Exercise set 8

Bjørn Christian Weinbach

28th October, 2020

Clear R environment

```r
rm(list = ls())
```

Exercise 8.4

Note: MLE for $\hat{\lambda} = n / \sum_{i=1}^{n} X_i$, where X denote time between failure

Refer to the air-conditioning data set **aircondit** provided in the boot library. The 12 observations are the times in hours between failures of air-conditioning equipment

3 5 7 18 43 85 91 98 100 130 230 487

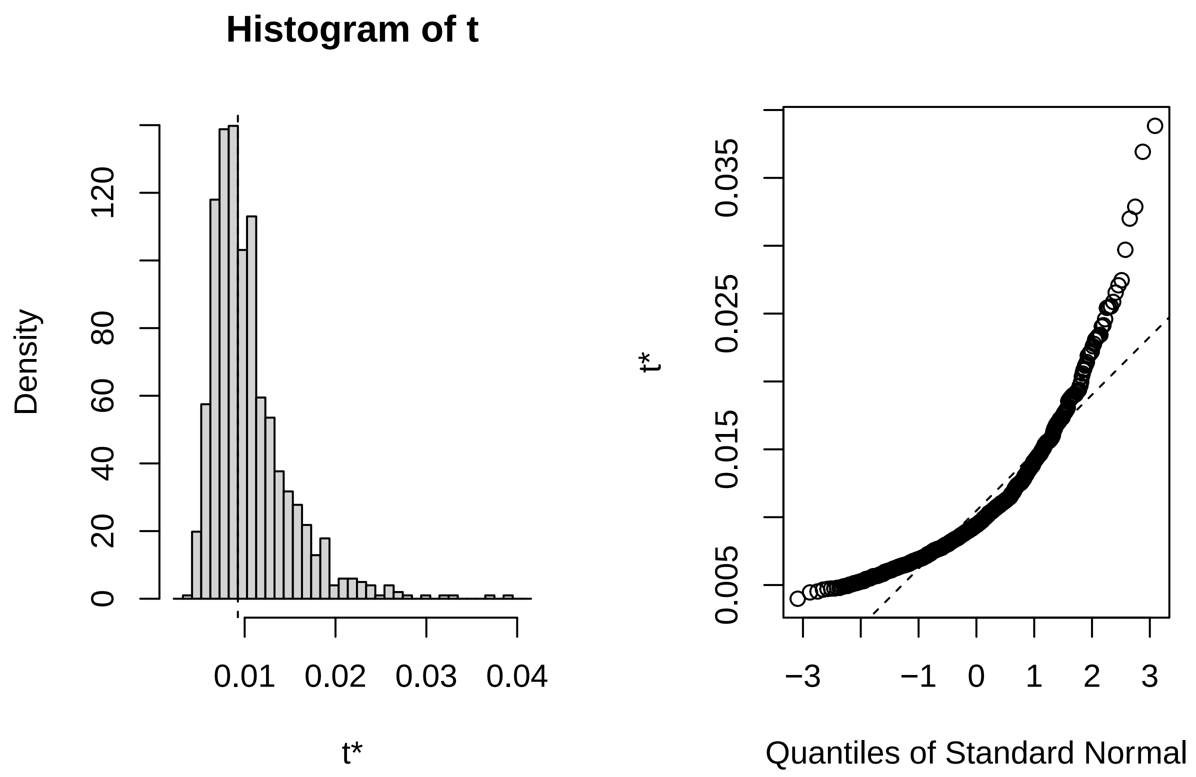
Assume that the times between failures follow an exponential model with rate λ. Obtain the MLE of the hazard rate λ and use bootstrap to estimate the bias and standard error of the estimate.

in R:

```r
library(boot)

# MLE for hazard rate of exponential distributed data
mle <- function(data, i) {
  return(length(data[i])/sum(data[i]))
}

# bootstrapping with 1000 replications
results <- boot(data=aircondit$hours, statistic = mle, R=1000)

# view results
results
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = aircondit$hours, statistic = mle, R = 1000)
##
##
## Bootstrap Statistics :
##       original       bias    std. error
## t1* 0.00925212 0.001228987  0.00427081
```

```r
plot(results)
```



```r
# get 95% confidence interval
boot.ci(results, type=c("bca", "norm", "perc"))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = results, type = c("bca", "norm", "perc"))
##
## Intervals :
## Level      Normal              Percentile            BCa
## 95%   (-0.0003,  0.0164 )   ( 0.0053,  0.0221 )   ( 0.0048,  0.0187 )
## Calculations and Intervals on Original Scale
## Some BCa intervals may be unstable
```

Exercise 8.5

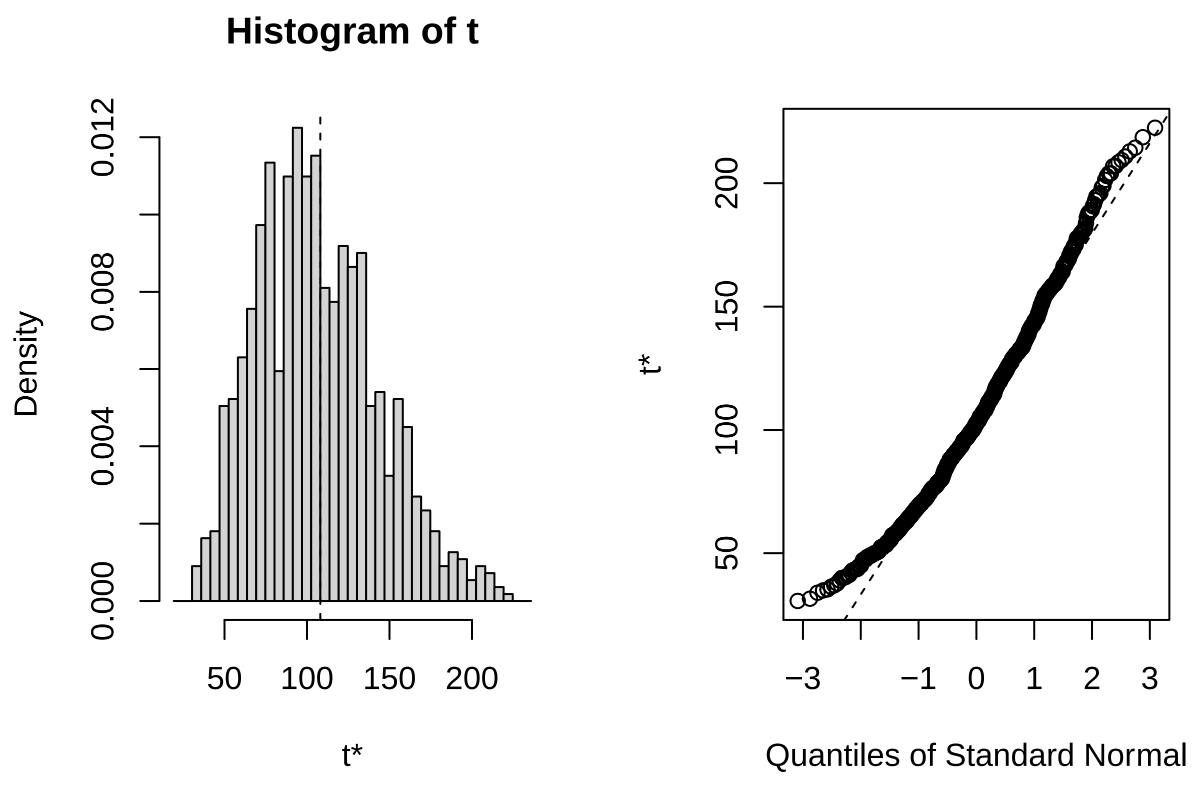
Refer to exercise 8.4. Compute 95% confidence interval for the mean time between failures by the standard normal, basic, percentile and BCa methods.

```r
# MLE for hazard rate of exponential distributed data
meantimeest <- function(data, i) {
  rate <- length(data[i])/sum(data[i])
  return(1/rate)
}
```

```r
# bootstrapping with 1000 replications
results <- boot(data=aircondit$hours, statistic = meantimeest, R=1000)

# view results
results
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = aircondit$hours, statistic = meantimeest, R = 1000)
##
##
## Bootstrap Statistics :
##     original    bias    std. error
## t1* 108.0833 -1.604167    36.58589
```

```r
plot(results)
```



```r
# get 95% confidence interval
boot.ci(results, type=c("norm", "basic", "perc", "bca"))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
```

```
## boot.ci(boot.out = results, type = c("norm", "basic", "perc",
##     "bca"))
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 38.0, 181.4 )   ( 28.0, 168.9 )
##
## Level     Percentile            BCa
## 95%   ( 47.3, 188.2 )   ( 57.9, 215.3 )
## Calculations and Intervals on Original Scale
## Some BCa intervals may be unstable
```

Exercise 11.3

Use metropolis-hastings sampler to generate random variables from a standard Cauchyy distribution. Discard the first 1000 of the chain, and compare the deciles of the generated observations with the deciles of the standard Cauchy distribution. Recall that a Cauchy(θ, η) has density

$$f(x) = \frac{1}{\theta\pi(1 + [(x-\eta)/\theta]^2)}, \quad -\infty < x < \infty, \quad \theta > 0. \tag{1}$$

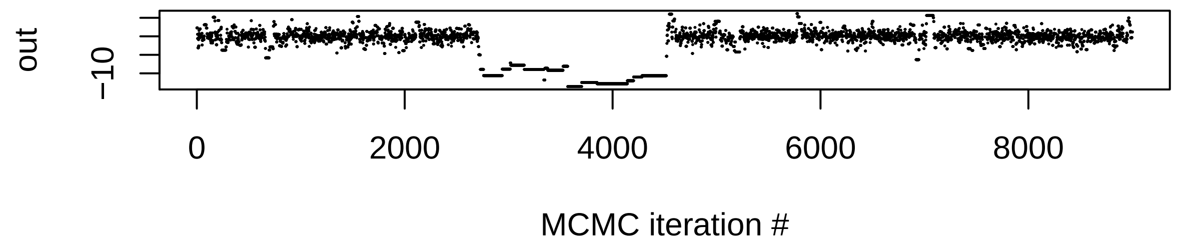
In R:

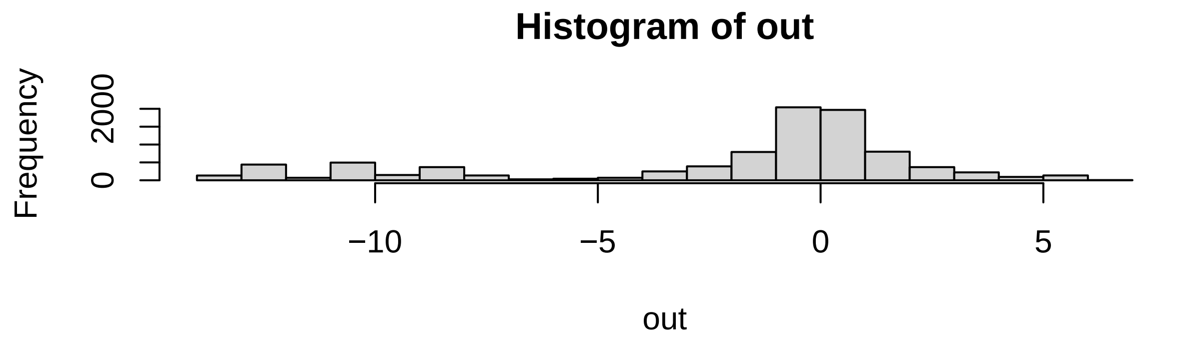
```
# general 1d Gaussian proposal random walk MH
oneD.RWMH <- function(lprob,
                      sigma=1.0,
                      theta1=0.0,
                      n.iter=10000){
  # space for output
  output <- numeric(n.iter)
  # first iterate given
  output[1] <- theta1
  # Calculate probability of theta 1
  lp.old <- lprob(theta1)
  # main iteration loop
  for(t in 2:n.iter){
    # proposal
    thetaStar <- output[t-1] + rnorm(1,sd=sigma)
    # accept probability, for numerical stability we compute
    lp.star <- lprob(thetaStar)
    # the log-accept prob, and then take exp
    alpha <- exp(min(0.0,lp.star-lp.old))
    # accept/reject step
    if(runif(1)<alpha && is.finite(alpha)){
      output[t] <- thetaStar
    } else {
      output[t] <- output[t-1]
    }
  }
print(paste0("RWMH done, accept prob : ",mean(abs(diff(output))>1.0e-14)))
  return(output[1000:length(output)])
}
```

```r
std_cauchy <- function(x){return(dcauchy(x, 0, 1, log=TRUE))}

# try algorithm (change initial condition and sigma for illustration)
out <- oneD.RWMH(std_cauchy, theta1 = 0, sigma=2)
```

```
## [1] "RWMH done, accept prob : 0.28992899289929"
```

```r
par(mfrow=c(2,1))
plot(1:length(out),out,pch=20,cex=0.1,xlab="MCMC iteration #")
hist(out)
```





This implementation uses standard normal proposals. We can see from the plot that the random walk stops suddenly on values far from the center of the distribution and skews the samples. Let's perform some diagnostics:

```r
#
# convergence tests
#
library(coda)
#
ESS <- function(x){ return(as.numeric(coda::effectiveSize(x))) }
ESS(out)
```

```
## [1] 11.07988
```

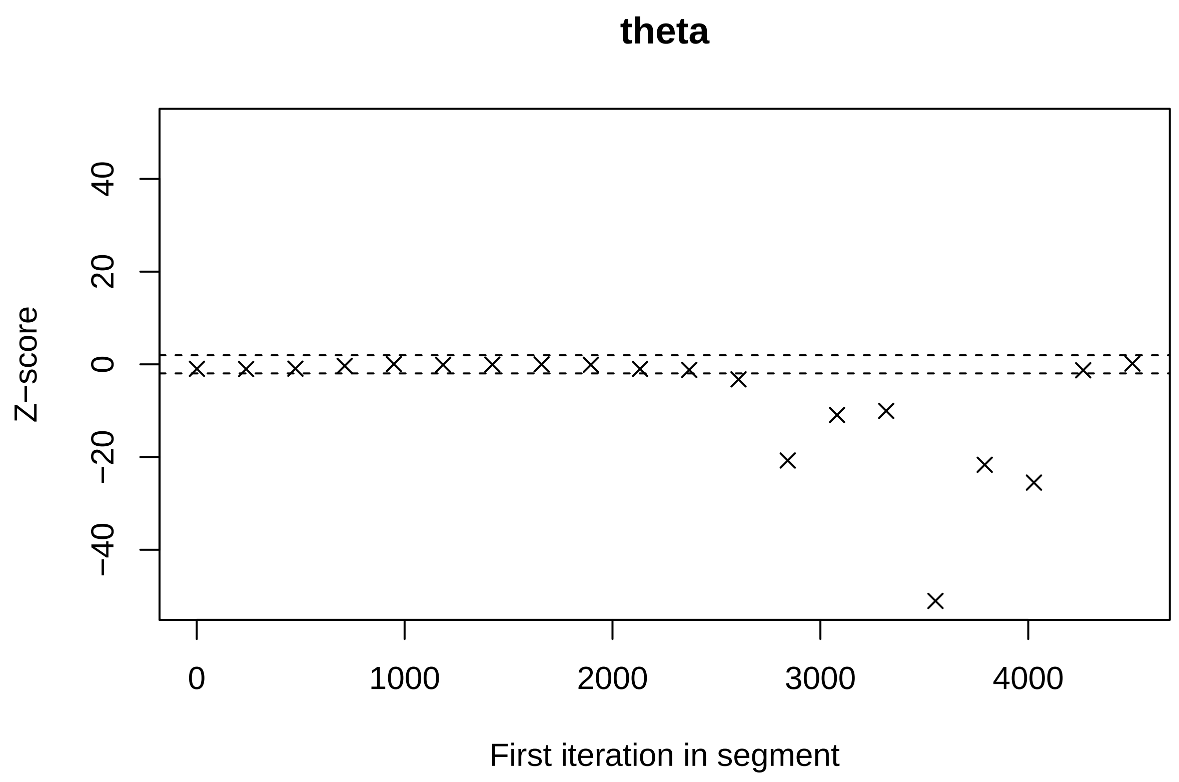
```r
ESS(out)/length(out)
```

```
## [1] 0.001230961
```

```r
# Geweke (single chain test)
#
# make an mcmc object used by the coda package
chain.mcmc <- mcmc(data=out)
varnames(chain.mcmc) <- "theta"
# returns z-score (to be compared two-sided with standard normal)
geweke.diag(chain.mcmc)
```

```
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##   theta
## -0.9605
```

```r
# check if we should discard more burn-in
geweke.plot(chain.mcmc)
```



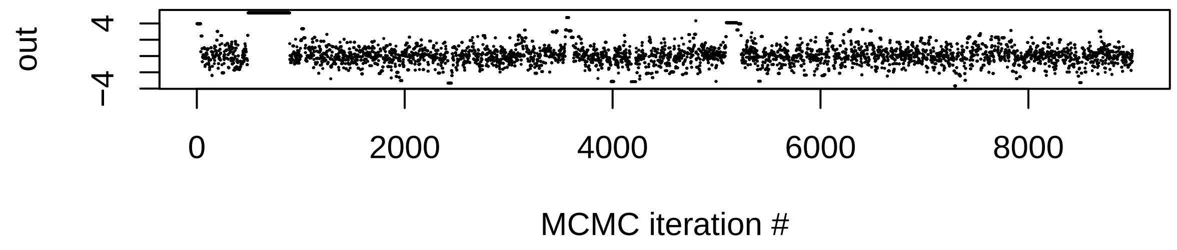
Exercise 11.6

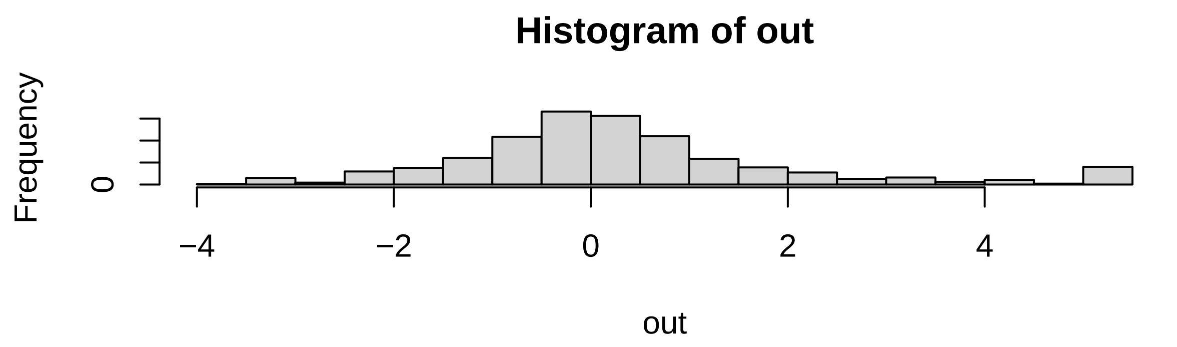
Implement a random walk Metropolis sampler for generating the standard Laplace distribution. For the increment, simulate from a normal distribution. Compare the chains generated when different variances are used for the proposal distribution. Also, compute the acceptance rates of each chain.

```r
# general 1d Gaussian proposal random walk MH
oneD.RWMH <- function(prob,
                      sigma=1.0,
                      theta1=0.0,
                      n.iter=10000){
  # space for output
  output <- numeric(n.iter)
  # first iterate given
  output[1] <- theta1
  # Calculate probability of theta 1
  p.old <- prob(theta1)
  # main iteration loop
  for(t in 2:n.iter){
    # proposal
    thetaStar <- output[t-1] + rnorm(1,sd=sigma)
    # accept probability, for numerical stability we compute
    p.star <- prob(thetaStar)
    # the log-accept prob, and then take exp
    alpha <- min(1,p.star/p.old)
    # accept/reject step
    if(runif(1)<alpha && is.finite(alpha)){
      output[t] <- thetaStar
    } else {
      output[t] <- output[t-1]
    }
  }
print(paste0("RWMH done, accept prob : ",mean(abs(diff(output))>1.0e-14)))
  return(output[1000:length(output)])
}

laplace <- function(x){return(0.5*exp(-abs(x)))}

# try algorithm (change initial condition and sigma for illustration)
out <- oneD.RWMH(laplace, theta1 = 0.0, sigma=1.3)
```

```
## [1] "RWMH done, accept prob : 0.343234323432343"
```

```r
par(mfrow=c(2,1))
plot(1:length(out),out,pch=20,cex=0.1,xlab="MCMC iteration #")
hist(out)
```





Let's perform some diagnostics

```r
ESS <- function(x){ return(as.numeric(coda::effectiveSize(x))) }
ESS(out)
```

```
## [1] 71.47245
```

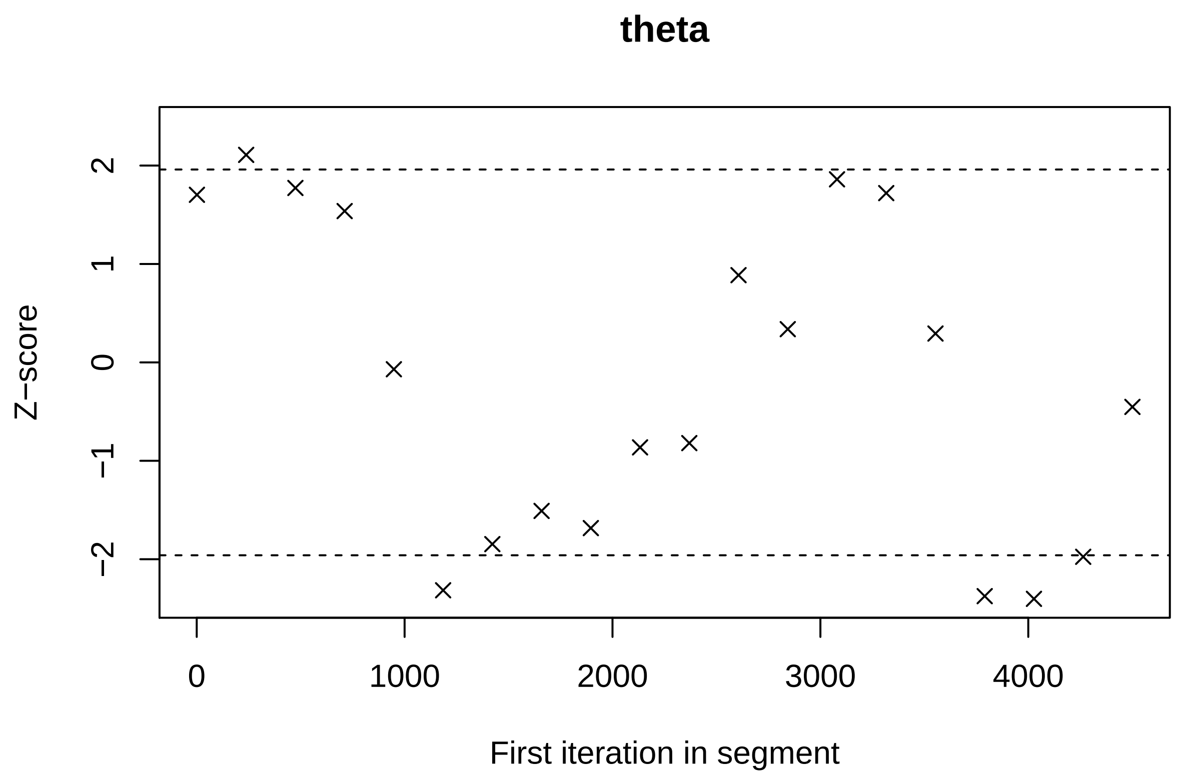
```r
ESS(out)/length(out)
```

```
## [1] 0.007940501
```

```r
#
# convergence tests
#
library(coda)
#
# Geweke (single chain test)
#
# make an mcmc object used by the coda package
chain.mcmc <- mcmc(data=out)
varnames(chain.mcmc) <- "theta"
# returns z-score (to be compared two-sided with standard normal)
geweke.diag(chain.mcmc)
```

```
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
## theta
## 1.703
```

```
# check if we should discard more burn-in
geweke.plot(chain.mcmc)
```



Bibliography