretical
Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.0
1.0
2.0
3.0
Data

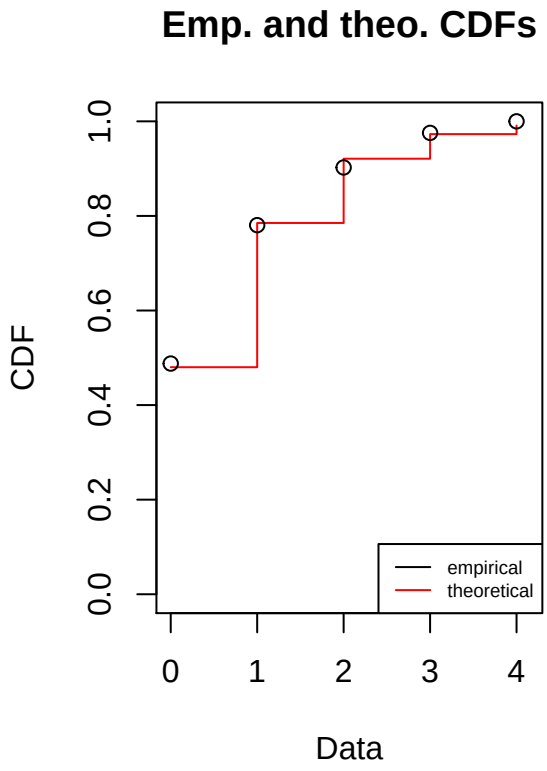

Emp. and theo. CDFs
CDF
0.0
0.2
0.4
0.6
0.8
1.0
empirical
theoretical
0.0
1.0
2.0
3.0
Data

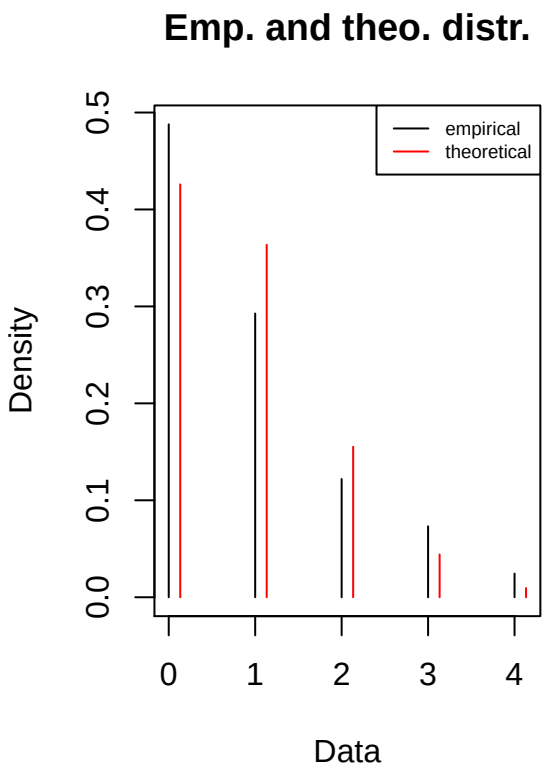

Emp. and theo. distr.
empirical
theoretical
Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.0
1.0
2.0
3.0
Data

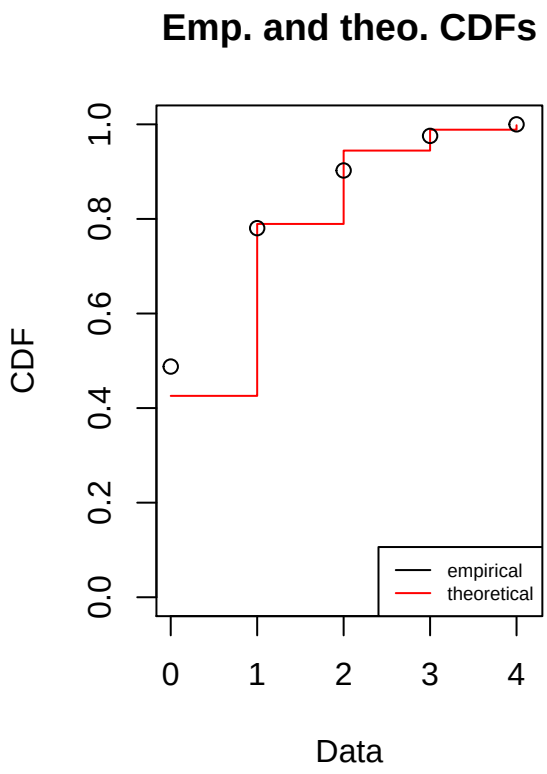

Emp. and theo. CDFs
CDF
0.0
0.2
0.4
0.6
0.8
1.0
empirical
theoretical
0.0
1.0
2.0
3.0
Data


Histogram of X.num[, k]
Frequency
0
5
10
15
20
0
1
2
3
4
X.num[, k]


Emp. and theo. distr.
Density
0.0
0.1
0.2
0.3
0.4
0.5
0
1
2
3
4
Data
empirical
theoretical


Emp. and theo. CDFs
CDF
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
Data
empirical
theoretical






```
colnames(mle.nb.glm.params) <- c(colnames(snb$coefficients),colnames(snb$coefficients))

colnames(mle.pois.glm.params) <- c(colnames(sp$coefficients),colnames(sp$coefficients))
library(pander)

pander(mle.nb.glm.params)
```

Table 1: Table continues below

Estimate	Std. Error	t value	Pr(>|t|)	Estimate	Std. Error	t value
-0.01112	0.3868	-0.02876	0.9772	-0.01508	0.01725	-0.8741
0.01262	0.2792	0.04522	0.9642	0.02806	0.01028	2.73
-0.1967	0.3437	-0.5724	0.5703	0.01123	0.01356	0.828
0.3951	0.3504	1.127	0.2665	-0.02732	0.01636	-1.67
-1.325	0.4159	-3.187	0.002834	0.05012	0.01406	3.564
0.2638	0.2475	1.066	0.2931	0.0124	0.009672	1.282
0.02877	0.436	0.06599	0.9477	-0.02511	0.01991	-1.261
-1.034	0.5317	-1.944	0.05909	0.01436	0.02081	0.69
-0.5377	0.429	-1.253	0.2176	0.01715	0.01672	1.026

Pr(>|t|)
0.3874
0.009462
0.4127

Pr(>|t|)
0.1028
0.0009826
0.2074
0.2148
0.4943
0.3114

```
pander(mle.pois.glm.params)
```

Table 3: Table continues below

Estimate	Std. Error	z value	Pr(>|z|)	Estimate	Std. Error	z value
-0.02045	0.3472	-0.05891	0.953	-0.01461	0.01557	-0.9385
0.01857	0.2683	0.0692	0.9448	0.0278	0.009817	2.832
-0.1918	0.3293	-0.5825	0.5602	0.011	0.01295	0.8492
0.3631	0.3029	1.199	0.2306	-0.02566	0.01447	-1.773
-1.325	0.4539	-2.919	0.003506	0.05012	0.01535	3.266
0.2638	0.2602	1.014	0.3106	0.0124	0.01017	1.22
0.01881	0.3579	0.05255	0.9581	-0.02458	0.017	-1.446
-1.034	0.4922	-2.101	0.03565	0.01437	0.01906	0.7538
-0.5177	0.376	-1.377	0.1686	0.01624	0.01444	1.124

Pr(>|z|)
0.348
0.004629
0.3958
0.07627
0.001092
0.2226
0.1481
0.451
0.2608

Fit JAGS GLM Models for Poisson

```
library(rjags)
```

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

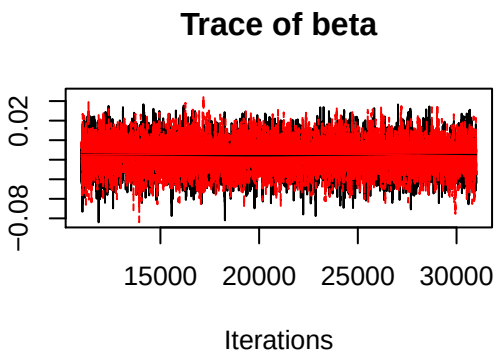
```
## Linked to JAGS 4.3.0
```

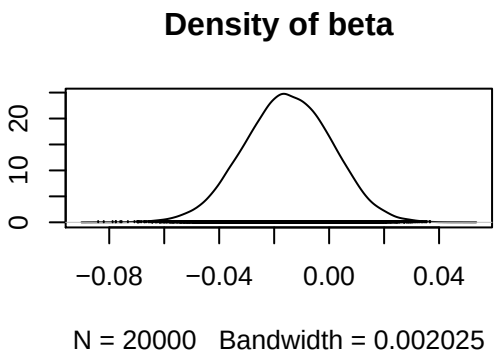
```
## Loaded modules: basemod,bugs
```

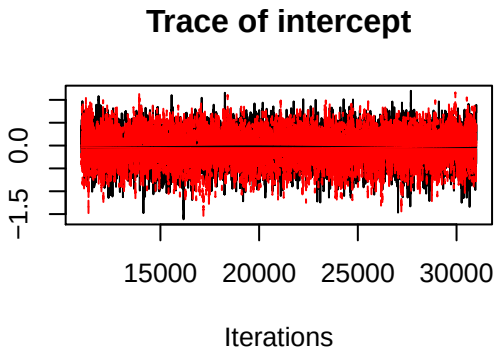
```
library(coda)
model_code = '
model
{
    ## Likelihood
```

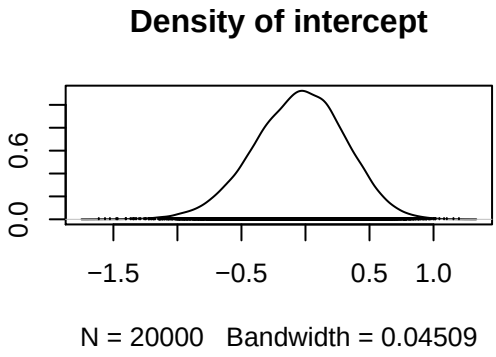
```
    for(i in 1:N){
      Y[i] ~ dpois(lambda[i])
      log(lambda[i]) <- mu[i]
      mu[i] <- intercept + beta*t[i]
    }

  ## Priors
  beta ~ dnorm(mu.beta,tau.beta)
  intercept ~ dnorm(mu.intercept,tau.intercept)
}
'
for(k in 1:5)
{
  print(paste("JAGS GLM Models for Poisson ", k,sep = " "))
  # Set up the data
  model_data = list(N = 41, t=seq(1:41),Y=X.num[,k],mu.beta=0,tau.beta=.0001,mu.intercept=0,tau.intercep
  # Choose the parameters to watch
  model_parameters =  c("beta", "intercept")
  n.chains =2;nSamples = 10000
  model <- jags.model(textConnection(model_code),data = model_data,n.chains = n.chains)#Compile Model G
  update(model, nSamples, progress.bar="none"); # Burnin
  out.coda  <- coda.samples(model, variable.names=model_parameters,n.iter=2*nSamples)
  plot(out.coda)
  #assess the posteriors??? stationarity, by looking at the Heidelberg-Welch convergence diagnostic:
  heidel.diag(out.coda)
  # check that our chain???s length is satisfactory.
  raftery.diag(out.coda)

  geweke.diag(out.coda)

  if(n.chains > 1)
  {
    gelman.srf <-gelman.diag(out.coda)
    plot(gelman.srf$psrf,main = "Gelman Diagnostic")
  }

  chains.ess <- lapply(out.coda,effectiveSize)
  first.chain.ess <- chains.ess[1]
  plot(unlist(first.chain.ess), main="Effective Sample Size")
}
```

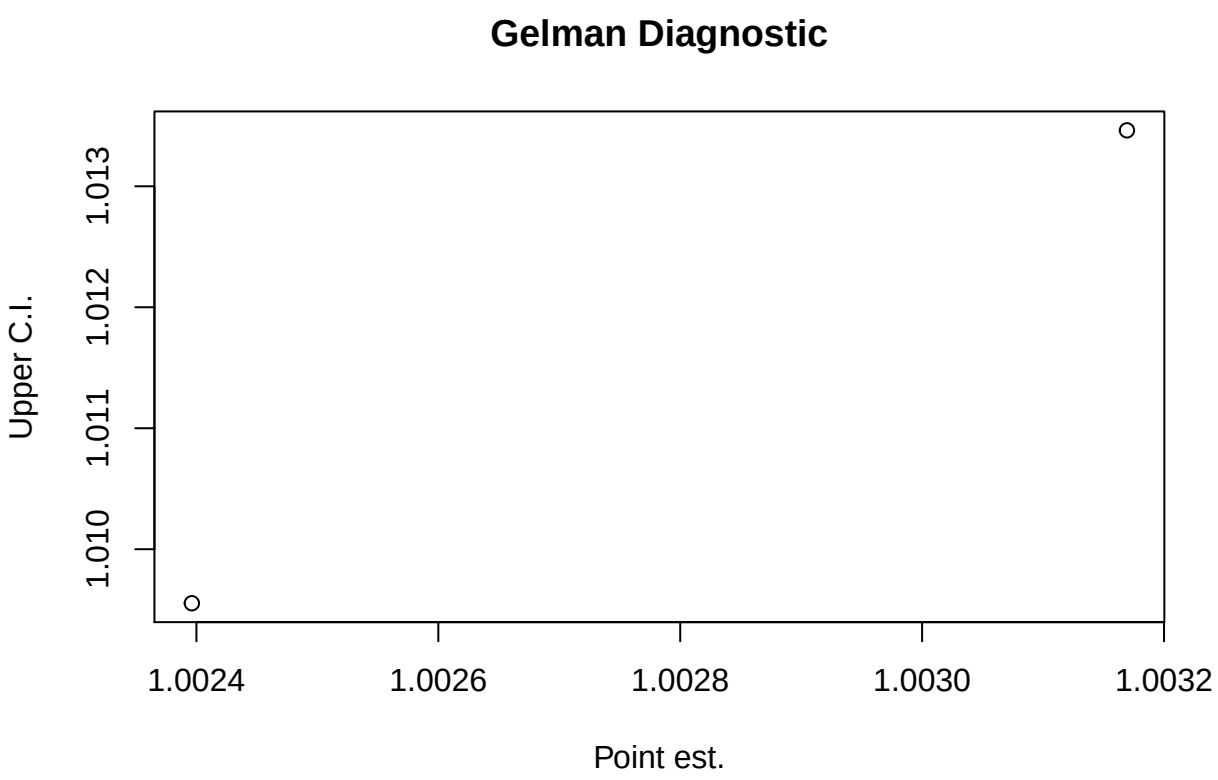
```
## [1] "JAGS GLM Models for Poisson  1"
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 41
##    Unobserved stochastic nodes: 2
##    Total graph size: 212
##
## Initializing model
```

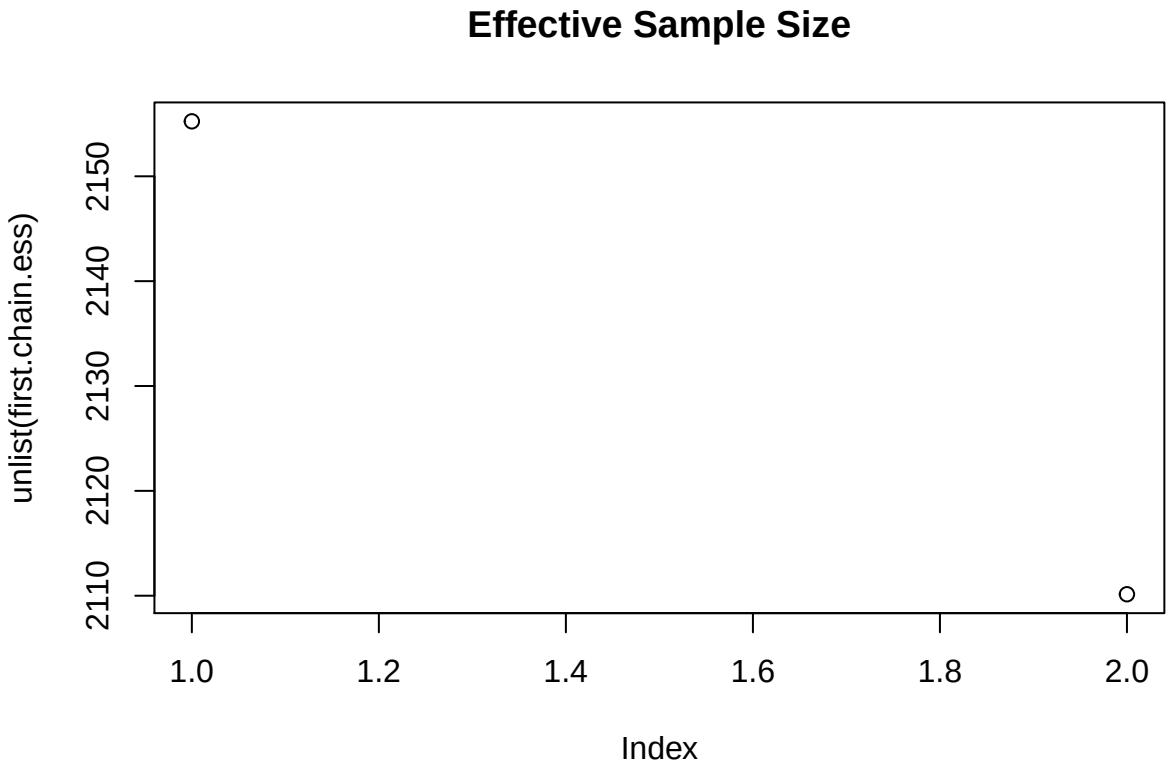

Trace of beta
−0.08
0.02
15000
20000
25000
30000
Iterations


Density of beta
0
10
20
−0.08
−0.04
0.00
0.04
N = 20000 Bandwidth = 0.002025

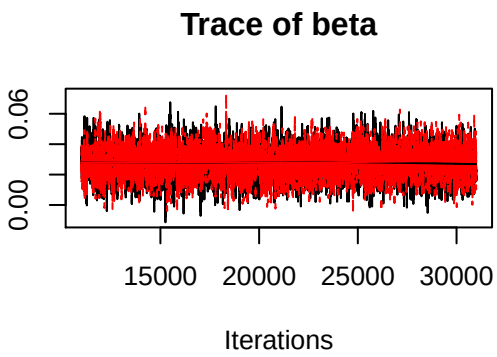

Trace of intercept
−1.5
0.0
15000
20000
25000
30000
Iterations

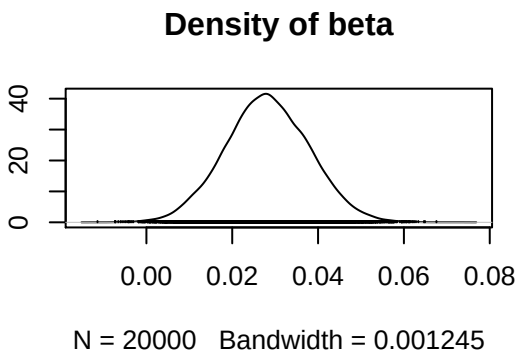

Density of intercept
0.0
0.6
−1.5
−0.5
0.5
1.0
N = 20000 Bandwidth = 0.04509

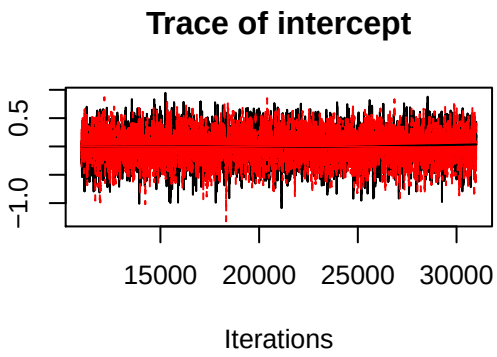

Gelman Diagnostic
Upper C.I.
1.010
1.011
1.012
1.013
1.0024
1.0026
1.0028
1.0030
1.0032
Point est.

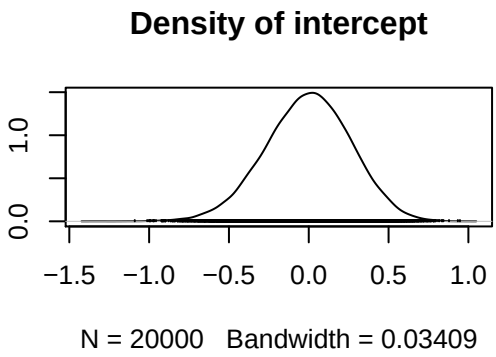


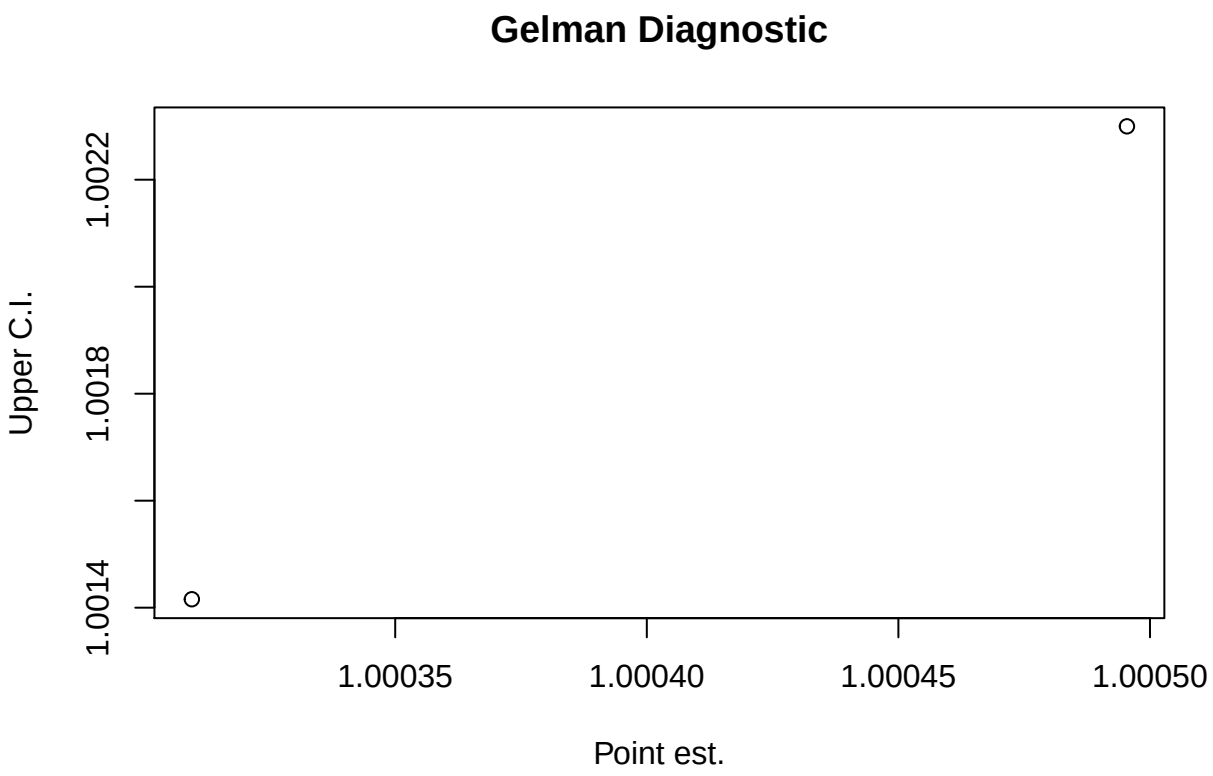

```
## [1] "JAGS GLM Models for Poisson  2"
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 41
##    Unobserved stochastic nodes: 2
##    Total graph size: 212
##
## Initializing model
```

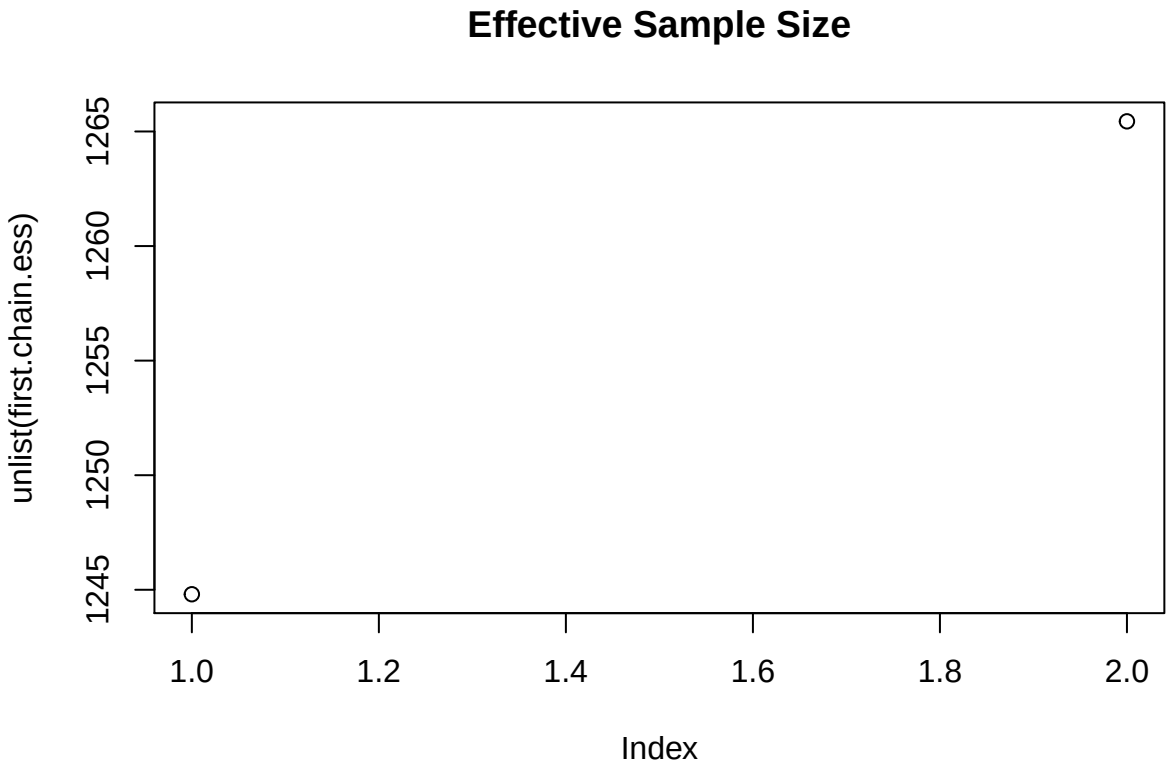

Trace of beta
0.00
0.06
15000
20000
25000
30000
Iterations


Density of beta
0
20
40
0.00
0.02
0.04
0.06
0.08
N = 20000 Bandwidth = 0.001245

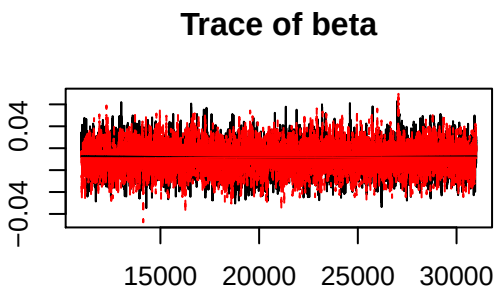

Trace of intercept
−1.0
0.5
15000
20000
25000
30000
Iterations

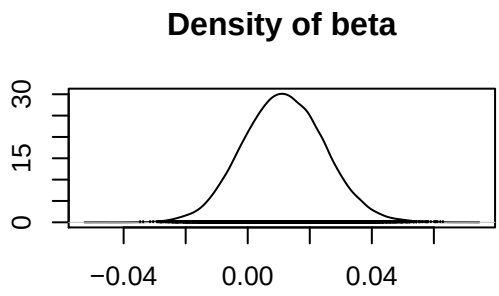

Density of intercept
0.0
1.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
N = 20000 Bandwidth = 0.03409

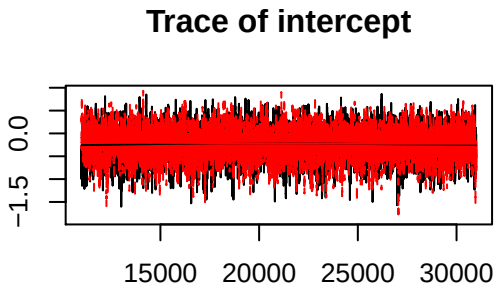

Gelman Diagnostic
Upper C.I.
1.0014
1.0018
1.0022
1.00035
1.00040
1.00045
1.00050
Point est.

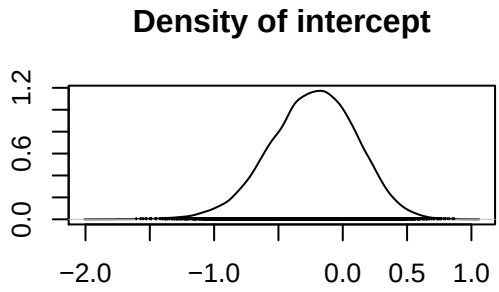


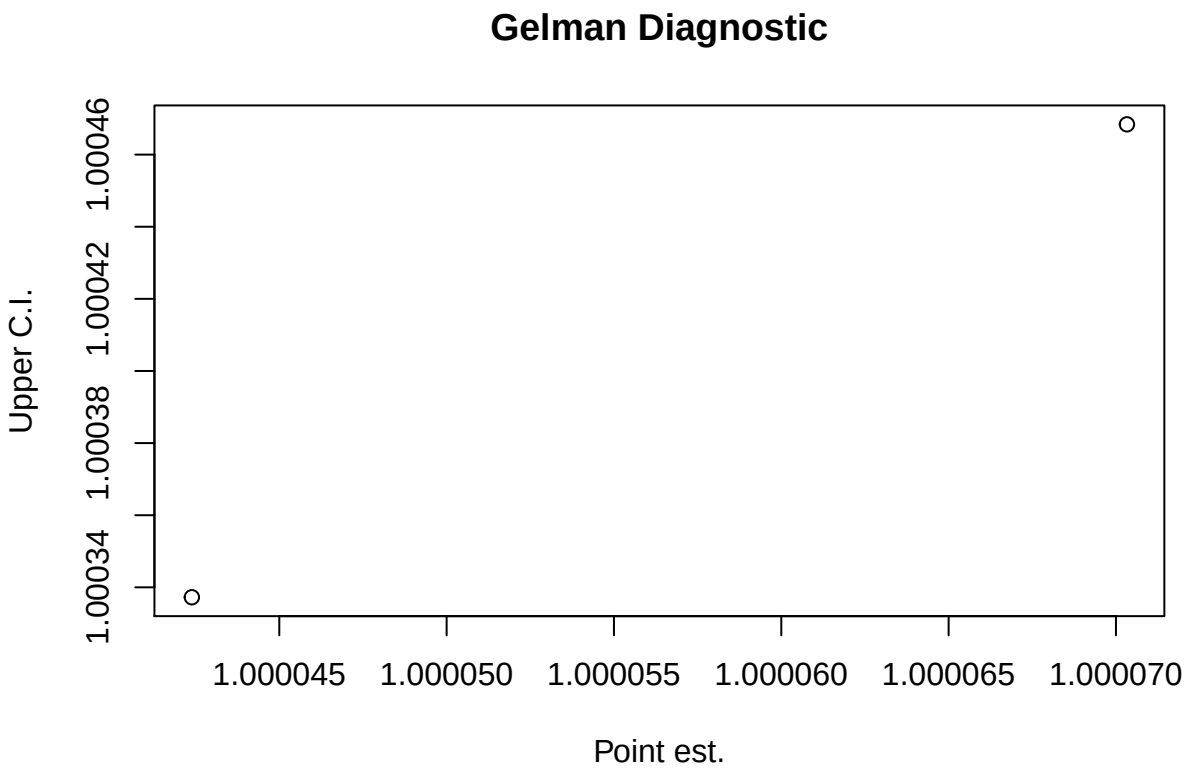

```
## [1] "JAGS GLM Models for Poisson  3"
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 41
##    Unobserved stochastic nodes: 2
##    Total graph size: 212
##
## Initializing model
```

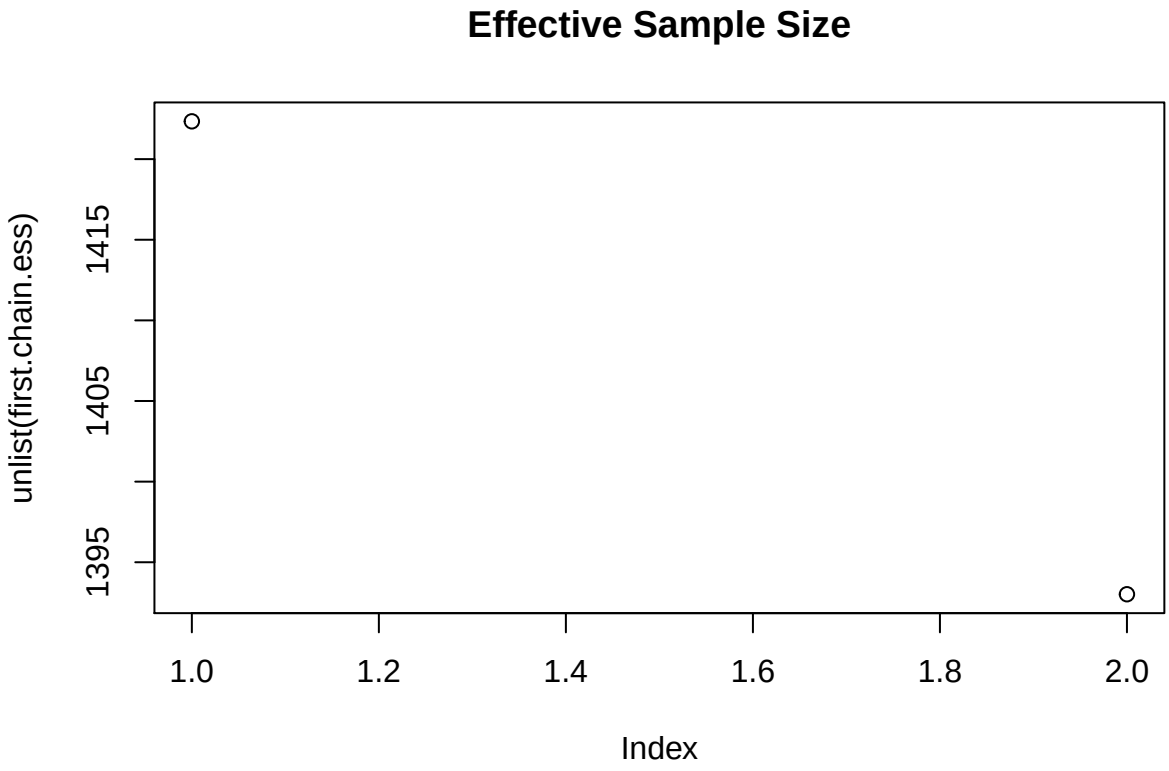

Trace of beta
−0.04
0.04
15000
20000
25000
30000
Iterations


Density of beta
0
15
30
−0.04
0.00
0.04
N = 20000 Bandwidth = 0.001664

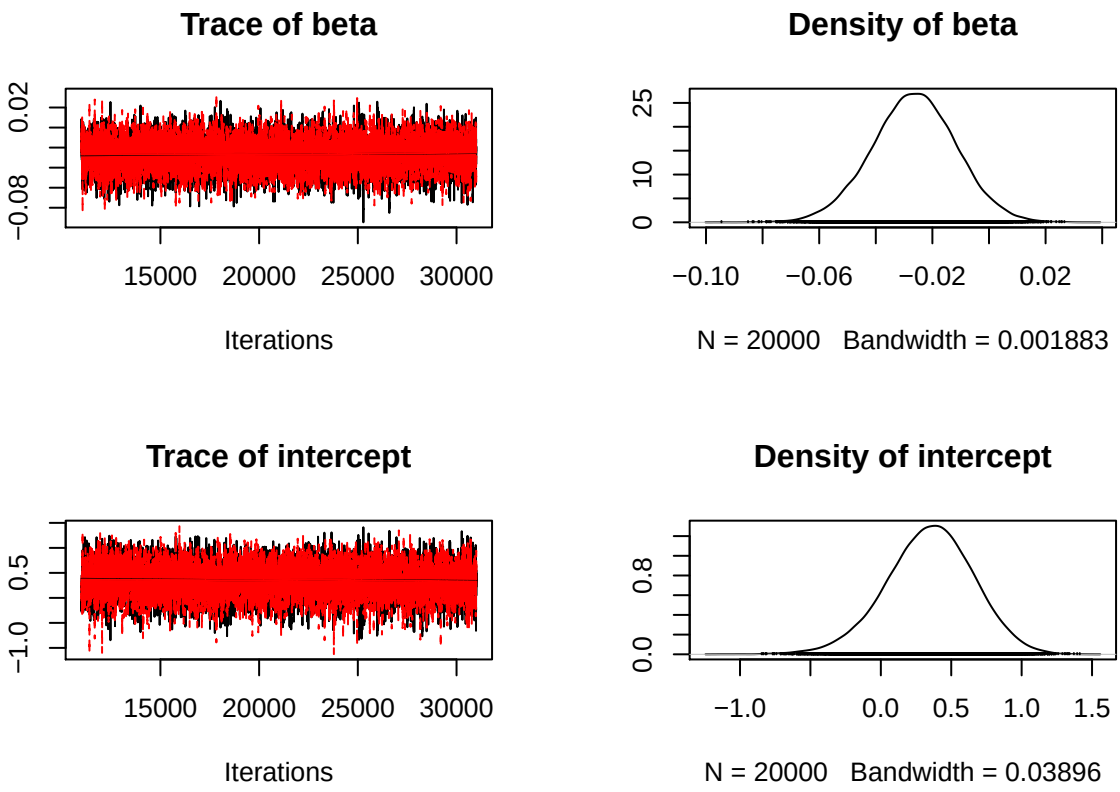

Trace of intercept
−1.5
0.0
15000
20000
25000
30000
Iterations

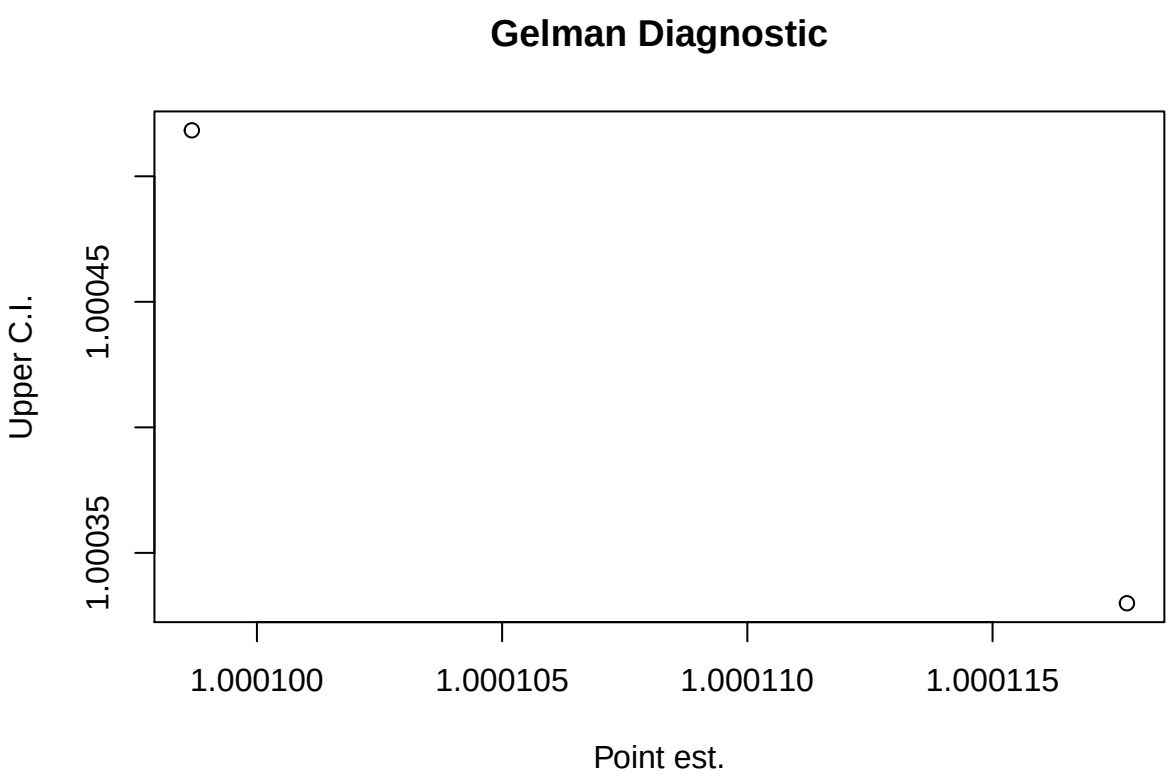

Density of intercept
0.0
0.6
1.2
−2.0
−1.0
0.0
0.5
1.0
N = 20000 Bandwidth = 0.04271

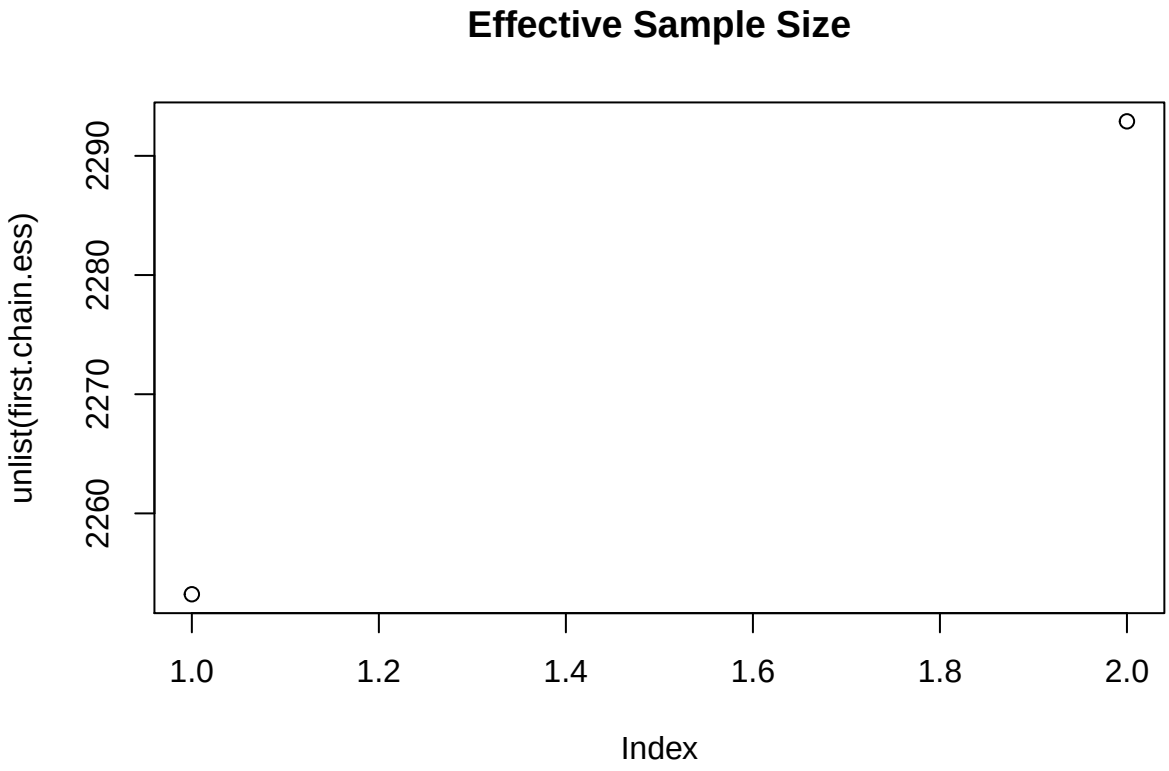

Gelman Diagnostic
Upper C.I.
1.00034
1.00038
1.00042
1.00046
1.000045
1.000050
1.000055
1.000060
1.000065
1.000070
Point est.



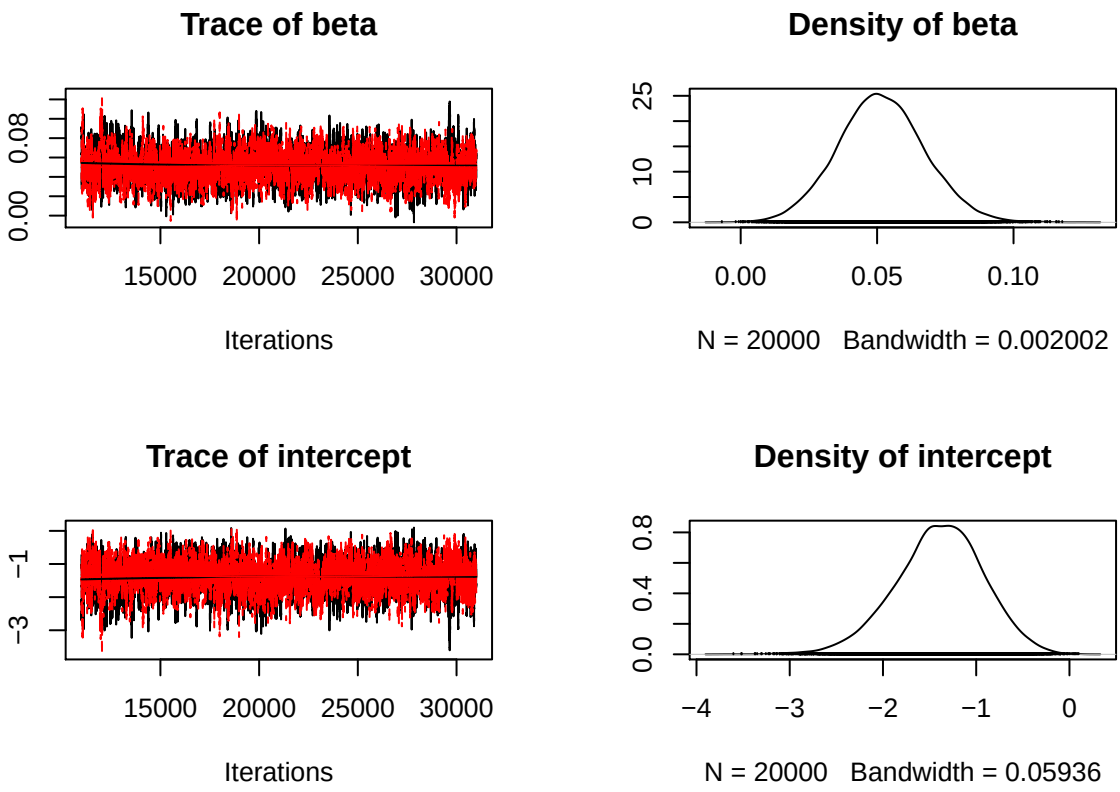

```
## [1] "JAGS GLM Models for Poisson  4"
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 41
##    Unobserved stochastic nodes: 2
##    Total graph size: 212
##
## Initializing model
```

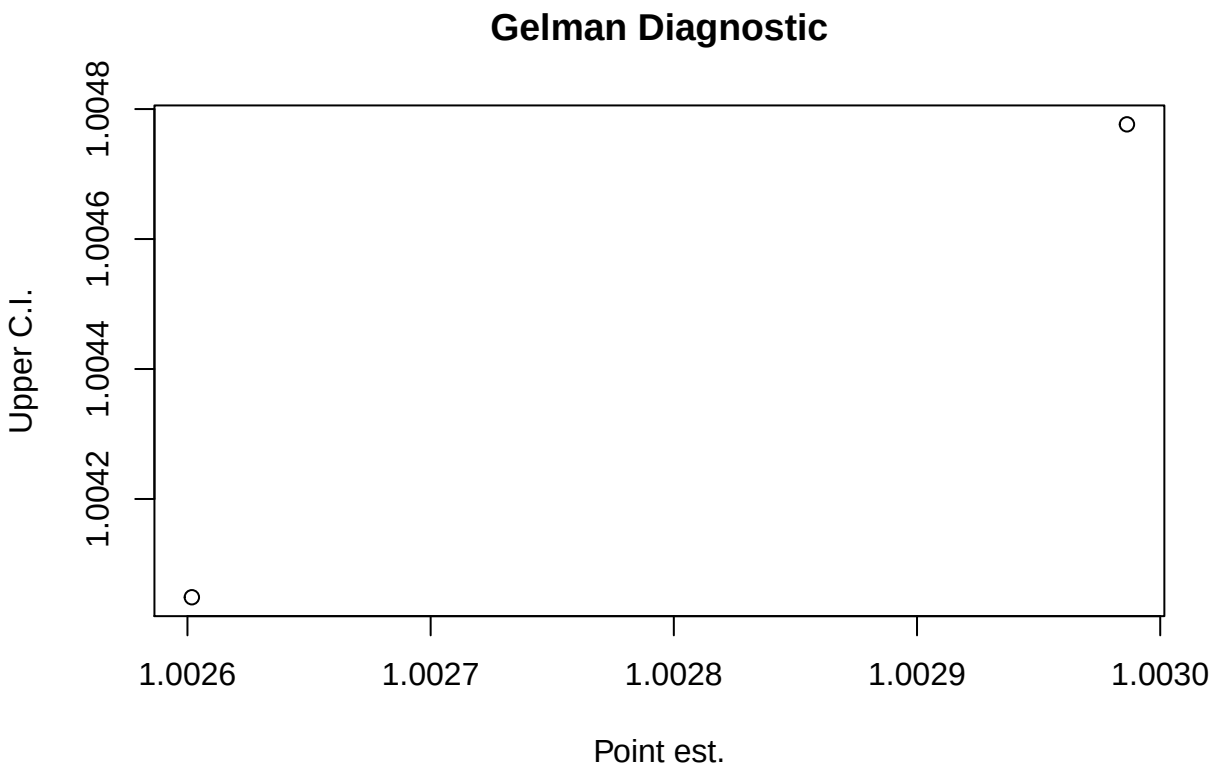

Trace of beta
0.02
−0.08
15000
20000
25000
30000
Iterations
Density of beta
25
10
0
−0.10
−0.06
−0.02
0.02
N = 20000 Bandwidth = 0.001883
Trace of intercept
0.5
−1.0
15000
20000
25000
30000
Iterations
Density of intercept
0.8
0.0
−1.0
0.0
0.5
1.0
1.5
N = 20000 Bandwidth = 0.03896

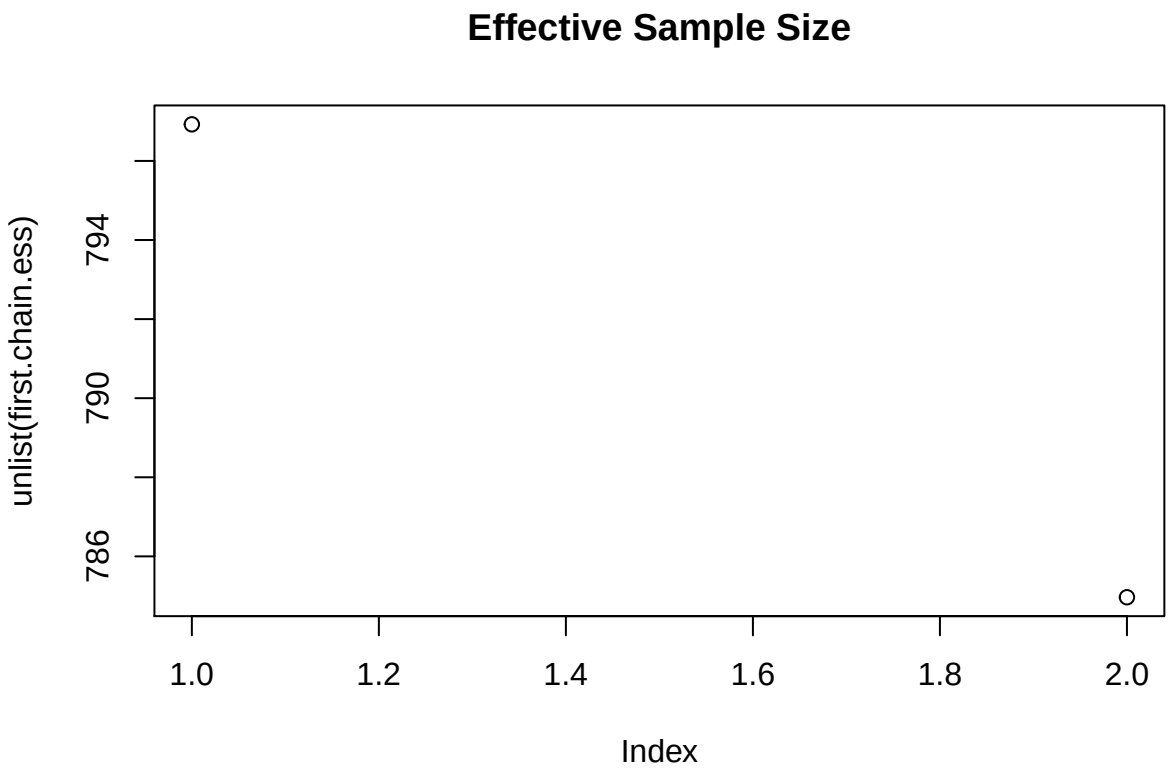

Gelman Diagnostic
Upper C.I.
1.00045
1.00035
1.000100
1.000105
1.000110
1.000115
Point est.




```
## [1] "JAGS GLM Models for Poisson  5"
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 41
##    Unobserved stochastic nodes: 2
##    Total graph size: 212
##
## Initializing model
```


Trace of beta
0.00
0.08
15000
20000
25000
30000
Iterations
Density of beta
0
10
25
0.00
0.05
0.10
N = 20000 Bandwidth = 0.002002
Trace of intercept
−3
−1
15000
20000
25000
30000
Iterations
Density of intercept
0.0
0.4
0.8
−4
−3
−2
−1
0
N = 20000 Bandwidth = 0.05936


Gelman Diagnostic
Upper C.I.
1.0042
1.0044
1.0046
1.0048
1.0026
1.0027
1.0028
1.0029
1.0030
Point est.



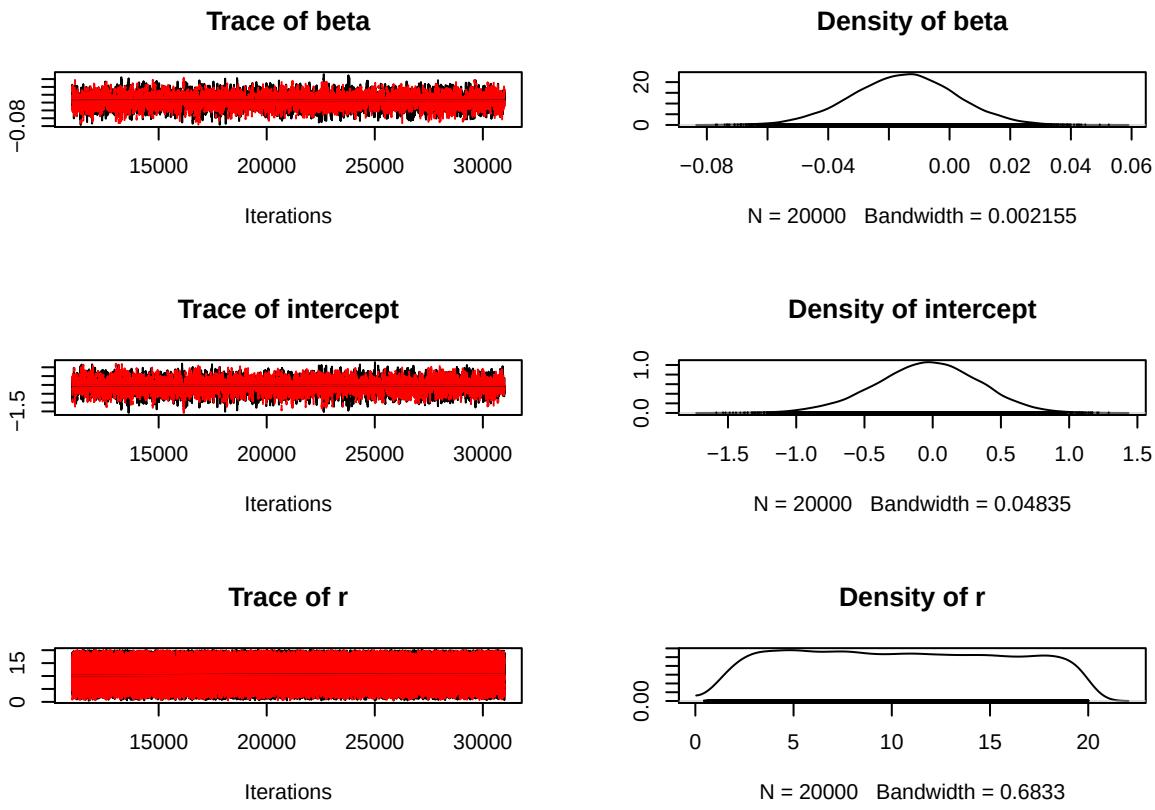
Fit JAGS GLM Models for Negative Binomial

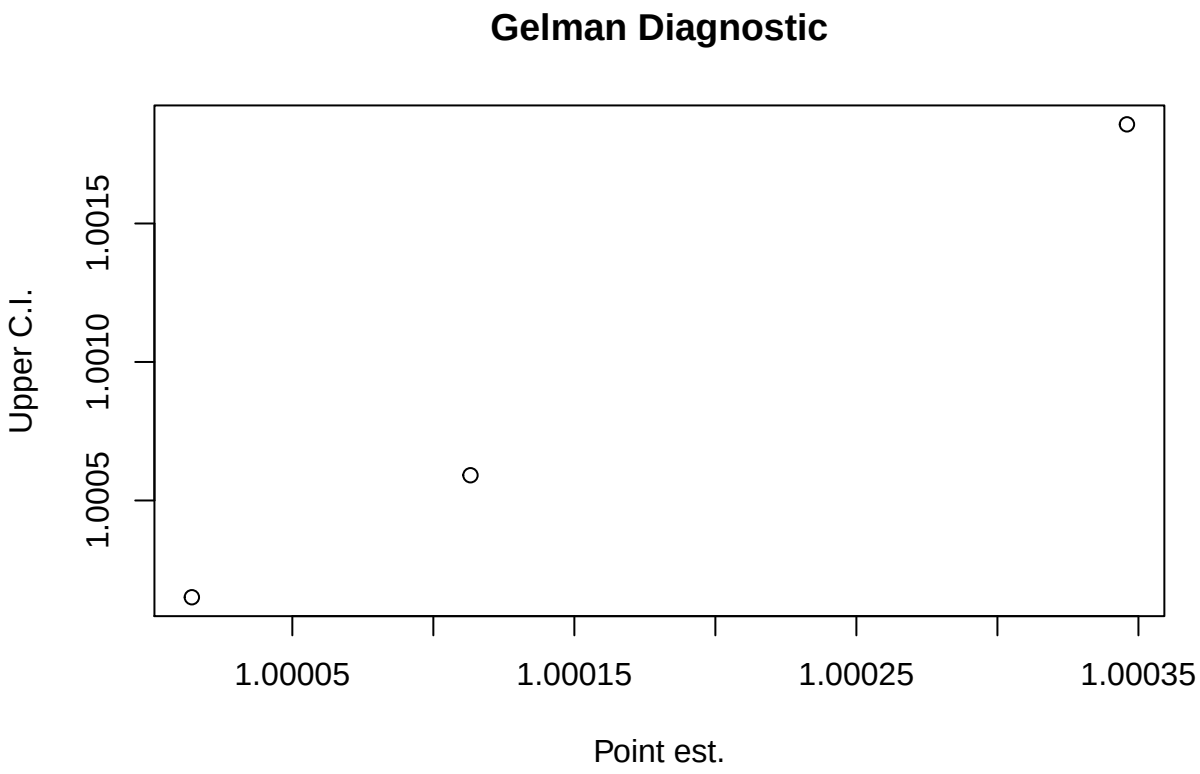
```
######################################## Fit JAGS GLM Models for Negative Binomial

#
# Direct estimation of the p and r parameters in a negative binomial distribution could involve bad aut
#
# The model specifications:
#
# Parameterized by p and r:
# model {
#     for( i in 1 : N ) {
#       y[i] ~ dnegbin( p , r )
#     }
#     p ~ dbeta(1.001,1.001)
#     r ~ dgamma(0.01,0.01)
#     m <- r*(1-p)/p
#     v <- r*(1-p)/(p*p)
# }
#
# Parameterized by m and r:
# model {
#     for( i in 1 : N ) {
#       y[i] ~ dnegbin( p , r )
```

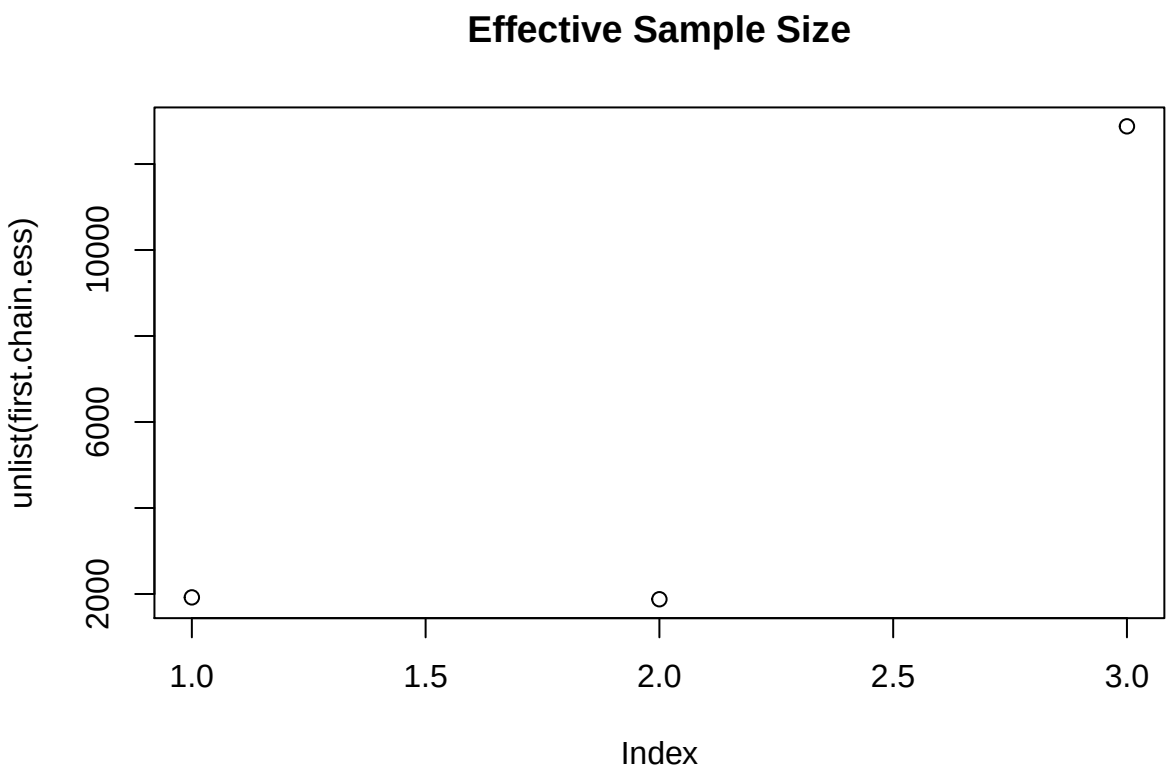
```
#     }
#     p <- r/(r+m)
#    )
#     m ~ dgamma(0.01,0.01)
#     v <- r*(1-p)/(p*p)
# }
library(rjags)
library(coda)
model_code = '
model
{
    ## Likelihood
    for(i in 1:N){
      Y[i] ~ dnegbin(p[i],r)

      p[i] <- r/(r+lambda[i])
      log(lambda[i]) <- mu[i]
      mu[i] <- intercept + beta*t[i]
    }

  ## Priors
  beta ~ dnorm(mu.beta,tau.beta)
  intercept ~ dnorm(mu.intercept,tau.intercept)
  r ~ dunif(0,20)
  #r ~ dgamma(0.01,0.01)
}
'
for(k in 1:5)
{
  print(paste("JAGS GLM Models for Negative Binomial ", k,sep = " "))
  # Set up the data
  model_data = list(N = 41, t=seq(1:41),Y=X.num[,k],mu.beta=0,tau.beta=.0001,mu.intercept=0,tau.intercep
  # Choose the parameters to watch
  model_parameters =  c("r","beta", "intercept")
  n.chains =2;nSamples = 10000
  model <- jags.model(textConnection(model_code),data = model_data,n.chains = n.chains)#Compile Model G
  update(model, nSamples, progress.bar="none"); # Burnin
  out.coda  <- coda.samples(model, variable.names=model_parameters,n.iter=2*nSamples)
  plot(out.coda)
  #assess the posteriors??? stationarity, by looking at the Heidelberg-Welch convergence diagnostic:
  heidel.diag(out.coda)
  # check that our chain???s length is satisfactory.
  raftery.diag(out.coda)

  geweke.diag(out.coda)

  if(n.chains > 1)
  {
    gelman.srf <-gelman.diag(out.coda)
    plot(gelman.srf$psrf,main = "Gelman Diagnostic")
  }

  chains.ess <- lapply(out.coda,effectiveSize)
```

```
  first.chain.ess <- chains.ess[1]
  plot(unlist(first.chain.ess), main="Effective Sample Size")
}
```

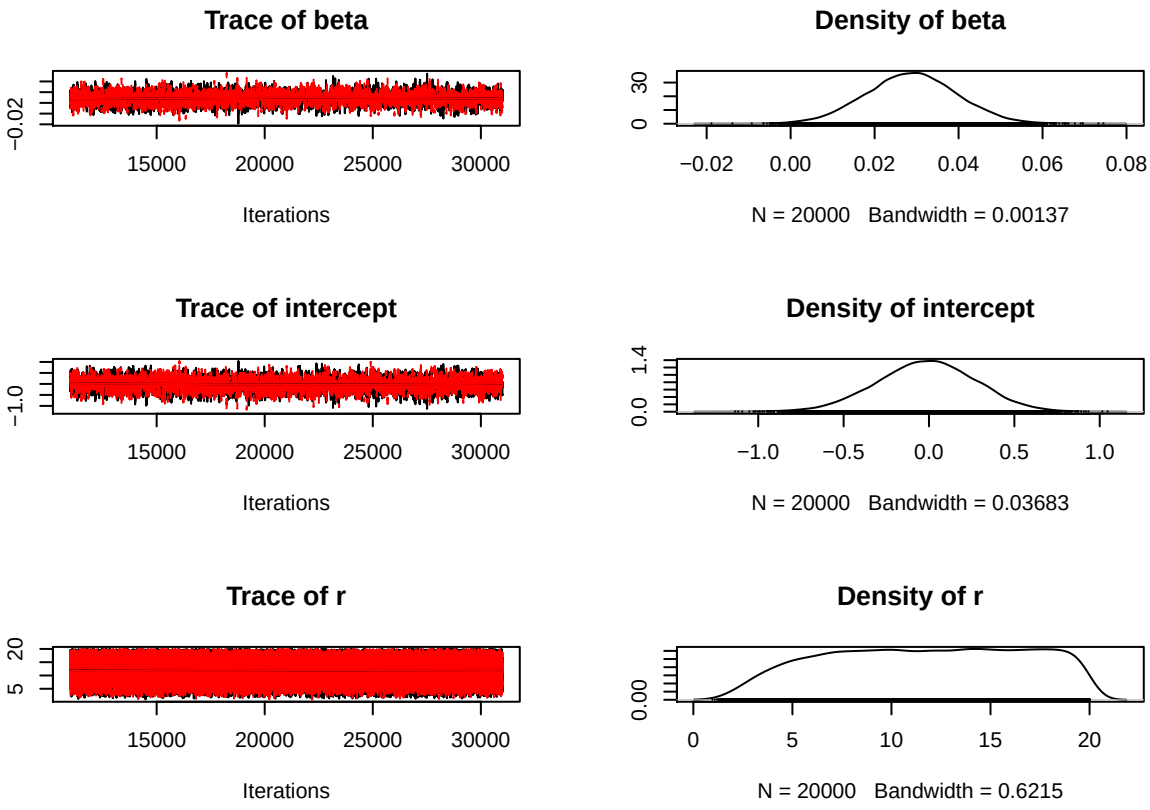
```
## [1] "JAGS GLM Models for Negative Binomial  1"
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 41
##    Unobserved stochastic nodes: 3
##    Total graph size: 297
##
## Initializing model
```

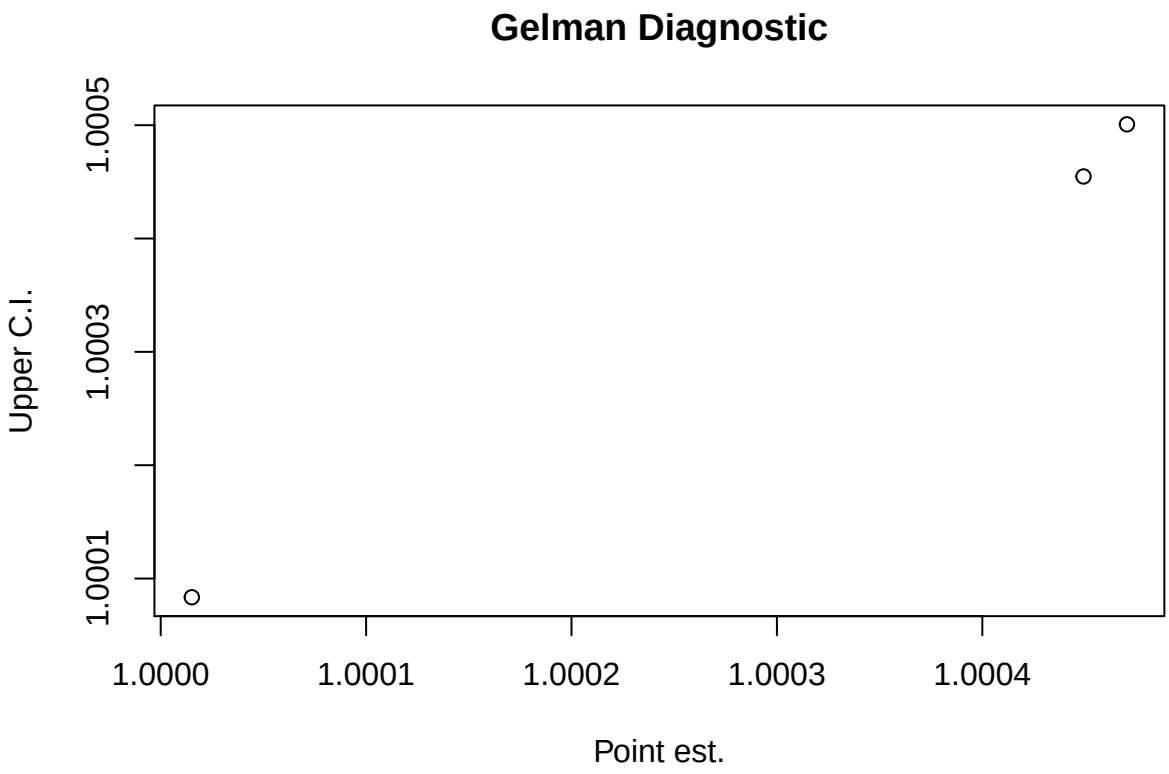


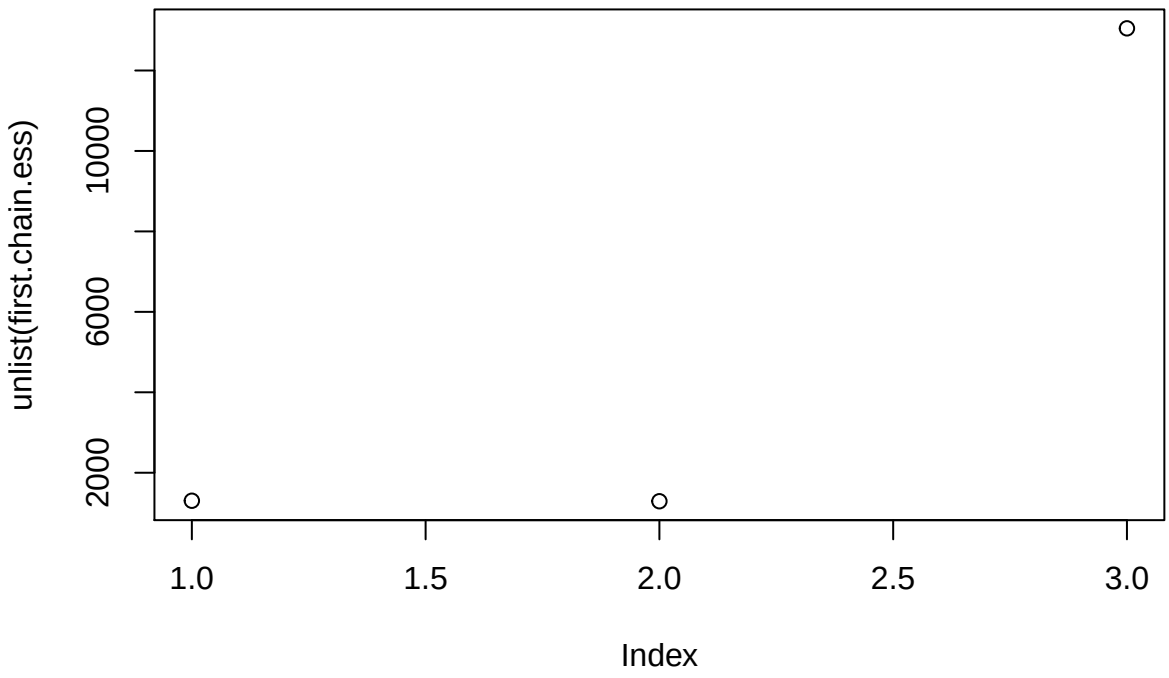

Gelman Diagnostic
Upper C.I.
1.0005
1.0010
1.0015
1.00005
1.00015
1.00025
1.00035
Point est.



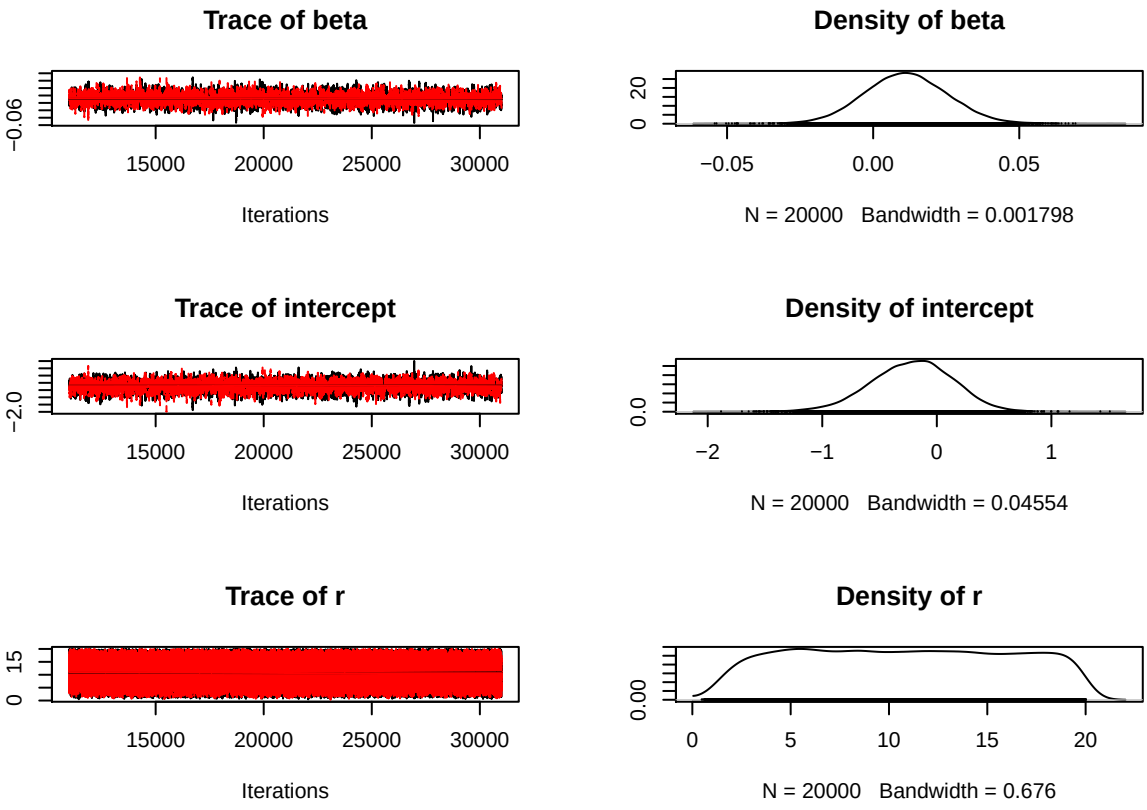

```
## [1] "JAGS GLM Models for Negative Binomial  2"
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 41
##    Unobserved stochastic nodes: 3
##    Total graph size: 297
##
## Initializing model
```

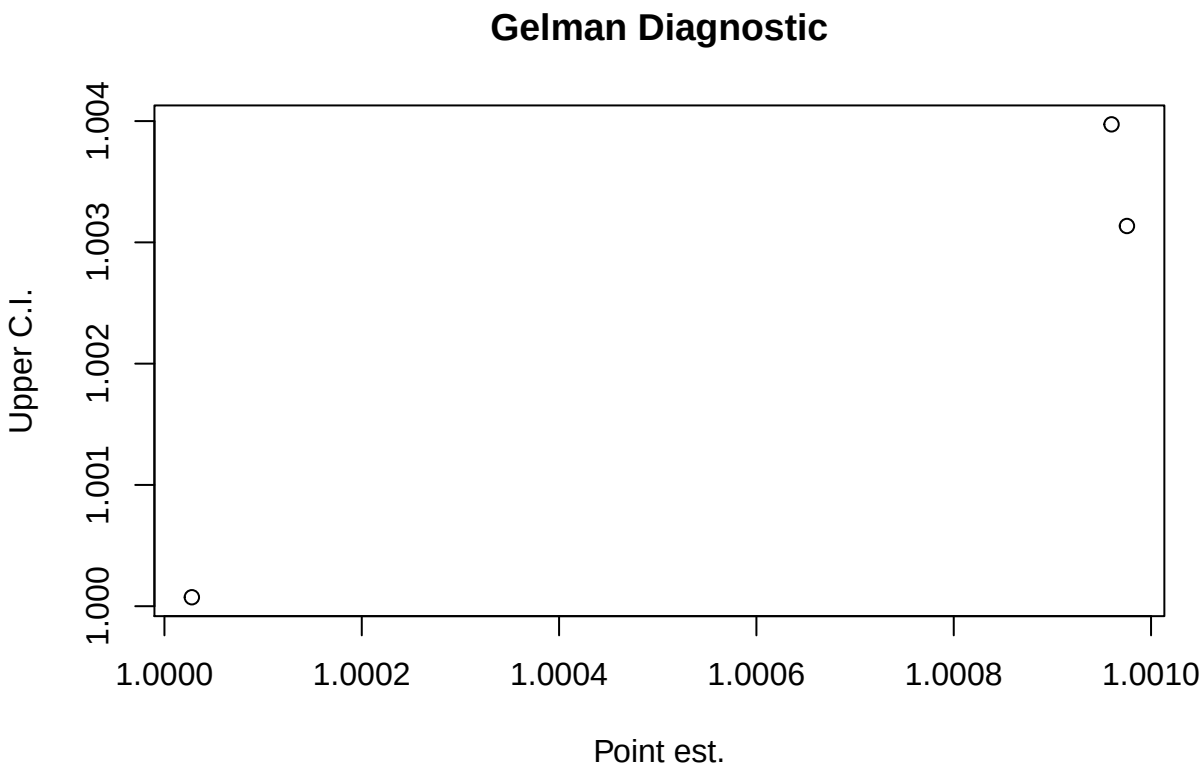

Trace of beta
−0.02
15000
20000
25000
30000
Iterations
Density of beta
0
30
−0.02
0.00
0.02
0.04
0.06
0.08
N = 20000 Bandwidth = 0.00137
Trace of intercept
−1.0
15000
20000
25000
30000
Iterations
Density of intercept
0.0
1.4
−1.0
−0.5
0.0
0.5
1.0
N = 20000 Bandwidth = 0.03683
Trace of r
5
20
15000
20000
25000
30000
Iterations
Density of r
0.00
0
5
10
15
20
N = 20000 Bandwidth = 0.6215

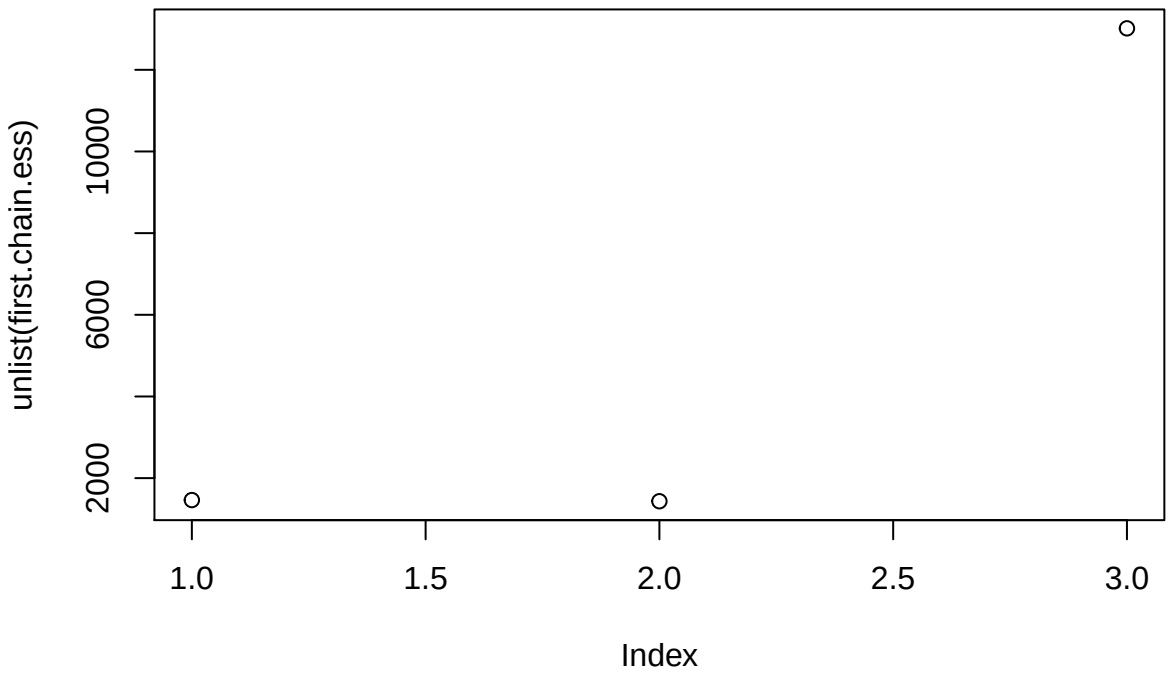

Gelman Diagnostic
Upper C.I.
1.0001
1.0003
1.0005
1.0000
1.0001
1.0002
1.0003
1.0004
Point est.



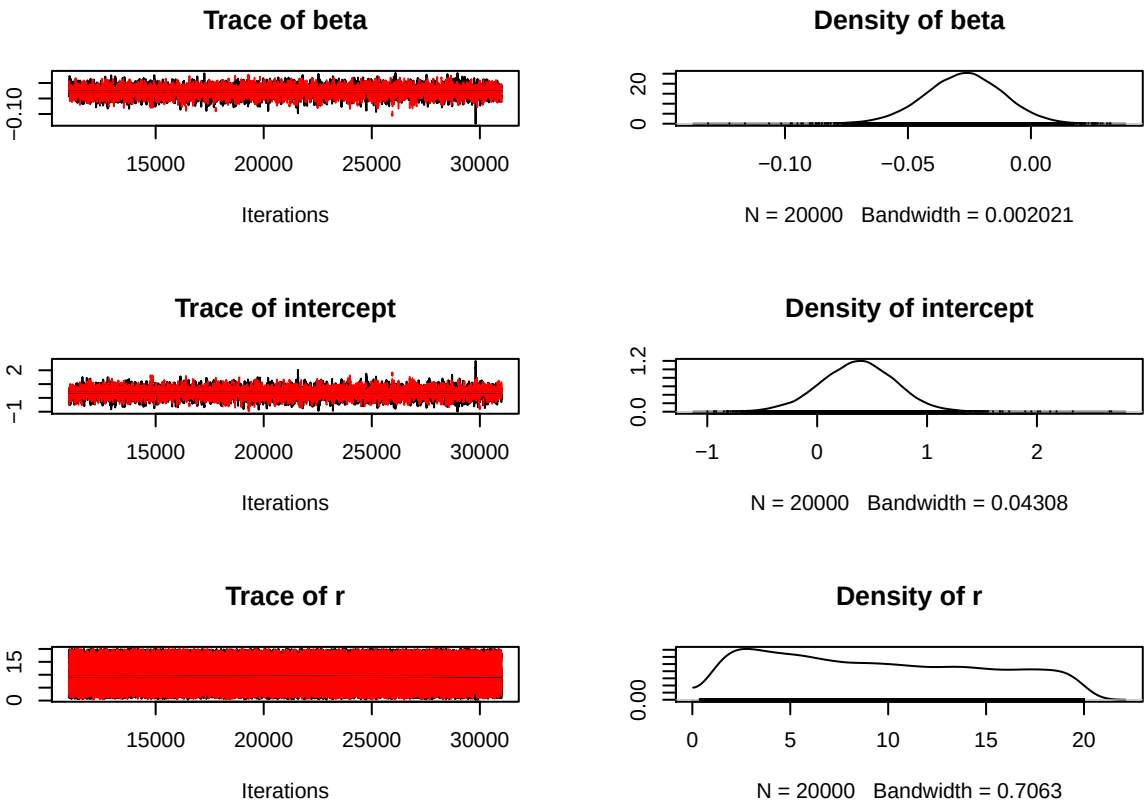

```
## [1] "JAGS GLM Models for Negative Binomial  3"
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 41
##    Unobserved stochastic nodes: 3
##    Total graph size: 297
##
## Initializing model
```

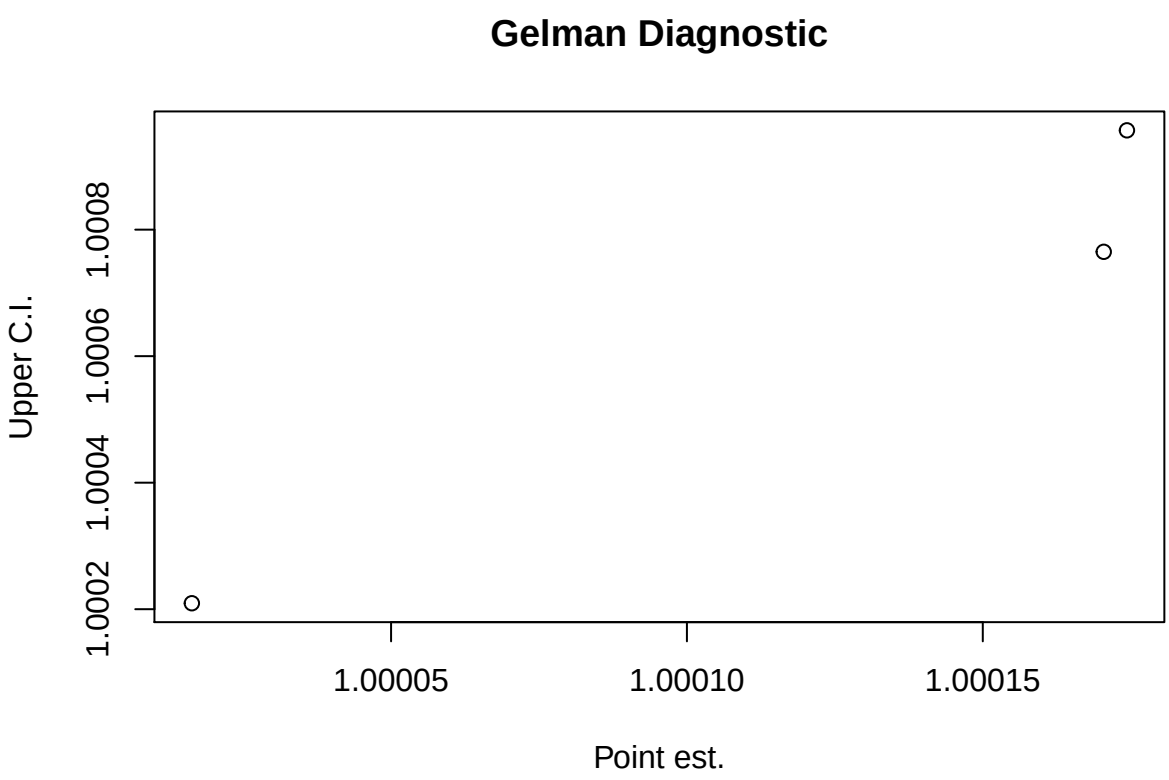

Trace of beta
−0.06
15000
20000
25000
30000
Iterations
Density of beta
0 20
−0.05
0.00
0.05
N = 20000 Bandwidth = 0.001798
Trace of intercept
−2.0
15000
20000
25000
30000
Iterations
Density of intercept
0.0
−2
−1
0
1
N = 20000 Bandwidth = 0.04554
Trace of r
0 15
15000
20000
25000
30000
Iterations
Density of r
0.00
0
5
10
15
20
N = 20000 Bandwidth = 0.676

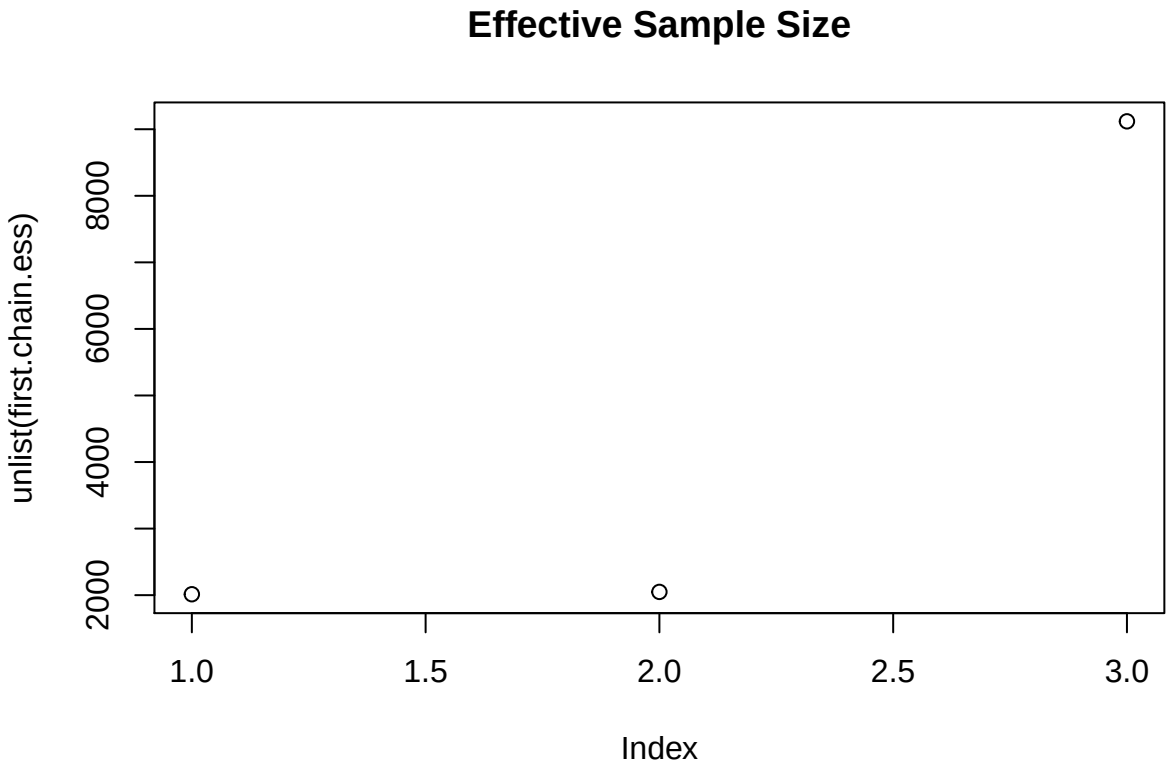

Gelman Diagnostic
Upper C.I.
1.000
1.001
1.002
1.003
1.004
1.0000
1.0002
1.0004
1.0006
1.0008
1.0010
Point est.



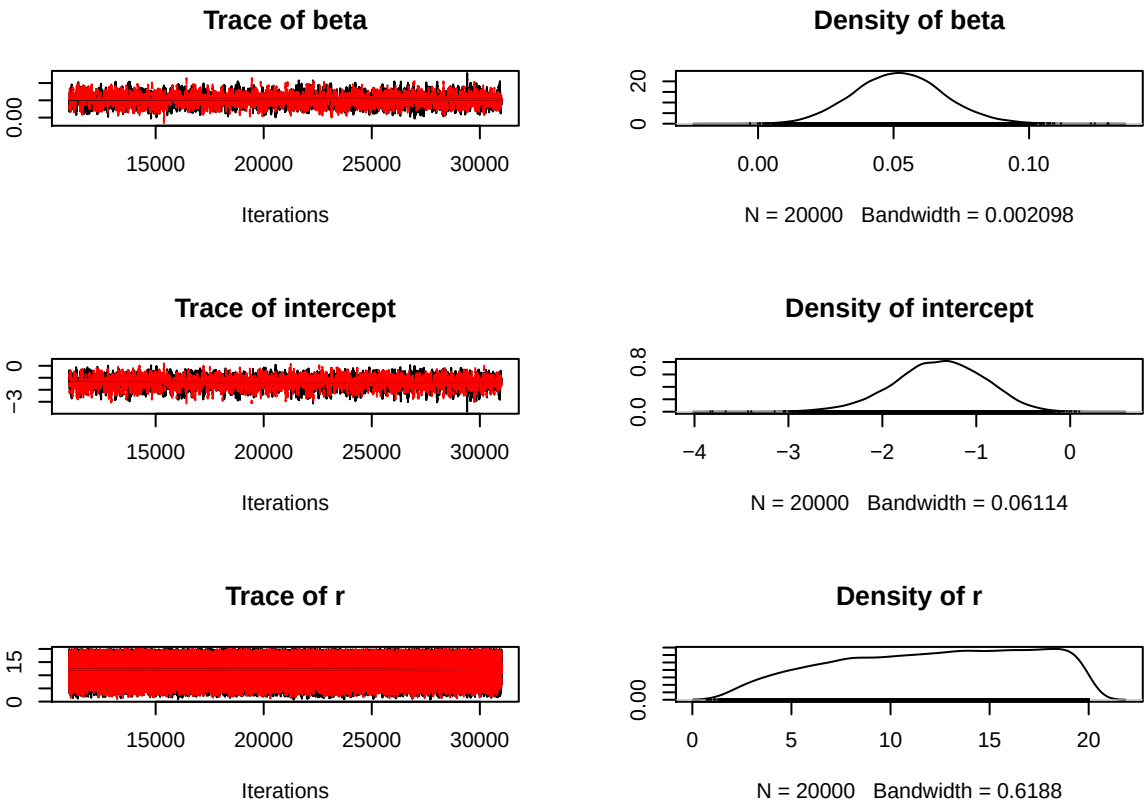

```
## [1] "JAGS GLM Models for Negative Binomial  4"
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 41
##    Unobserved stochastic nodes: 3
##    Total graph size: 297
##
## Initializing model
```

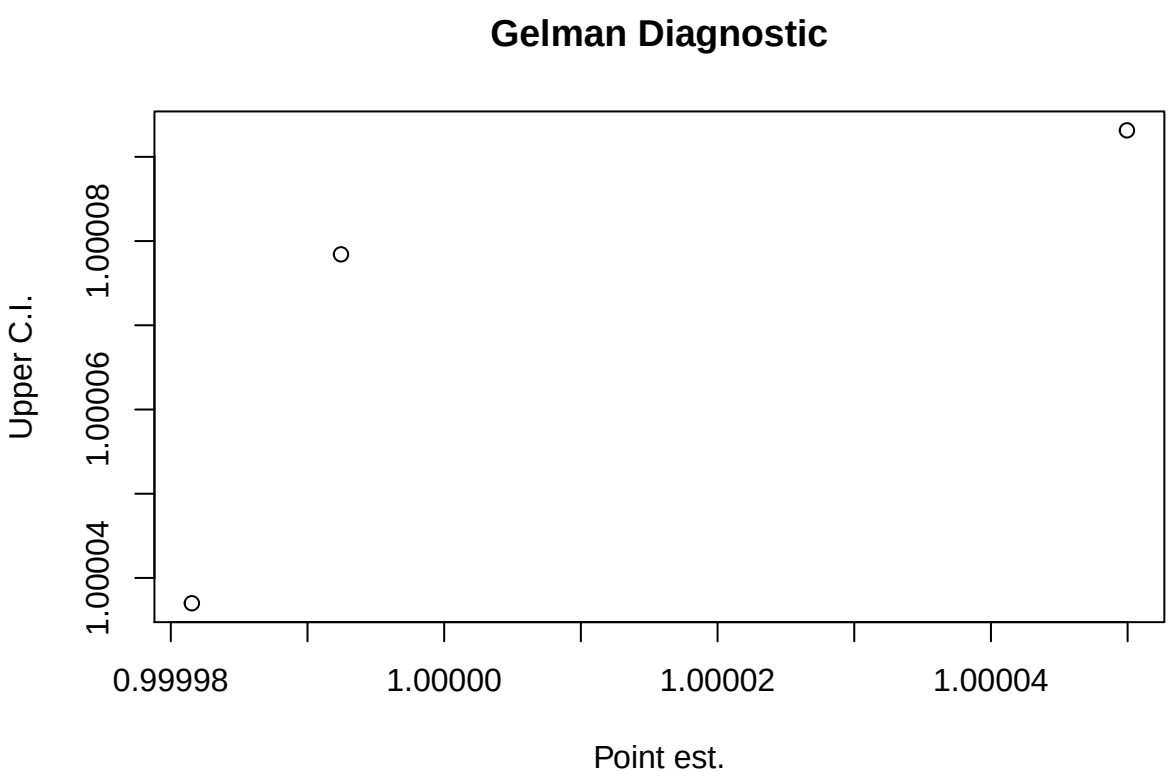

Trace of beta
−0.10
15000
20000
25000
30000
Iterations
Density of beta
0
20
−0.10
−0.05
0.00
N = 20000 Bandwidth = 0.002021
Trace of intercept
−1
2
15000
20000
25000
30000
Iterations
Density of intercept
0.0
1.2
−1
0
1
2
N = 20000 Bandwidth = 0.04308
Trace of r
0
15
15000
20000
25000
30000
Iterations
Density of r
0.00
0
5
10
15
20
N = 20000 Bandwidth = 0.7063

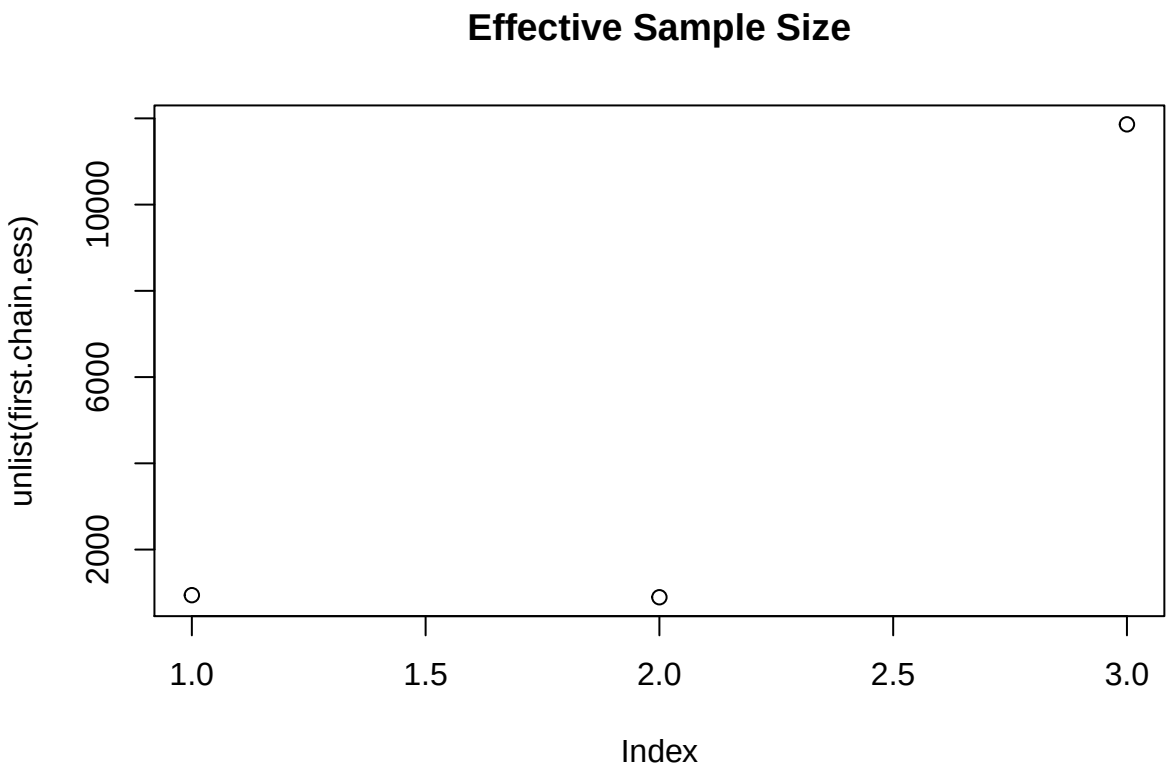

Gelman Diagnostic
Upper C.I.
1.0002
1.0004
1.0006
1.0008
1.00005
1.00010
1.00015
Point est.




```
## [1] "JAGS GLM Models for Negative Binomial  5"
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 41
##    Unobserved stochastic nodes: 3
##    Total graph size: 297
##
## Initializing model
```


Trace of beta
0.00
15000
20000
25000
30000
Iterations
Density of beta
0
20
0.00
0.05
0.10
N = 20000 Bandwidth = 0.002098
Trace of intercept
−3
0
15000
20000
25000
30000
Iterations
Density of intercept
0.0
0.8
−4
−3
−2
−1
0
N = 20000 Bandwidth = 0.06114
Trace of r
0
15
15000
20000
25000
30000
Iterations
Density of r
0.00
0
5
10
15
20
N = 20000 Bandwidth = 0.6188


Gelman Diagnostic
Upper C.I.
1.00004
1.00006
1.00008
0.99998
1.00000
1.00002
1.00004
Point est.



Fit JAGS Poisson Random Effects

```
######################################## Fit JAGS Poisson Random Effects
library(rjags)
library(coda)
model_code = '
model
{
    ## Likelihood
    for(i in 1:N){
      for(j in 1:9){
        Y[i,j] ~ dpois(lambda[i,j])
        log(lambda[i,j]) <- mu[i,j]
        mu[i,j] <- intercept + beta*t[i] + alpha[j]
      }
    }

  ## Priors
  # Random effects
  for(i in 1:9){
    alpha[i] ~ dnorm(0,taus)
  }
  taus ~ dgamma(0.1,0.1)
  beta ~ dnorm(mu.beta,tau.beta)
```

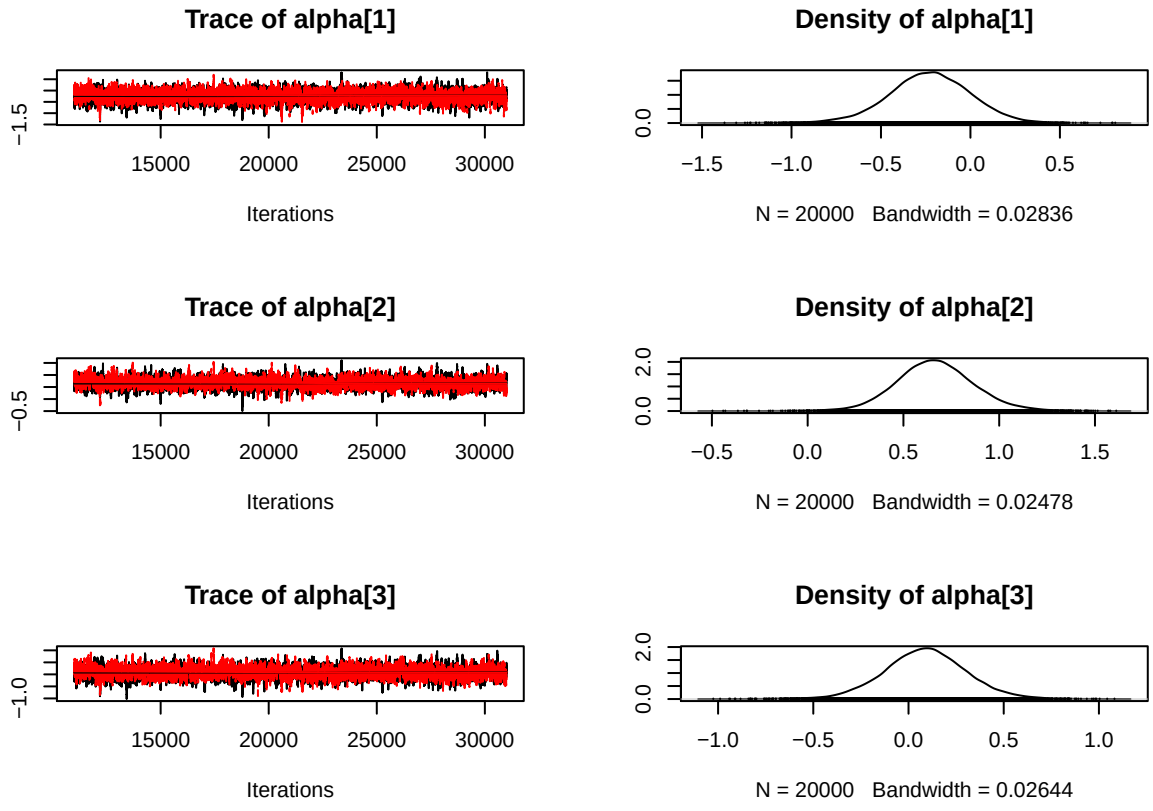
```
  intercept ~ dnorm(mu.intercept,tau.intercept)
}
'
  # Set up the data
  model_data = list(N = 41, t=seq(1:41),Y=X.num,mu.beta=0,tau.beta=.0001,mu.intercept=0,tau.intercept=.
  # Choose the parameters to watch
  model_parameters =  c("r","beta", "intercept", "alpha")
  n.chains =2;nSamples = 10000
  model <- jags.model(textConnection(model_code),data = model_data,n.chains = n.chains)#Compile Model G
```

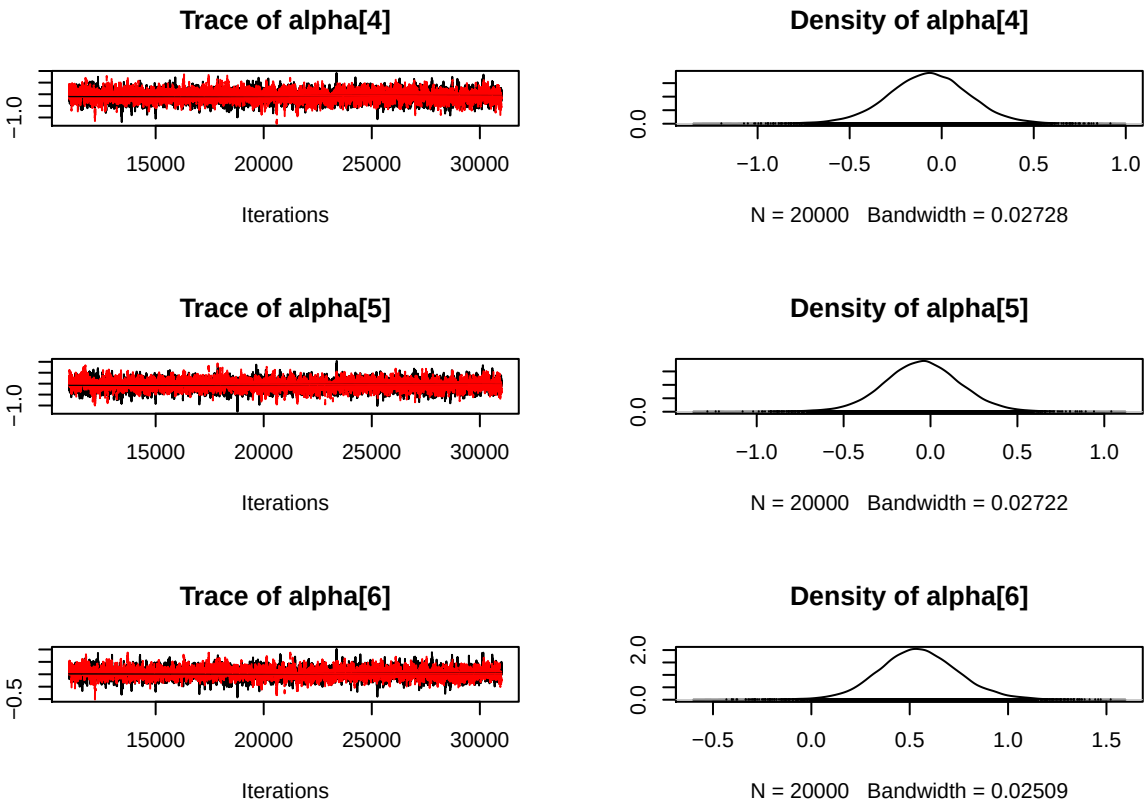
```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 369
##    Unobserved stochastic nodes: 12
##    Total graph size: 1208
##
## Initializing model
```

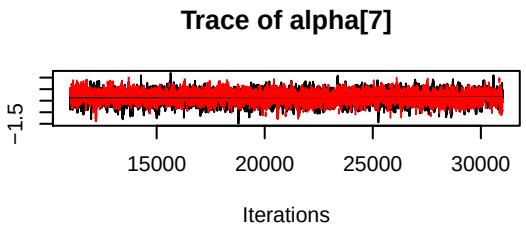
```
  update(model, nSamples, progress.bar="none"); # Burnin
  out.coda  <- coda.samples(model, variable.names=model_parameters,n.iter=2*nSamples)
```

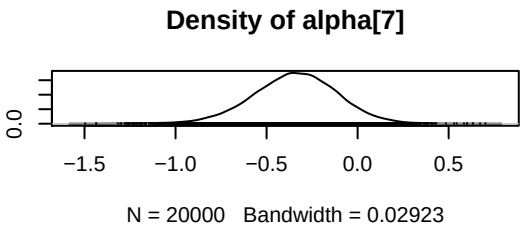
```
## Warning in jags.samples(model, variable.names, n.iter, thin, type = "trace", : Failed to set trace m
## Variable r not found
```

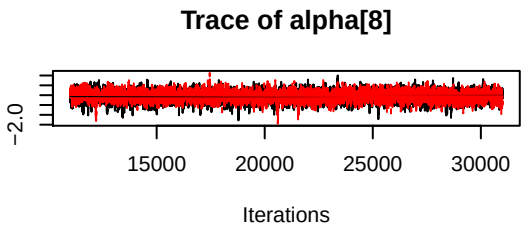
```
  plot(out.coda)
```

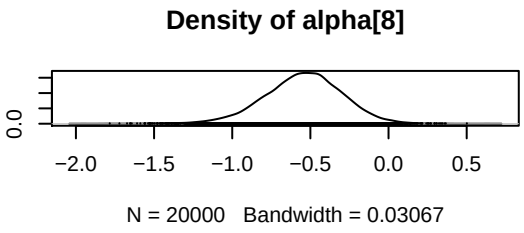


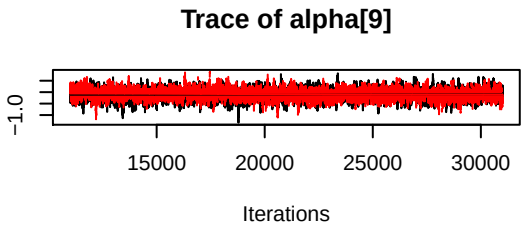

Trace of alpha[4]
−1.0
15000
20000
25000
30000
Iterations
Density of alpha[4]
0.0
−1.0
−0.5
0.0
0.5
1.0
N = 20000 Bandwidth = 0.02728
Trace of alpha[5]
−1.0
15000
20000
25000
30000
Iterations
Density of alpha[5]
0.0
−1.0
−0.5
0.0
0.5
1.0
N = 20000 Bandwidth = 0.02722
Trace of alpha[6]
−0.5
15000
20000
25000
30000
Iterations
Density of alpha[6]
0.0
2.0
−0.5
0.0
0.5
1.0
1.5
N = 20000 Bandwidth = 0.02509

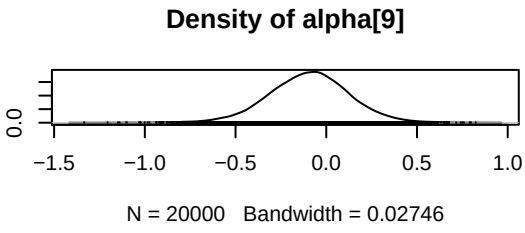

Trace of alpha[7]
−1.5
15000
20000
25000
30000
Iterations

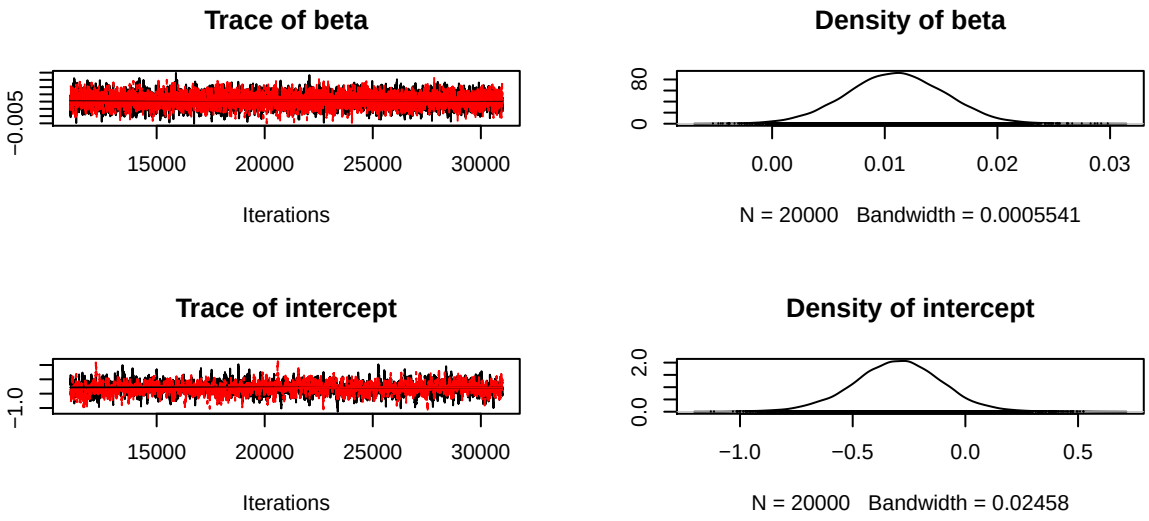

Density of alpha[7]
0.0
−1.5
−1.0
−0.5
0.0
0.5
N = 20000 Bandwidth = 0.02923


Trace of alpha[8]
−2.0
15000
20000
25000
30000
Iterations


Density of alpha[8]
0.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
N = 20000 Bandwidth = 0.03067


Trace of alpha[9]
−1.0
15000
20000
25000
30000
Iterations


Density of alpha[9]
0.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
N = 20000 Bandwidth = 0.02746




```
  #assess the posteriors??? stationarity, by looking at the Heidelberg-Welch convergence diagnostic:
  heidel.diag(out.coda)
```

```
## [[1]]
##
##           Stationarity start     p-value
##           test         iteration
## alpha[1]  passed       1         0.105
## alpha[2]  passed       1         0.236
## alpha[3]  passed       1         0.282
## alpha[4]  passed       1         0.274
## alpha[5]  passed       1         0.593
## alpha[6]  passed       1         0.329
## alpha[7]  passed       1         0.431
## alpha[8]  passed       1         0.179
## alpha[9]  passed       1         0.209
## beta      passed       1         0.861
## intercept passed       1         0.476
##
##           Halfwidth Mean    Halfwidth
##           test
## alpha[1]  passed    -0.2376 0.014224
## alpha[2]  passed     0.6583 0.014674
## alpha[3]  failed     0.0798 0.014422
## alpha[4]  failed    -0.0792 0.013646
## alpha[5]  failed    -0.0559 0.014255
```

```
## alpha[6]  passed     0.5376 0.014355
## alpha[7]  passed    -0.3520 0.013533
## alpha[8]  passed    -0.5563 0.013940
## alpha[9]  failed    -0.1020 0.013191
## beta      passed     0.0109 0.000205
## intercept passed    -0.2911 0.017849
##
## [[2]]
##
##           Stationarity start     p-value
##           test         iteration
## alpha[1]  passed       6001      0.0530
## alpha[2]  passed       6001      0.0649
## alpha[3]  passed       6001      0.0673
## alpha[4]  passed       4001      0.0871
## alpha[5]  passed       6001      0.1634
## alpha[6]  passed       6001      0.0510
## alpha[7]  passed       8001      0.5321
## alpha[8]  passed       6001      0.1846
## alpha[9]  passed       6001      0.0973
## beta      passed          1      0.1801
## intercept passed       2001      0.0529
##
##           Halfwidth Mean    Halfwidth
##           test
## alpha[1]  passed    -0.2288 0.016439
## alpha[2]  passed     0.6651 0.017896
## alpha[3]  failed     0.0874 0.017894
## alpha[4]  failed    -0.0700 0.015631
## alpha[5]  failed    -0.0512 0.017550
## alpha[6]  passed     0.5460 0.016334
## alpha[7]  passed    -0.3566 0.017698
## alpha[8]  passed    -0.5477 0.016362
## alpha[9]  failed    -0.0972 0.017726
## beta      passed     0.0110 0.000209
## intercept passed    -0.3055 0.018346
```

```
  # check that our chain???s length is satisfactory.
  raftery.diag(out.coda)
```

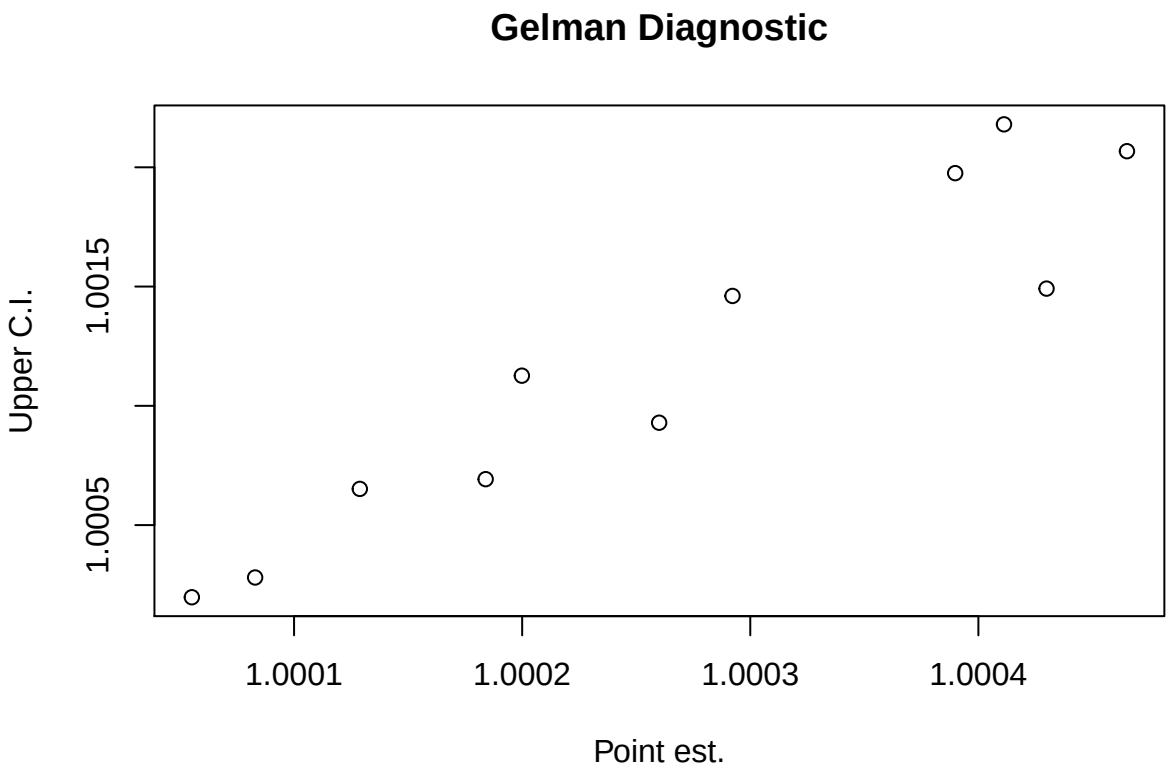
```
## [[1]]
##
## Quantile (q) = 0.025
## Accuracy (r) = +/- 0.005
## Probability (s) = 0.95
##
##            Burn-in  Total Lower bound  Dependence
##            (M)      (N)   (Nmin)       factor (I)
##  alpha[1]  30       33978 3746          9.07
##  alpha[2]  30       34140 3746          9.11
##  alpha[3]  28       29992 3746          8.01
##  alpha[4]  25       31705 3746          8.46
##  alpha[5]  25       28360 3746          7.57
##  alpha[6]  49       50379 3746         13.40
##  alpha[7]  30       39582 3746         10.60
```

```
##  alpha[8]  25       27400 3746          7.31
##  alpha[9]  20       26525 3746          7.08
##  beta      18       19641 3746          5.24
##  intercept 54       54870 3746         14.60
##
##
## [[2]]
##
## Quantile (q) = 0.025
## Accuracy (r) = +/- 0.005
## Probability (s) = 0.95
##
##            Burn-in  Total Lower bound  Dependence
##            (M)      (N)   (Nmin)       factor (I)
##  alpha[1]  28       34818 3746          9.29
##  alpha[2]  48       52452 3746         14.00
##  alpha[3]  42       46536 3746         12.40
##  alpha[4]  30       32850 3746          8.77
##  alpha[5]  28       35917 3746          9.59
##  alpha[6]  35       40572 3746         10.80
##  alpha[7]  32       41912 3746         11.20
##  alpha[8]  40       55904 3746         14.90
##  alpha[9]  42       50463 3746         13.50
##  beta      18       18009 3746          4.81
##  intercept 50       51735 3746         13.80
```

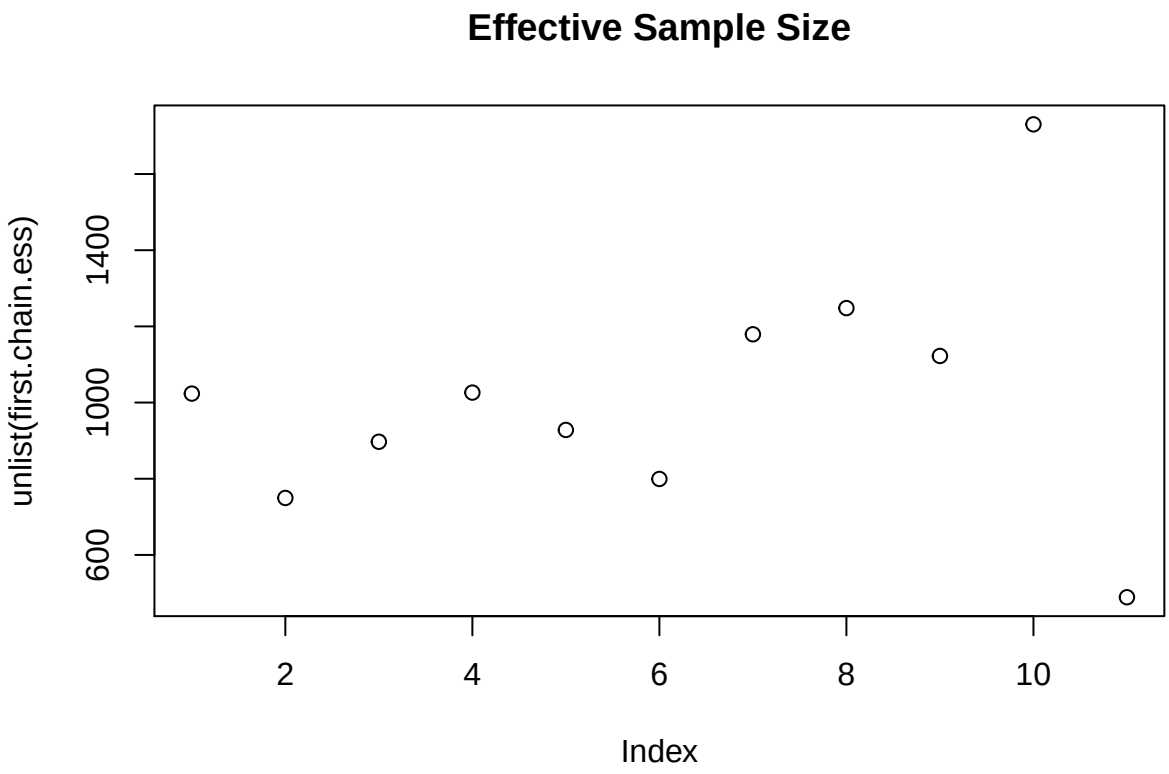
```
  geweke.diag(out.coda)
```

```
## [[1]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##  alpha[1]  alpha[2]  alpha[3]  alpha[4]  alpha[5]  alpha[6]  alpha[7]
##  -0.18363  -0.32794   0.06228   0.01257   0.29577   0.05159   0.30095
##  alpha[8]  alpha[9]      beta intercept
##   0.18407  -0.22364   0.96630  -0.25604
##
##
## [[2]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##  alpha[1]  alpha[2]  alpha[3]  alpha[4]  alpha[5]  alpha[6]  alpha[7]
##     1.973     2.086     1.916     1.668     2.071     1.705     2.030
##  alpha[8]  alpha[9]      beta intercept
##     1.730     1.619     1.588    -1.899
```

```
  if(n.chains > 1)
  {
    gelman.srf <-gelman.diag(out.coda)
    plot(gelman.srf$psrf,main = "Gelman Diagnostic")
  }
```




```
chains.ess <- lapply(out.coda,effectiveSize)
first.chain.ess <- chains.ess[1]
plot(unlist(first.chain.ess), main="Effective Sample Size")
```



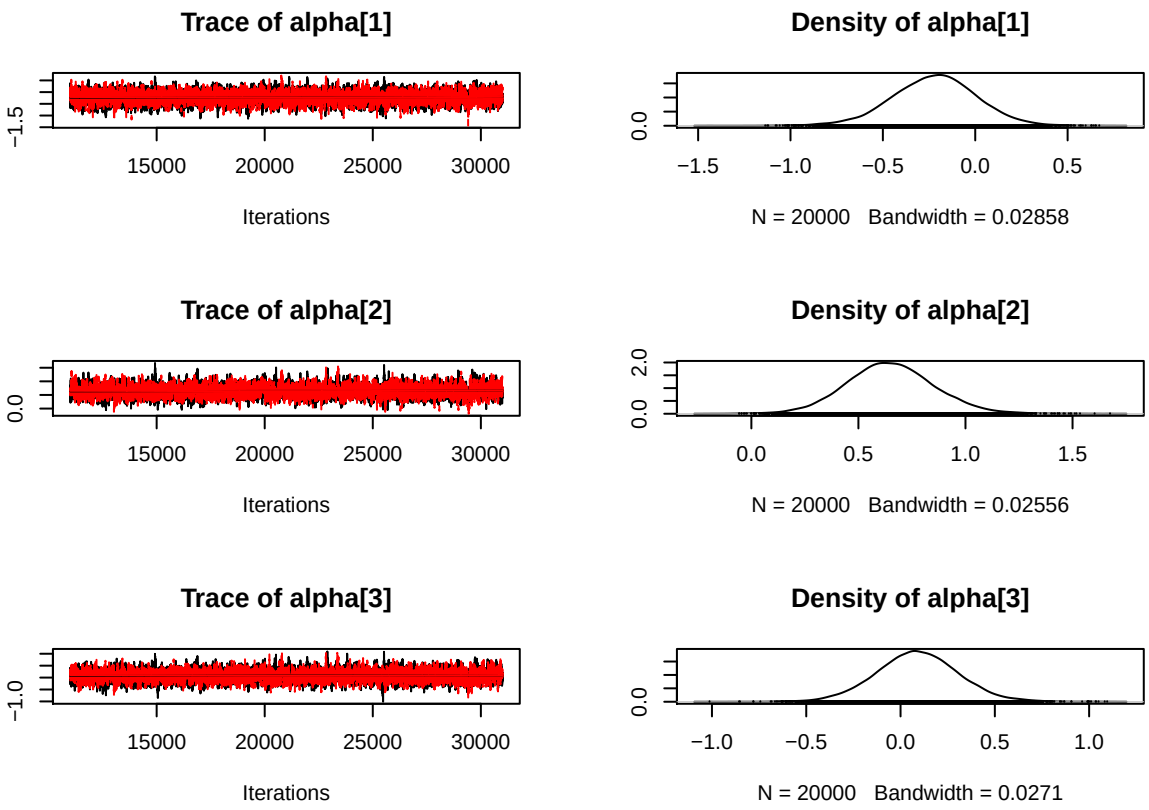
Fit JAGS Negative Binomial Random Effects

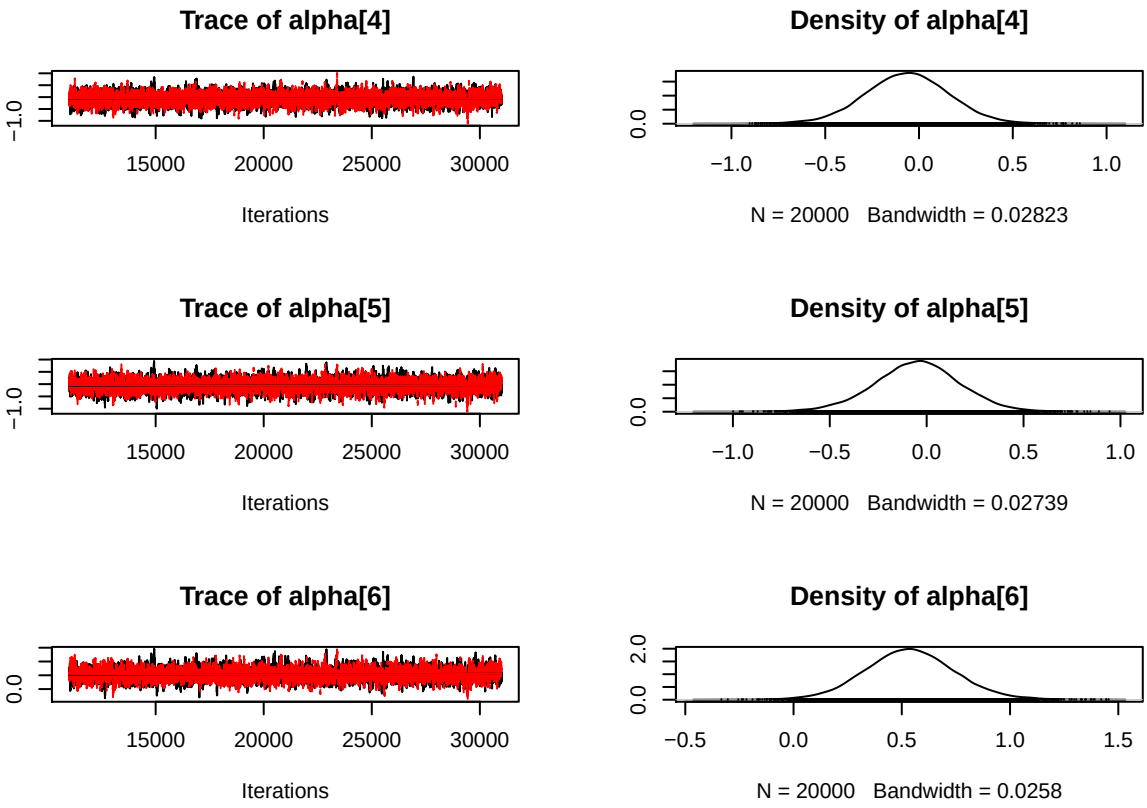
```
######################################## Fit JAGS Negative Binomial Random Effects
library(rjags)
library(coda)
model_code = '
model
{
    ## Likelihood
    for(i in 1:N){
      for(j in 1:9){
        Y[i,j] ~ dnegbin(p[i,j],r)
        p[i,j] <- r/(r+lambda[i,j])
        log(lambda[i,j]) <- mu[i,j]
        mu[i,j] <- intercept + beta*t[i] + alpha[j]
      }
    }

  ## Priors
  # Random effects
  for(i in 1:9){
    alpha[i] ~ dnorm(0,taus)
  }
  taus ~ dgamma(0.1,0.1)
```

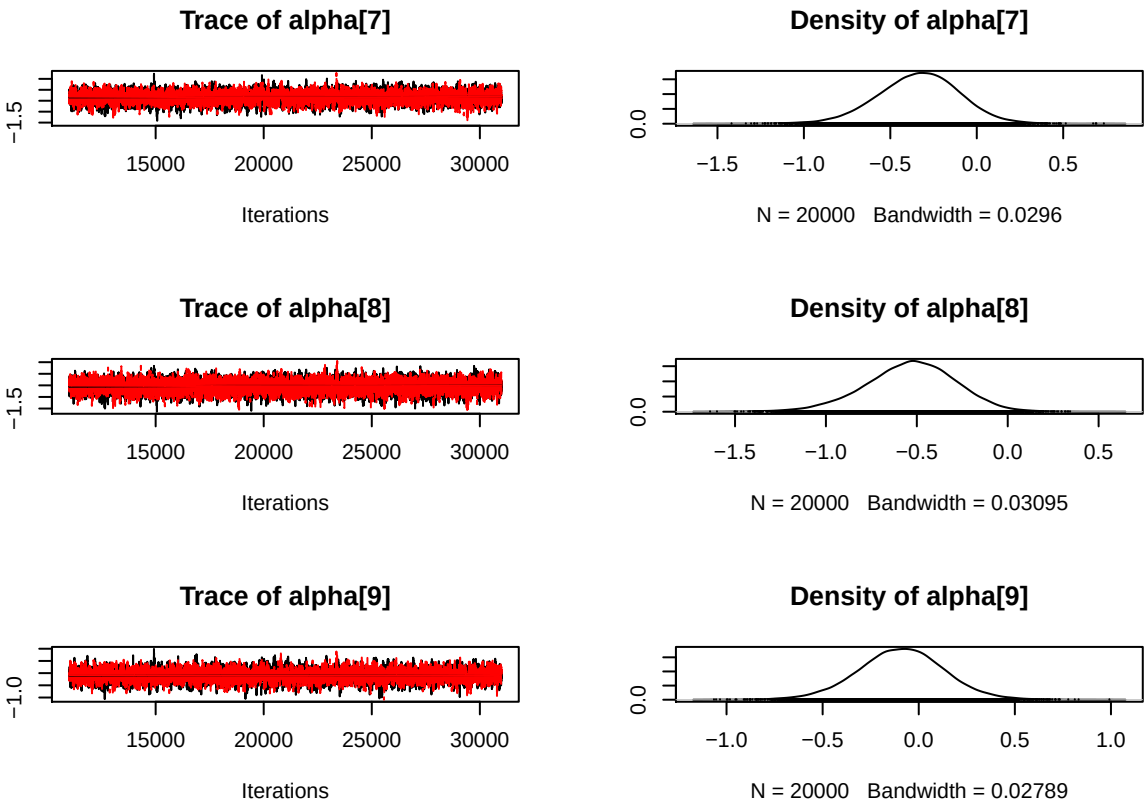
```
  r ~ dunif(0,20)
  beta ~ dnorm(mu.beta,tau.beta)
  intercept ~ dnorm(mu.intercept,tau.intercept)
}
'
  # Set up the data
  model_data = list(N = 41, t=seq(1:41),Y=X.num,mu.beta=0,tau.beta=.0001,mu.intercept=0,tau.intercept=.
  # Choose the parameters to watch
  model_parameters =  c("r","beta", "intercept", "alpha")
  n.chains =2;nSamples = 10000
  model <- jags.model(textConnection(model_code),data = model_data,n.chains = n.chains)#Compile Model G
```

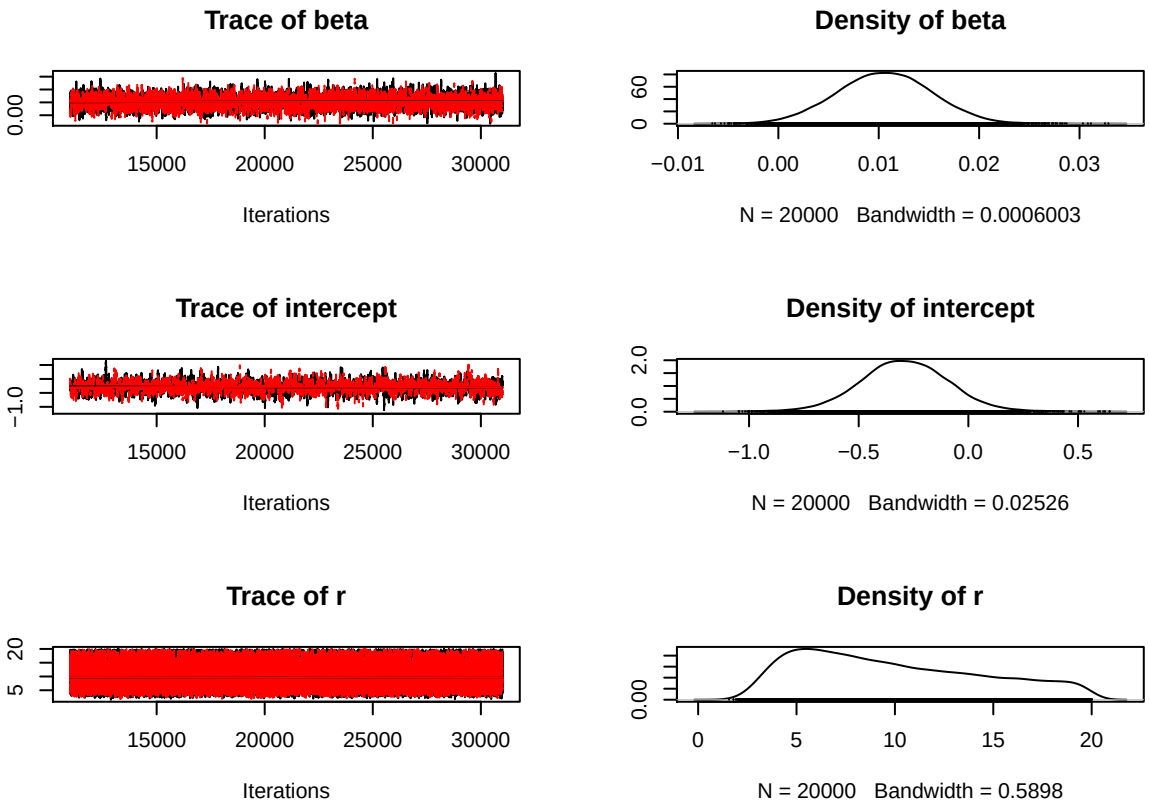
```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 369
##    Unobserved stochastic nodes: 13
##    Total graph size: 1948
##
## Initializing model
```

```
  update(model, nSamples, progress.bar="none"); # Burnin
  out.coda  <- coda.samples(model, variable.names=model_parameters,n.iter=2*nSamples)
  plot(out.coda)
```




Trace of alpha[4]
−1.0
15000
20000
25000
30000
Iterations
Density of alpha[4]
0.0
−1.0
−0.5
0.0
0.5
1.0
N = 20000 Bandwidth = 0.02823
Trace of alpha[5]
−1.0
15000
20000
25000
30000
Iterations
Density of alpha[5]
0.0
−1.0
−0.5
0.0
0.5
1.0
N = 20000 Bandwidth = 0.02739
Trace of alpha[6]
0.0
15000
20000
25000
30000
Iterations
Density of alpha[6]
0.0
2.0
−0.5
0.0
0.5
1.0
1.5
N = 20000 Bandwidth = 0.0258


Trace of alpha[7]
−1.5
15000
20000
25000
30000
Iterations
Density of alpha[7]
0.0
−1.5
−1.0
−0.5
0.0
0.5
N = 20000 Bandwidth = 0.0296
Trace of alpha[8]
−1.5
15000
20000
25000
30000
Iterations
Density of alpha[8]
0.0
−1.5
−1.0
−0.5
0.0
0.5
N = 20000 Bandwidth = 0.03095
Trace of alpha[9]
−1.0
15000
20000
25000
30000
Iterations
Density of alpha[9]
0.0
−1.0
−0.5
0.0
0.5
1.0
N = 20000 Bandwidth = 0.02789




```
  #assess the posteriors??? stationarity, by looking at the Heidelberg-Welch convergence diagnostic:
  heidel.diag(out.coda)
```

```
## [[1]]
##
##           Stationarity start     p-value
##           test         iteration
## alpha[1]  passed          1      0.1324
## alpha[2]  passed          1      0.4053
## alpha[3]  passed          1      0.2897
## alpha[4]  passed          1      0.4878
## alpha[5]  passed          1      0.3330
## alpha[6]  passed          1      0.1299
## alpha[7]  passed          1      0.2904
## alpha[8]  passed          1      0.1111
## alpha[9]  passed          1      0.3589
## beta      passed       2001      0.0651
## intercept passed          1      0.1132
## r         passed          1      0.6124
##
##           Halfwidth Mean    Halfwidth
##           test
## alpha[1]  passed    -0.2220 0.011530
## alpha[2]  passed     0.6500 0.012891
## alpha[3]  failed     0.0901 0.012442
## alpha[4]  failed    -0.0637 0.012137
```

```
## alpha[5]  failed    -0.0485 0.012543
## alpha[6]  passed     0.5383 0.012539
## alpha[7]  passed    -0.3355 0.011454
## alpha[8]  passed    -0.5335 0.011363
## alpha[9]  failed    -0.0918 0.012007
## beta      passed     0.0107 0.000228
## intercept passed    -0.2947 0.016526
## r         passed     9.8348 0.092130
##
## [[2]]
##
##           Stationarity start     p-value
##           test         iteration
## alpha[1]  passed       1         0.983
## alpha[2]  passed       1         0.999
## alpha[3]  passed       1         1.000
## alpha[4]  passed       1         0.950
## alpha[5]  passed       1         0.996
## alpha[6]  passed       1         0.998
## alpha[7]  passed       1         0.835
## alpha[8]  passed       1         0.825
## alpha[9]  passed       1         0.988
## beta      passed       1         0.977
## intercept passed       1         0.996
## r         passed       1         0.647
##
##           Halfwidth Mean    Halfwidth
##           test
## alpha[1]  passed    -0.2210 0.013116
## alpha[2]  passed     0.6488 0.013165
## alpha[3]  failed     0.0881 0.013383
## alpha[4]  failed    -0.0648 0.013001
## alpha[5]  failed    -0.0499 0.013399
## alpha[6]  passed     0.5389 0.013779
## alpha[7]  passed    -0.3381 0.012567
## alpha[8]  passed    -0.5368 0.012682
## alpha[9]  failed    -0.0932 0.012821
## beta      passed     0.0105 0.000227
## intercept passed    -0.2881 0.017536
## r         passed     9.8491 0.098147
```

```
  # check that our chain???s length is satisfactory.
  raftery.diag(out.coda)
```

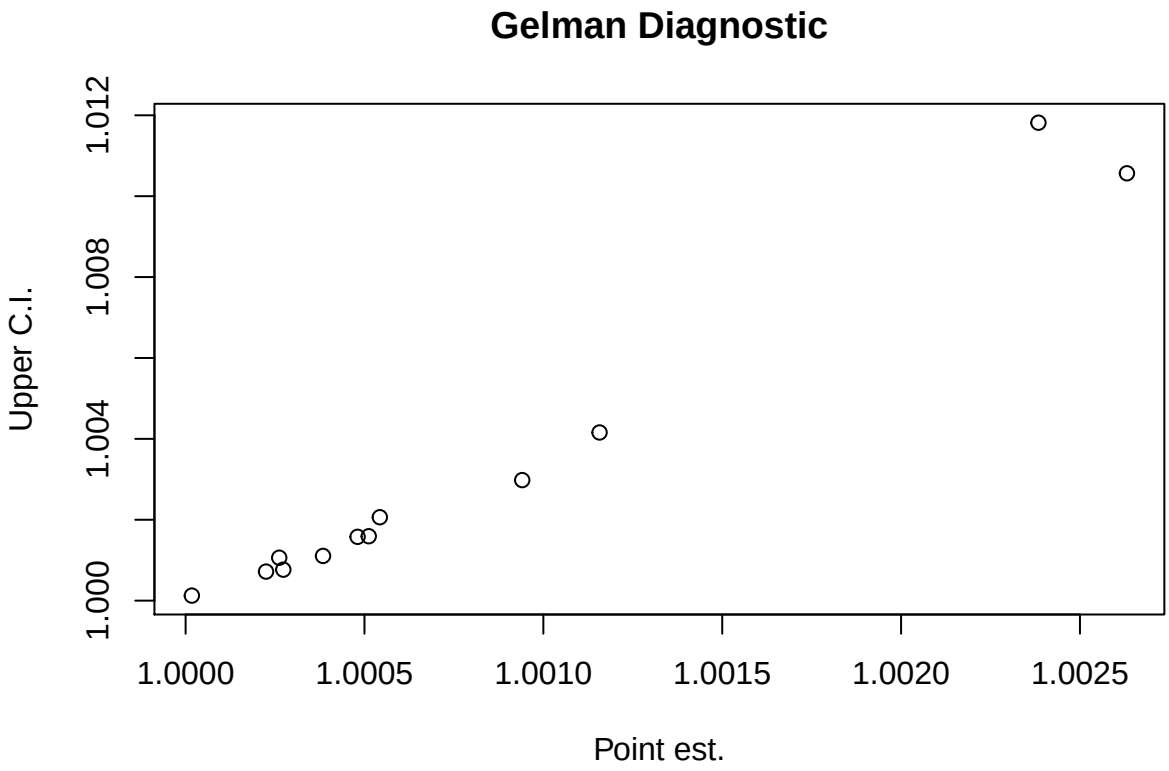
```
## [[1]]
##
## Quantile (q) = 0.025
## Accuracy (r) = +/- 0.005
## Probability (s) = 0.95
##
##            Burn-in  Total Lower bound  Dependence
##            (M)      (N)   (Nmin)       factor (I)
##  alpha[1]  21       27286 3746          7.28
##  alpha[2]  30       32030 3746          8.55
##  alpha[3]  24       26166 3746          6.99
```

```
##  alpha[4]  24       28494 3746          7.61
##  alpha[5]  16       17588 3746          4.70
##  alpha[6]  24       29322 3746          7.83
##  alpha[7]  12       15006 3746          4.01
##  alpha[8]  16       18496 3746          4.94
##  alpha[9]  16       18172 3746          4.85
##  beta      15       18408 3746          4.91
##  intercept 42       40614 3746         10.80
##  r         3        4428  3746          1.18
##
##
## [[2]]
##
## Quantile (q) = 0.025
## Accuracy (r) = +/- 0.005
## Probability (s) = 0.95
##
##            Burn-in  Total Lower bound  Dependence
##            (M)      (N)   (Nmin)       factor (I)
##  alpha[1]  20       23880 3746          6.37
##  alpha[2]  35       40103 3746         10.70
##  alpha[3]  30       29900 3746          7.98
##  alpha[4]  25       25560 3746          6.82
##  alpha[5]  25       33760 3746          9.01
##  alpha[6]  24       26332 3746          7.03
##  alpha[7]  20       24335 3746          6.50
##  alpha[8]  20       23965 3746          6.40
##  alpha[9]  30       40242 3746         10.70
##  beta      15       18486 3746          4.93
##  intercept 42       43972 3746         11.70
##  r         3        4373  3746          1.17
```

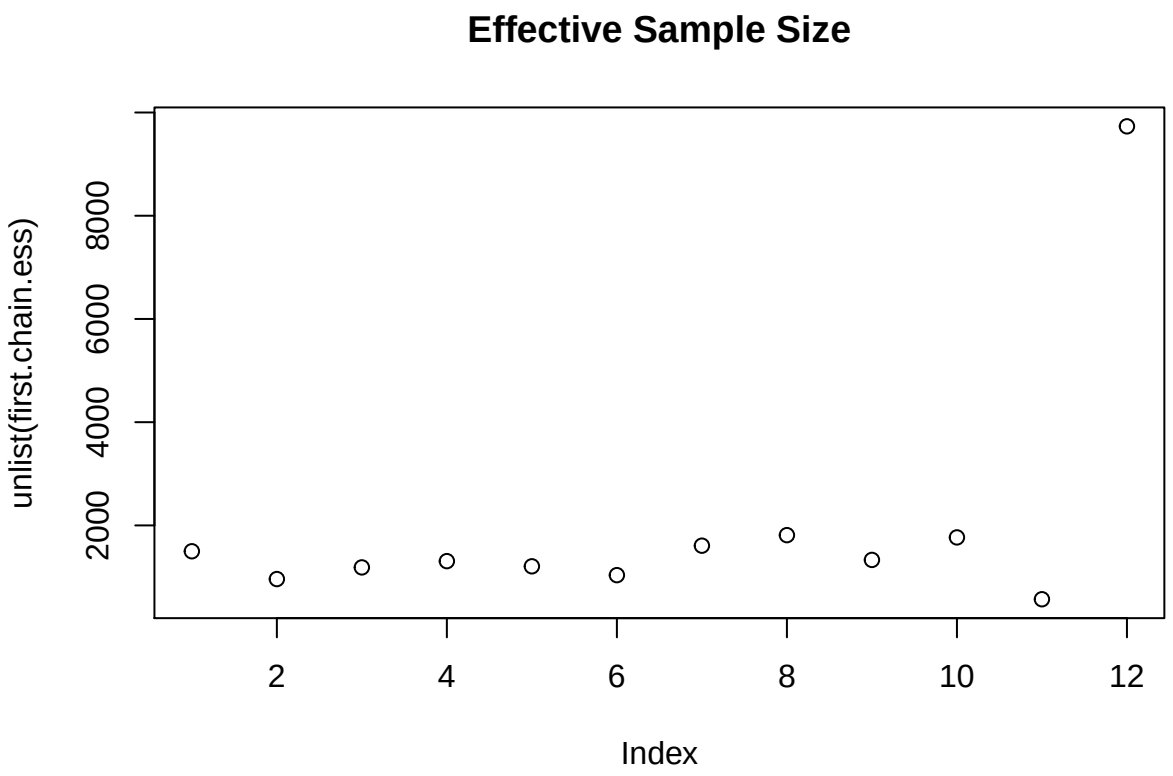
```
geweke.diag(out.coda)
```

```
## [[1]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##  alpha[1]  alpha[2]  alpha[3]  alpha[4]  alpha[5]  alpha[6]  alpha[7]
##   -1.4251   -0.3992   -1.2804   -1.0409   -0.9972   -1.1202   -1.1090
##  alpha[8]  alpha[9]      beta intercept         r
##   -1.9664   -0.4029   -1.4711    1.1859    0.6916
##
##
## [[2]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##  alpha[1]  alpha[2]  alpha[3]  alpha[4]  alpha[5]  alpha[6]  alpha[7]
##    1.6304    1.0406    1.0067    1.2836    1.4355    1.2204    1.8260
##  alpha[8]  alpha[9]      beta intercept         r
##    2.0767    1.1904    0.4412   -1.2131   -0.5063
```

```r
if(n.chains > 1)
{
  gelman.srf <-gelman.diag(out.coda)
  plot(gelman.srf$psrf,main = "Gelman Diagnostic")
}
```




```r
chains.ess <- lapply(out.coda,effectiveSize)
first.chain.ess <- chains.ess[1]
plot(unlist(first.chain.ess), main="Effective Sample Size")
```


Effective Sample Size
unlist(first.chain.ess)
2000
4000
6000
8000
2
4
6
8
10
12
Index