Markov Chain Monte Carlo Methods

Su2016 - Dr. Junvie Pailden

July 27, 2016

Contents

Background Material 1

Introduction to Discrete Markov Chains 3

Markov Chain Monte Carlo (MCMC) 6

Markov Chain Monte Carlo Integration 8

Markov Chain Monte Carlo Methods 8

The Metropolis Sampler 14

The Independence Sampler 19

The Gibbs sampler 21

Running several chains 24

A strategy in Bayesian computing 28

Background Material

Law of Total Probability

- If events $A_1, \dots, A_k$ partition a sample space S into mutually exclusive and exhaustive nonempty events, then the *Law of Total Probability* states that the probability of an event B is

$$\begin{aligned} P(B) &= P(A_1B) + \cdots + P(A_kB) \\ &= P(B|A_1)P(A_1) + \cdots + P(B|A_k)P(A_k) \\ &= \sum_{j=1}^{k} P(B|A_j)P(A_j) \end{aligned}$$

- For continuous random variables X and Y we have the distributional form

$$f_Y(y) = \int_{-\infty}^{\infty} f_{Y|X=x}(y) f_X(x) dx.$$

- For discrete random variables X and Y we can write the distributional form

$$f_Y(y) = P(Y = y) = \sum_x P(Y = y|X = x)P(X = x).$$

Bayes Theorem

- Bayes Theorem provides a method for inverting conditional probabilities.
- In simplest form, if A and B are events and $P(B) > 0$, then
$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$
Often the Law of Total Probability is applied to compute $P(B)$ in the denominator.
- For continuous random variables the distributional form is
$$f_{X|Y=y}(x) = \frac{f_{Y|X=x}(y)f_X(x)}{f_Y(y)} = \frac{f_{Y|X=x}(y)f_X(x)}{\int_{-\infty}^{\infty} f_{Y|X=x}(y)f_X(x)dx}$$
- For discrete random variables
$$f_{X|Y=y}(x) = P(X = x|Y = y) = \frac{P(Y = y|X = x)P(X = x)}{\sum_x P(Y = y|X = x)P(X = x)}.$$

Bayesian Statistics

- In the frequentist approach to statistics, the parameters of a distribution are considered to be fixed but unknown constants. The Bayesian approach views the unknown parameters of a distribution as random variables.
- In Bayesian analysis, probabilities can be computed for parameters as well as the sample statistics.
- Bayes Theorem allows one to revise their prior belief about an unknown parameter based on observed data. The prior belief reflects the relative weights that one assigns to the possible values for the parameters.
- Suppose that X has the density $f(x|\theta)$. The conditional density of θ given the sample observatins $x_1, \ldots, x_n$ is called the posterior density,
$$f_{\theta|x}(\theta) = \frac{f(x_1, \ldots, x_n|\theta)f_\theta(\theta)}{\int f(x_1, \ldots, x_n|\theta)f_\theta(\theta)d\theta},$$
where $f_\theta(\theta)$ is the pdf of the prior distribution of θ.
- The posterior distribution summarizes our modified beliefs about the unknown parameters, taking into account the data that has been observed.
- Then one is interested in computing posterior quantities such as posterior means, posterior modes, posterior standard deviations, etc.
- Note that any constant in the likelihood function cancels out of the posterior density. The basic relation is
$$\textit{posterior} \propto \textit{prior} \times \textit{likelihood},$$
which describes the shape of the posterior density up to a multiplicative constant.
- Often the evaluation of the constant is difficult and the integral cannot be obtained in closed form.
- In this topic, we will cover available Monte Carlo methods that do not require the evaluation of the constant in order to sample from the posterior distribution and estimate posterior quantities of interest.

Introduction to Discrete Markov Chains

- Suppose a person takes a random walk on a number line on the values 1, 2, 3, 4, 5, 6. If the person is currently at an interior values (2, 3, 4, or 5), in the next second she is equally likely to remain at that number or move to an adjacent number.
- If she does move, she is equally likely to move left or right.
- If the person is currently at one of the end values (1 or 6), in the next second she is equally likely to stay still or move to the adjacent location.
- This is a simple example of a discrete Markov chain.

  A Markov chain describes the probabilistic movement between a number of states.

- Here there are six possible states, 1 through 6, corresponding to the possible locations of the walker. Given that the person is at current location, she moves to other locations with specified probabilities.
- The probability that she moves to another location depends only on her currrent location and not on previous locations visited.
- We describe movement between states in terms of transition probabilities - they describe the likelihood of moving between all possible states in a single step in a Markov chain.

Transition Probability Matrix

- We summarize the transition probabilites by means of a transition matrix P:

$$P = \begin{bmatrix} .50 & .50 & 0 & 0 & 0 & 0 \\ .25 & .50 & .25 & 0 & 0 & 0 \\ 0 & .25 & .50 & .25 & 0 & 0 \\ 0 & 0 & .25 & .50 & .25 & 0 \\ 0 & 0 & 0 & .25 & .50 & .25 \\ 0 & 0 & 0 & 0 & .50 & .50 \end{bmatrix}$$

- The first row in P gives the probabilities of moving to all states 1 through 6 in a single step from location 1, the second row gives the transition probabilites in a single step from location 2, and so on.

Markov chain properties

- A Markov chain is **irreducible** if it is possible to go from every state to every state in one or more steps.
- The previous example is an irreducible Markov chain.
- Given that the person is in a particular state, if the person can only return to this state at regular intervals, then the Markov chain is said to be **periodic**.
- The previous example is **aperiodic** since it is not a periodic Markov chain.

Transition probability matrix operations

- We can represent one's current location as a probability row vector of the form

$$p = (p_1, p_2, p_3, p_4, p_5, p_6),$$

  where p_i represents the probability that the person is currently in state i.

- If p^j represents the location of the traveler at step j, then the locaton of the traveler at the $j+1$ step is given by the matrix product

$$p^{j+1} = p^j P.$$

Stationary Distribution

- Suppose we can find a probability vector w such that $wP = w$. Then w is said to be the stationary distribution.

  If a Markov chain is irreducible and aperiodic, then it has a unique stationary distribution. Moreover, the limiting distribution of this Markov chain, as the number of steps approaches infinity, will be equal to the stationary distribution.

- We can empirically demonstrate the existence of the stationary distribution of our Markov chain by running a simulation experiment.
- We start our random walk at a particular state, say location 3, and then simulate many steps of the Markov chain using the transition matrix P.
- The relative frequency of our traveler in the siz locations after many steps will eventually approach the stationary distribution w.

Example 1: Simulation Experiment

```r
P <- matrix(c(.5,.5,0,0,0,0,.25,.5,.25,0,0,0,0,.25,.5,.25,
  0,0,0,0,.25,.5,.25,0,0,0,0,.25,.5,.25,0,0,0,0,.5,.5),
         nrow=6, ncol=6, byrow = TRUE)
P
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,] 0.50 0.50 0.00 0.00 0.00 0.00
## [2,] 0.25 0.50 0.25 0.00 0.00 0.00
## [3,] 0.00 0.25 0.50 0.25 0.00 0.00
## [4,] 0.00 0.00 0.25 0.50 0.25 0.00
## [5,] 0.00 0.00 0.00 0.25 0.50 0.25
## [6,] 0.00 0.00 0.00 0.00 0.50 0.50
```

```r
s <- array(0, c(50000, 1))
# starting location for our traveler, state 3
s[1] <- 3
# perform a loop to simulate 50,000 draws
# from the Markov chain
for(j in 2: 50000)
    s[j] <- sample(1:6, size =1, prob = P[s[j-1],])
m <- c(500, 2000, 8000, 50000)
res <- matrix(NA, nrow=4, ncol=6)
for(i in 1:4)
    res[i,] = table(s[1: m[i]])/m[i]
colnames(res) <- 1:6
rownames(res) <- m
res
```

```
##             1        2        3        4        5       6
## 500   0.16600 0.204000 0.194000 0.182000 0.166000 0.08800
## 2000  0.13900 0.218500 0.196500 0.191500 0.170500 0.08400
## 8000  0.12425 0.226625 0.195875 0.191375 0.179875 0.08200
## 50000 0.10372 0.206360 0.198440 0.196600 0.196220 0.09866
```

- It appears from the output that the relative frequencies of the states are converging to the stationary distribution
$$w = (0.1, 0.2, 0.2, 0.2, 0.2, 0.1)$$
- We can confirm that w is indeed the stationary distribution of this chain by multiplying w by the transition matrix P.

```r
w <- c(0.1, 0.2, 0.2, 0.2, 0.2, 0.1)
w%*%P
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]  0.1  0.2  0.2  0.2  0.2  0.1
```

- It's also possible to arrive at the stationary distribution by just multipying the transition matrix by itself over and over again until it finally converge.

```r
options(digits = 4) # display  limit to four decimals
SD <- P
for(i in 1:20) { # 20-step transition matrix
  SD <- SD %*% P
}
print(SD)
```

```
##         [,1]   [,2]   [,3]   [,4]   [,5]    [,6]
## [1,] 0.12433 0.2393 0.2150 0.1849 0.1607 0.07572
## [2,] 0.11967 0.2318 0.2121 0.1878 0.1682 0.08035
## [3,] 0.10749 0.2121 0.2047 0.1954 0.1878 0.09247
## [4,] 0.09247 0.1878 0.1954 0.2047 0.2121 0.10749
## [5,] 0.08035 0.1682 0.1878 0.2121 0.2318 0.11967
## [6,] 0.07572 0.1607 0.1849 0.2150 0.2393 0.12433
```

```
# no convergence yet
for(i in 1:200) { # 220-step transition matrix
  SD <- SD %*% P
}
print(SD)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]  0.1  0.2  0.2  0.2  0.2  0.1
## [2,]  0.1  0.2  0.2  0.2  0.2  0.1
## [3,]  0.1  0.2  0.2  0.2  0.2  0.1
## [4,]  0.1  0.2  0.2  0.2  0.2  0.1
## [5,]  0.1  0.2  0.2  0.2  0.2  0.1
## [6,]  0.1  0.2  0.2  0.2  0.2  0.1
```

```
# convergence reached
```

- You'll note that you have a 6 by 6 matrix with all rows equal. This tells you that the n-step transition matrix has converged.

Your turn! Seatwork..

Given a Markov chain with the following transition matrix:

$$P = \begin{bmatrix} 0.6 & 0.1 & 0.3 \\ 0.1 & 0.7 & 0.2 \\ 0.2 & 0.2 & 0.6 \end{bmatrix}$$

1. Verify that the above Markov chian is irreducible and aperiodic.
2. Find the stationary distribution of the above Markov chain.

Markov Chain Monte Carlo (MCMC)

- Recall that Monte Carlo integration estimates the integral

$$\int_A g(t)dt$$

  with a sample mean, by restating the integration problem as an expectation with respect to some density function $f(\cdot)$.

- The integration problem then is reduced to finding a way to generate samples from the target density $f(\cdot)$.
- The MCMC approach to sampling from $f(\cdot)$ is to construct a Markov chain with stationary distribution $f(\cdot)$, and run the chain for a sufficiently long time until the chain converges (approximately) to its stationary distribution.

Bayesian Inference

- Many applications of Markov Chain Monte Carlo methods are problems that arise in Bayesian inference.
- From a Bayesian perspective, in a statistical model both the observables and the parameters are random.
- Given observed data $x = \{x_1, \ldots, x_n\}$, and parameters θ, x depends on the **prior** distribution $f_\theta(\theta)$.
- This dependence is expressed by the likelihood $f(x_1, \ldots, x_n|\theta)$.
- The joint distribution of (x, θ) is therefore

$$f_{x,\theta}(x, \theta) = f_{x|\theta}(x_1, \ldots, x_n|\theta) \times f_\theta(\theta).$$

- One can then update the distribution of θ conditional on the information in the sample $x = \{x_1, \ldots, x_n\}$, so that by Bayes Theorem the **posterior distribution** of θ is given by

$$\begin{aligned} f_{\theta|x}(\theta|x) &= \frac{f_{x,\theta}(x,\theta)}{f(x)} \\ &= \frac{f_{x,\theta}(x,\theta)}{\int f_{x,\theta}(x,\theta)d\theta} \\ &= \frac{f_{x|\theta}(x)f_\theta(\theta)}{\int f_{x|\theta}(x)f_\theta(\theta)d\theta} \\ &= \frac{f_{x|\theta}(x_1,\ldots,x_n|\theta)f_\theta(\theta)}{\int f_{x|\theta}(x_1,\ldots,x_n|\theta)f_\theta(\theta)d\theta} \end{aligned}$$

- Thus, the conditional expectation of a function $g(\theta)$ with respect to the posterior density is

$$E[g(\theta|x)] = \int g(\theta)f_{\theta|x}(\theta)d\theta = \frac{\int g(\theta)f_{x|\theta}(x)f_\theta(\theta)}{\int f_{x|\theta}(x)f_\theta(\theta)d\theta} \tag{1}$$

- In general,

$$E[g(Y)] = \frac{\int g(t)\pi(t)dt}{\int \pi(t)dt} \tag{2}$$

  where $\pi(\cdot)$ is proportional to a density or a likelihood.
- If $\pi(\cdot)$ is a density function, then equation $E[g(Y)]$ (2) is just the usual $E[g(Y)] = \int g(t)f_Y(t)dt$.
- If $\pi(\cdot)$ is a likelihood, then the normalizing constant in the denominatior is needed.

Integration Problem

- In Bayesian analysis, $\pi(\cdot)$ is a posterior density. The expectation (2) can be evaluated even if $\pi(\cdot)$ is known only up to a constant.
- This simplifies the problem because in practice the normalizing constant for a posterior density $f_{\theta|x}(\theta)$ is often difficult to evaluate.
- The practical problem is that the integrations in $E[g(Y)]$ (2) are often mathematically intractable, and difficult to compute by numerical methods, especially in higher dimensions.
- Markov Chain Monte Caro provides a methods for this type of integration problem.

Markov Chain Monte Carlo Integration

- The Monte Carlo estimate of $E[g(\theta)] = \int g(\theta) f_{\theta|x}(\theta) d\theta$ is the sample mean

$$\bar{g} = \frac{1}{m} \sum_{i=1}^{m} g(x_i),$$

  where $x_1, \ldots, x_m$ is a sample from the distribution with density $f_{\theta|x}$.

- If $x_1, \ldots, x_m$ is indeed a independent (random sample) then by LLN, the sample mean $\bar{g}$ converges in probability to $E[g(\theta)]$ as the sample size $m \to \infty$.
- In this case, one can draw a large Monte Carlo sample to obtained the desired precision in $\bar{g}$.
- The Markov Chain in MCMC is not needed since Monte Carlo integration can be used as described previously.
- However, in problem such as $E[g(\theta|x)]$ (1) it may be difficult to implement a method for generating independent observations from density $f_{\theta|x}$.
- Nevertheless, even if the sample observations are dependent, a Monte Carlo integration can be applied if the observations can be generated so that their joint density is roughly the same as the joint denisty of the sample.

Markov Chain Monte Carlo Methods

- Up to now we generated IID (independent and identically distributed) variables directly from the density of interest f or indirectly using either accept-reject, importance sampling methods, etc.
- A Markov chain $\{X_t\}$ is a sequence of dependent random variables

$$\{X_0, X_1, \ldots\}$$

  such that the probability distribution of X_t given the past variables depends only on X_{t-1}.

- This essentially is a continuous-valued generalization of the discrete Markov chain setup described earlier.
- Markov chain Monte Carlo methods instead generates correlated variables from a Markov chain.
- The chain (values) generated by the MCMC methods enjoy a very strong stability property, i.e., a stationary probability distribution exists by construction for those chains.
- A popular way of simulating from a general (posterior) distribution is by using Markov chain Monte Carlo (MCMC) methods.

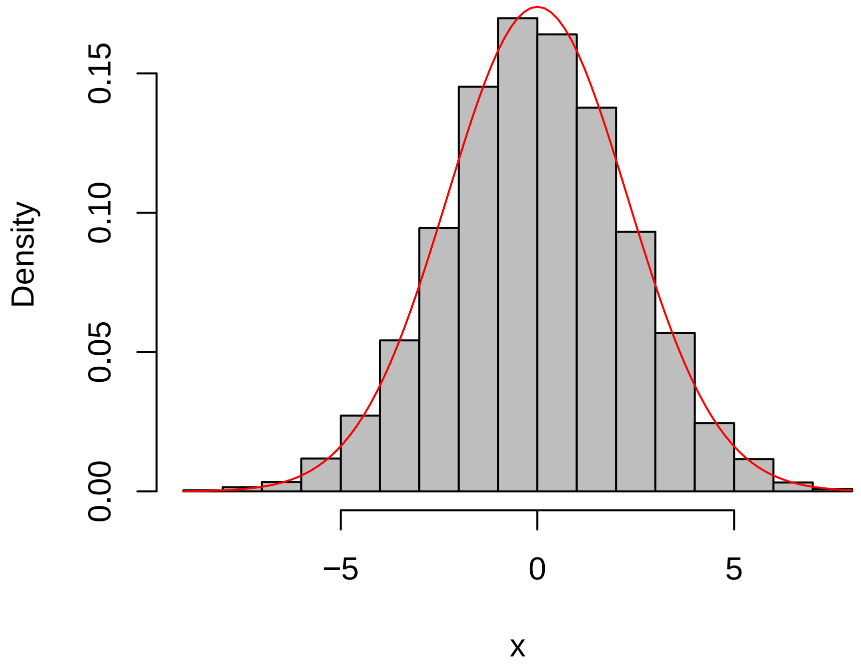
Example 2: Markov chain

Consider the Markov chain defined by

$$X_{t+1} = \gamma X_t + \epsilon_t, \quad \epsilon_t \sim N(0,1).$$

Simulating $X_0 \sim N(0,1)$, plot the histogram of a sample of X_t for $t \leq 10^4$ and $\gamma = 0.9$. Check the potential fit of the stationary distribution $N(0, 1/(1-\gamma^2))$.

```r
x <- 1:10^4
x[1] <- rnorm(1)
gamma <- 0.9
for (i in 2:10^4){
  x[i]<- gamma*x[i-1] + rnorm(1)
  }
hist(x, prob = T, col="grey", main="")
curve(dnorm(x, sd = 1/sqrt(1-gamma^2)), add=T, col = "red")
```



Metropolis-Hastings Algorithms

- The Metropolis-Hastings algorithm generates the Markov chain $\{X_0, X_1, \dots\}$ as follows:

1. Choose a proposal distribution $g(\cdot|X_t)$ with the same suppport set as the target distribution.
2. Generate X_0 from a distribution g.
3. Repeat (until the chain has converged to a stationary distribution according to some criteria):
    a. Generate Y from a distribution $g(\cdot|X_t)$.
    b. Generate U from $Uniform(0, 1)$.
    c. If
    $$U \leq \frac{f(Y)g(X_t|Y)}{f(X_t)g(Y|X_t)}$$
    accept Y and set $X_{t+1} = Y$; otherwise set $X_{t+1} = X_t$.

d. Increment t.

- Note in step (3c) the candidate point Y is accepted with probability

$$\alpha(X_t, Y) = \min\left(1, \frac{f(Y)g(X_t|Y)}{f(X_t)g(Y|X_t)}\right) \tag{3}$$

so that it is only necessary to know the density of the target distribution f up to a constant Y.

Example 2: Metropolis-Hastings Sampler

- Use the M-H sampler to generate a sample from a Rayleigh distribution with pdf

$$f(x) = \frac{x}{\sigma^2} e^{-x^2/(2\sigma^2)},\ x \geq 0,\, \sigma > 0.$$

- The Rayleigh distribution is used to model lifetimes subject to rapid aging, because the hazard rate is linearly increasing.
- For the proposal distribution, let us try the chi-squared distribution with degrees of freedom X_t, $\chi^2(X_t)$

Algorithm (MH Sampler)

1. Set $g(\cdot|X)$ to the density $\chi^2(X)$.
2. Generate X_0 from distribution $\chi^2(1)$ and store in `x[1]`.
3. Repeat for $i = 2, \ldots, N$:
   a. Generate Y from a $\chi^2(df = X_t) = \chi^2(df = x[i-1])$.
   b. Generate U from $Uniform(0, 1)$.
   c. With $X_t = x[i-1]$, compute

$$r(X_t, Y) \leq \frac{f(Y)g(X_t|Y)}{f(X_t)g(Y|X_t)}$$

   where
   - f is the Rayleigh density with parameter σ,
   - $g(Y|X_t)$ is the $\chi^2(df = X_t)$ density evaluated at Y, and
   - $g(X_t|Y)$ is the $\chi^2(df = Y)$ density evaluated at X_t.

   d. If $U \leq r(X_t, Y)$ accept Y and set $X_{t+1} = Y$; otherwise set $X_{t+1} = X_t$. Store X_{t+1} in `x[1]`.
4. Increment t.

```r
# function to compute Rayleigh pdf at x
f <- function(x, sigma) {
    if (any(x < 0)) return (0) # if x < 0, return f(x) = 0
    stopifnot(sigma > 0)       # stop if std. dev <= 0
    return((x / sigma^2) * exp(-x^2 / (2*sigma^2)))   # compute rayleigh pdf at x
}

set.seed(1234)
m <- 10000 # chain length
sigma <- 4
x <- numeric(m)
x[1] <- rchisq(1, df = 1) # generate first obs in the chain from candidate g()
```

```
k <- 0      # counter for number of rejected values
u <- runif(m)
for (i in 2:m) {
    # at each transition, the candidate point y
    # is generated from chi^2(df = X{t-1}
    xt <- x[i-1]
    y <- rchisq(1, df = xt)
    num <- f(y, sigma) * dchisq(xt, df = y)  #y is evaluated at f; xt is evaltd at g given y
    den <- f(xt, sigma) * dchisq(y, df = xt) #xt is evaluated at f; y is evaltd at g given xt
    if (u[i] <= num/den)
        x[i] <- y    # if y is accepted, assign y to next value in the chain
      else {
        x[i] <- xt   # else assign old value of x to the next value
        k <- k+1     # k record number of y rejected
        }
    }
print(100*k/m)
```

```
## [1] 40.54
```

About 40% of the candidate points are rejected, so the chain is somewhat inefficient.

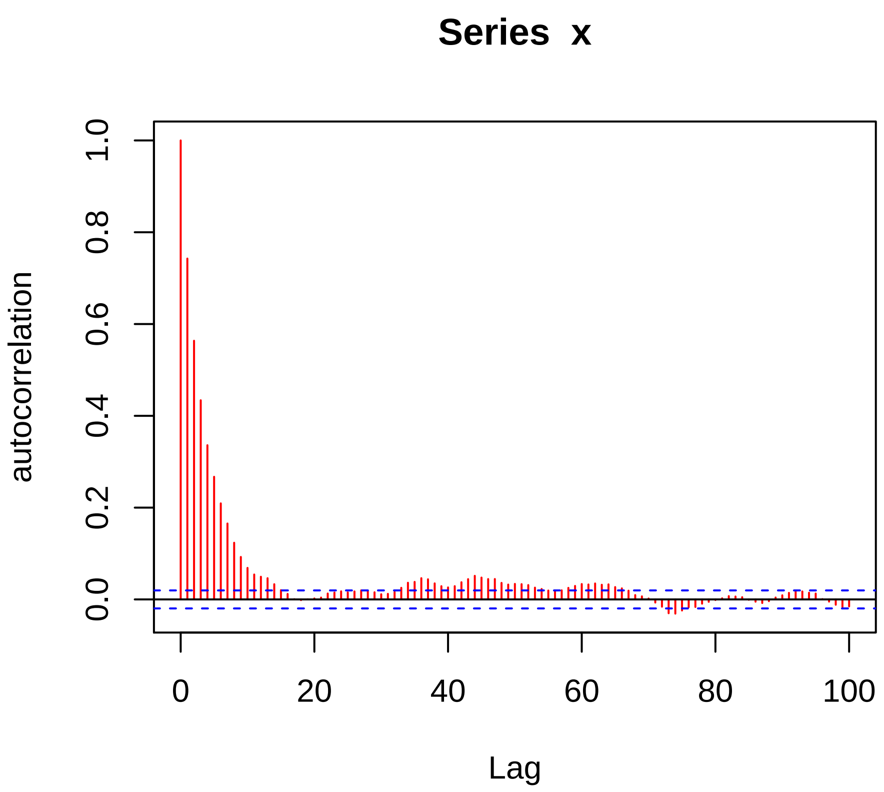
To check whether the generated values are independent (necessary for a random sample), we computed the autocorrelation for 100 adjacent values (or equivalently 10000/100 = 100 lags). It is easier to just check plot the autocorrelation values. To plot the autocorrelation, we use the function `acf` in the`VGAM` package in R.

```
library(VGAM) # install VGAM package
```

```
## Loading required package: stats4
```

```
## Loading required package: splines
```

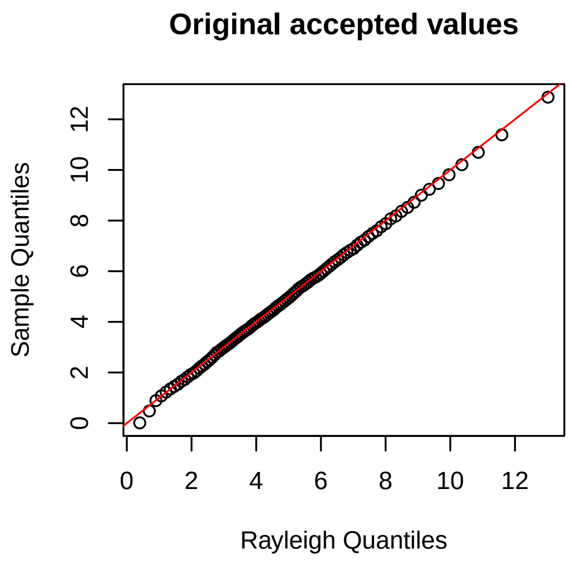
```
# computes autocorrelation for 100 adjacent values
acf(x, col = "red", lag.max = 100, ylab = "autocorrelation")
```

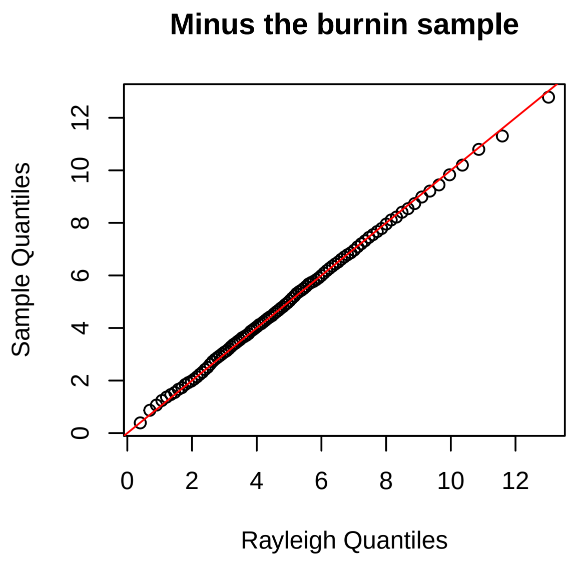


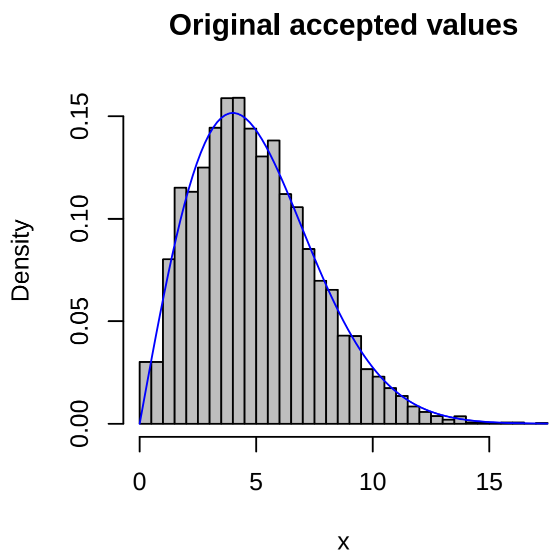
- Note the highly correlated values in the first part of the chain. This tells us that the chain has not converged yet. We refer these early values as **burnin sample**.
- In practice, we usually discard this first burnin sample values. From the autocorrelation plot above, we see that the autocorrelation started to hover around 0 after the 2000th (or 20th lag) value. With this, we discard the first 2000 values in the chain.
- The last 8000 values can now be treated as the generated values from the Rayleigh distribution.

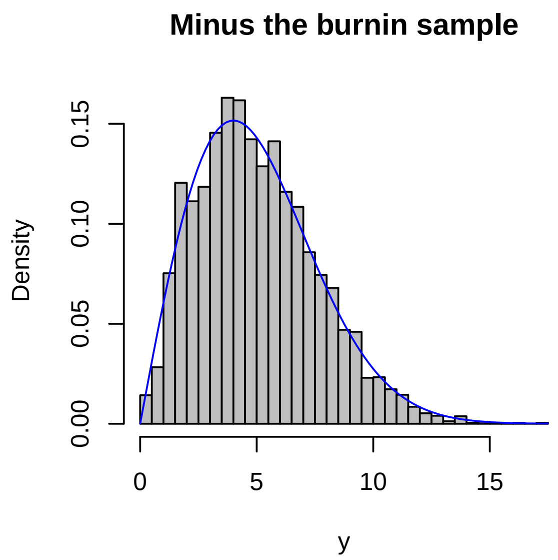
```
par(mfrow = c(2, 2))
a <- ppoints(100)
QR <- sigma * sqrt(-2 * log(1 - a))  #quantiles of Rayleigh
Qx <- quantile(x, a)
qqplot(QR, Qx, xlab ="Rayleigh Quantiles",
            ylab ="Sample Quantiles",
            main = "Original accepted values")
abline(0,1, col=2)

b <- 2001      #discard the burnin sample
y <- x[b: m]
Qy <- quantile(y, a)
qqplot(QR, Qy, xlab="Rayleigh Quantiles",
            ylab ="Sample Quantiles",
            main = "Minus the burnin sample")
abline(0, 1, col = 2)
```

```r
hist(x, breaks = 30, prob = T, col = "gray", main = "Original accepted values")
curve(drayleigh(x, scale=4), add=TRUE, col="blue")

hist(y, breaks = 30, prob = T, col = "gray", main = "Minus the burnin sample")
curve(drayleigh(x, scale=4), add=TRUE, col="blue")
```









The Metropolis Sampler

- The Metropolis-Hastings sampler is a generalization of the *Metropolis sampler*. In the Metropolis algorithm, the proposal distribution is symmetric. That is, the proposal distribution $g(\cdot|X_t)$ satisfies

  $$g(X|Y) = g(Y|X)$$

  so that in $\alpha(X_t, Y)$ (3) the proposal distribution g cancels from

  $$r(X_t, Y) = \frac{f(Y)g(X_t|Y)}{f(X_t)g(Y|X_t)},$$

  and the candidate point Y is accepted with probability

  $$\alpha(X_t, Y) = \min\left(1, \frac{f(Y)}{f(X_t)}\right).$$

Random Walk Metropolis

- The random walk Metropolis is an example of a Metropolis sampler.
- Suppose a candidate point Y is generated from a symmetric proposal distribution

  $$g(Y|X_t) = g(|X_t - Y|).$$

- Then at each iteration, a random increment Z is generated from $g(\cdot)$, and define Y as

  $$Y = X_t + Z$$

- Convergence of the random walk Metropolis is often sensitive to the choice of scale parameter.
- When variance of the increment is too large, most of the candidate points are rejected and the algorithm is very inefficient. If the variance of the increment is too small, the candidate points are almost all accepted, so the random walk Metropolis generates a chain that is almost like a true random walk, which is also inefficient.
- Select the parameter scale parameter to monitor the acceptance rates (between 50% to 85%).

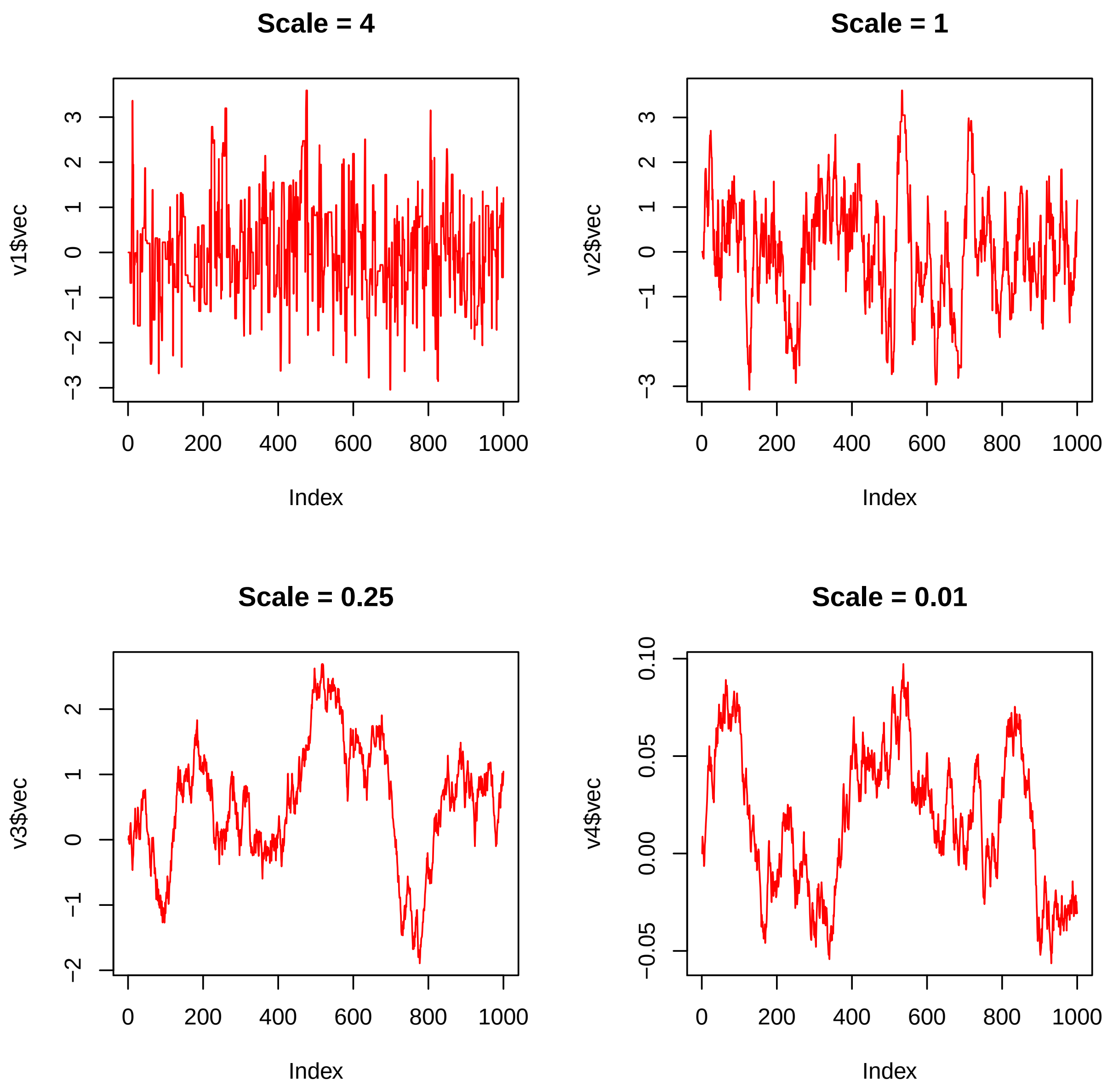
Example 3: Random Walk Metropolis Example

- Let's look at simulating from a target normal distribution with zero mean and unit variance using a Metropolis algorithm with uniform candidate/proposal distribution.
- First a candidate value (can) is generated from the candidate uniform distribution.
- Then the acceptance probability is computed, and the chain is updated appropriately depending on whether the proposed new value is accepted.
- We consider different scales for the uniform distribution to investigate the acceptance rates.

```r
metro.norm <-function (m, alpha,x) {
    vec <- vector("numeric", m)
    k <- 0
    vec[1] <- x
    u <- runif(m)
    for (i in 2:m) {
      innov <- runif(1, -alpha, alpha)
      can <- x + innov
      aprob <- min(1, dnorm(can)/dnorm(x))
      if (u[i] < aprob){
        x <- can
        k <- k+1 # acceptance rate
    }
      vec[i] <- x
    }
    list(vec=vec,accept = k/m)
}
# initial value
x = 0
m = 1000 # length of the chain
set.seed(17)
# generate from uniform scale = 4
v1<-metro.norm(m,4,x)
# generate from uniform scale = 1
v2<-metro.norm(m,1,x)
# generate from uniform scale = 0.25
v3<-metro.norm(m,0.25,x)
# generate from uniform scale = 0.01
v4<-metro.norm(m,0.01,x)

# acceptance rates
data.frame(v1$accept,v2$accept,v3$accept,v4$accept)
```

```
##   v1.accept v2.accept v3.accept v4.accept
## 1     0.417     0.785     0.953     0.999
```

```r
par(mfrow=c(2,2))
plot(v1$vec, col=2,type="l", main ="Scale = 4")
plot(v2$vec, col=2,type="l", main ="Scale = 1")
plot(v3$vec, col=2,type="l", main ="Scale = 0.25")
plot(v4$vec, col=2,type="l", main ="Scale = 0.01")
```



- With $\alpha = 4$, the first chain converges very fast but is very inefficient with acceptance rate of around 40%.
- With $\alpha = 1$, the second chain converges fast with an acceptable acceptance rate of around 80%.
- With $\alpha = 0.25$, the third chain converges much slower than compared to chains with larger scale. Also, there seems to be a semblance of random walk in the path of the chain. Acceptance is very high at above 95%.
- With $\alpha = 0.01$, the fourth chain behaves like a true random walk. All candidate values are accepted for this chain.

Quantiles comparison

- Let us compare the quantiles of these MCMC chains to the theoritical quantiles of the target normal distribution. We discard the burnin values of the fist 500 rows of each chain.

```
a <- c(.05, seq(.1, .9, .1), .95)
Q <- qnorm(a)
rw <- cbind(v1$vec, v2$vec, v3$vec, v4$vec)
mc <- rw[501:m, ]
colMeans(mc)
```

```
## [1] -0.08756 -0.13348  0.73782  0.01785
```

```
Qrw <- apply(mc, 2, function(x) quantile(x, a))
colnames(Qrw) <- c("alpha=4","alpha=1",
        "alpha=0.25","alpha=0.01")
round(cbind(Q, Qrw), 3)
```

```
##          Q alpha=4 alpha=1 alpha=0.25 alpha=0.01
## 5%  -1.645  -1.697  -2.206     -1.366     -0.038
## 10% -1.282  -1.422  -1.686     -1.034     -0.032
## 20% -0.842  -0.997  -1.188     -0.103     -0.022
## 30% -0.524  -0.611  -0.765      0.442     -0.001
## 40% -0.253  -0.372  -0.509      0.703      0.008
## 50%  0.000  -0.091  -0.235      0.853      0.019
## 60%  0.253   0.196   0.058      1.057      0.029
## 70%  0.524   0.535   0.378      1.317      0.037
## 80%  0.842   0.816   0.823      1.567      0.050
## 90%  1.282   1.093   1.587      2.152      0.068
## 95%  1.645   1.725   2.650      2.321      0.077
```

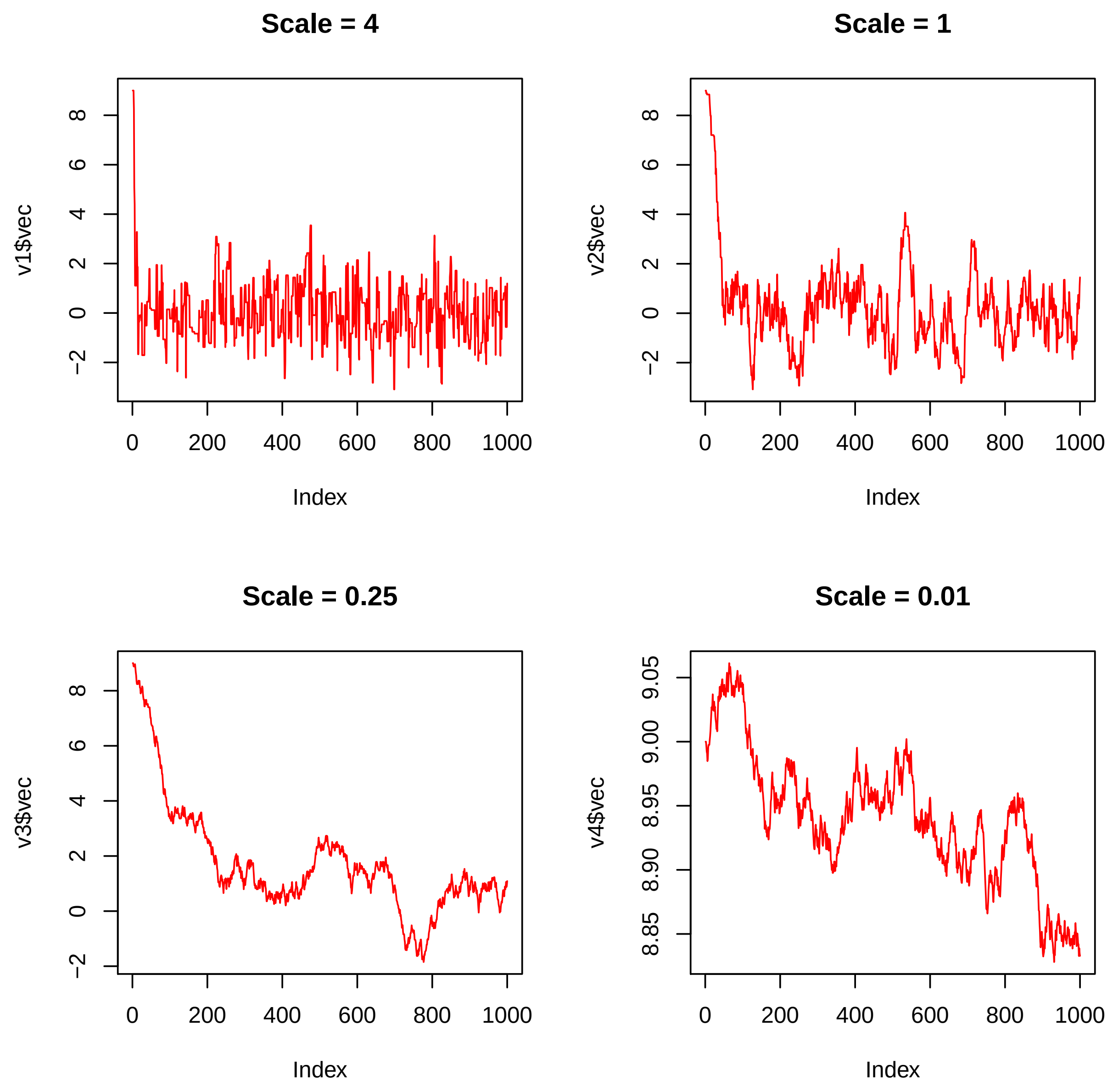
Random walk with different initial value

- What happens to the chain when the initial value is at $x = 9$ - far from the standard normal mean?

```
# initial value
x = 9
m = 1000
set.seed(17)
# generate from uniform scale = 4
v1<-metro.norm(m,4,x)
# generate from uniform scale = 1
v2<-metro.norm(m,1,x)
# generate from uniform scale = 0.25
v3<-metro.norm(m,0.25,x)
# generate from uniform scale = 0.01
v4<-metro.norm(m,0.01,x)

# acceptance rates
data.frame(v1$accept,v2$accept,v3$accept,v4$accept)
```

```
##   v1.accept v2.accept v3.accept v4.accept
## 1     0.407     0.769     0.903     0.978
```

```
par(mfrow=c(2,2))
plot(v1$vec, col=2,type="l", main ="Scale = 4")
plot(v2$vec, col=2,type="l", main ="Scale = 1")
plot(v3$vec, col=2,type="l", main ="Scale = 0.25")
plot(v4$vec, col=2,type="l", main ="Scale = 0.01")
```



```
a <- c(.05, seq(.1, .9, .1), .95)
Q <- qnorm(a)
rw <- cbind(v1$vec, v2$vec, v3$vec, v4$vec)
```

```r
mc <- rw[501:m, ]
Qrw <- apply(mc, 2, function(x) quantile(x, a))
colnames(Qrw) <- c("alpha=4","alpha=1",
        "alpha=0.25","alpha=0.01")
round(cbind(Q, Qrw), 3)
```

```
##          Q alpha=4 alpha=1 alpha=0.25 alpha=0.01
## 5%  -1.645  -1.703  -2.028     -1.320      8.842
## 10% -1.282  -1.425  -1.576     -0.988      8.847
## 20% -0.842  -1.008  -1.189     -0.057      8.863
## 30% -0.524  -0.639  -0.838      0.487      8.891
## 40% -0.253  -0.417  -0.524      0.749      8.905
## 50%  0.000  -0.116  -0.182      0.898      8.915
## 60%  0.253   0.187   0.077      1.103      8.927
## 70%  0.524   0.538   0.362      1.363      8.935
## 80%  0.842   0.809   0.795      1.612      8.945
## 90%  1.282   1.206   1.672      2.198      8.965
## 95%  1.645   1.680   2.693      2.366      8.984
```

- With $\alpha = 4$, the first chain converges very fast but is very inefficient with acceptance rate of around 40%. Also, the initial value plays a little role in the convergence of this chain.
- With $\alpha = 1$, the second chain converges fast with an acceptable acceptance rate of around 80%. The initial value also plays small role since the chain converges at the 50-100 steps.
- With $\alpha = 0.25$, the third chain converges much slower than compared to chains with larger scale. If initial value is at $X = 9$, the chain did not covergence until 200-400 steps. Also, there seems to be a semblance of random walk in the path of the chain. Acceptance is very high at slightly above 90% dependending on the initial value.
- With $\alpha = 0.01$, the fourth chain behaves like a true random walk. If initial value is at $X = 9$, the chain did not converge even after 1000 steps. Almost all candidate value are accepted for this chain.
- If initial value is from the center of the target distribution, then large scales are advantageous since the empirical quantiles (after discarding the burnin sample) is closer to the true quantiles. Of course, you trade for inefficiency in this case.

The Independence Sampler

- Another special case of the Metropolis-Hastings sampler.
- The proposal distribution in the independence sampling algorithm does not depend on the previous value of the chain. Meaning, r.v. Y is independent of X_t.
- Thus, $g(Y|X_t) = g(Y)$ and the acceptance probability is

$$\alpha(X_t, Y) = \min\left(1, \frac{f(Y)g(X_t)}{f(X_t)g(Y)}\right).$$

- The independence sampler is easy to implement and tends to work well when the proposal density is a close match to the target density.

Example 4: Metropolis Independence sampler

- We want to simulate a sequence $X_1, \dots, X_m$ such that X_i follows a laplace distribution $Laplace(2,1)$ with pdf

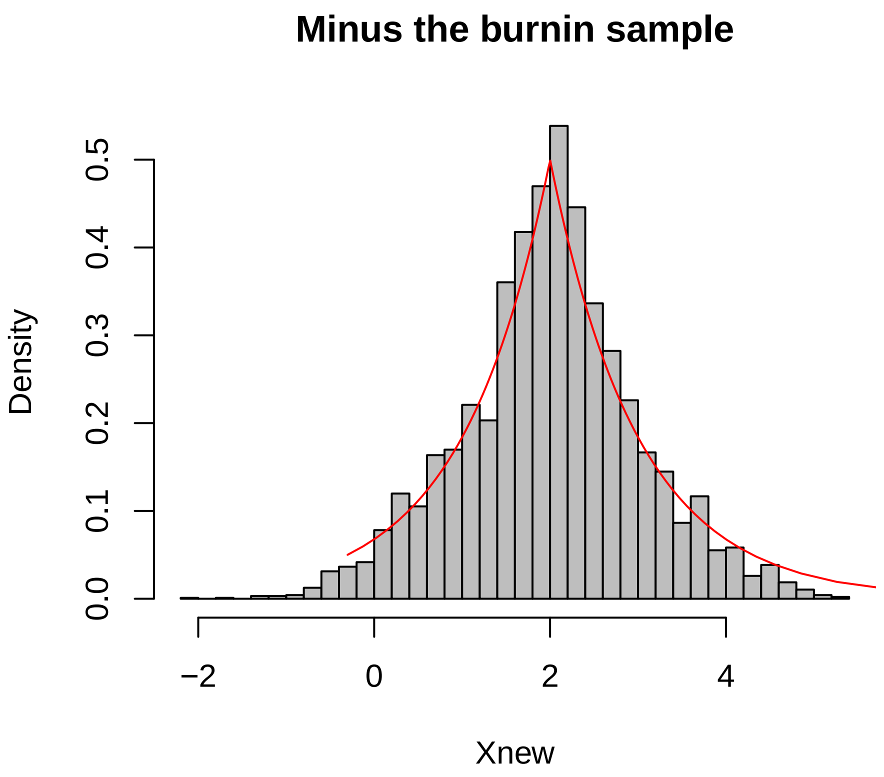
$$f(x) = \frac{1}{2}\exp(-|x-2|),$$

  using $N(2,1)$ as proposal distribution.

```r
# define laplace density
f <- function(x,b=1,mu=2){exp(-abs(x-mu)/b)/(2*b)}

m <- 5000  #lenght of the chain
X <- numeric(m)
X[1] <- rnorm(1,mean=2) # Init
for(i in 2:m) # Iterate random update
{
    Y <- rnorm(n=1,mean=2) # propose candidate value
    # compute Metropolis ratio
    # note the independence of the proposal pdf
    R <- (f(Y) * dnorm(X[i],mean=2)) /
            (f(X[i]) * dnorm(Y,mean=2))
    # compute acceptance probability
    p <- min(1,R)
    # toss an unfair coin to decide
    # if candidate accepted
    accept <- rbinom(n=1,size=1,prob=p)
    X[i] <- ifelse(accept==1, Y, X[i-1])
}

Xnew <- X[201:m]
# Mean of the chain w/ and w/o the burnin sample
c(mean(X),mean(Xnew))
```

```
## [1] 2.004 2.004
```

```r
u <- seq(-1/2+0.05,1/2,length.out=100)
mu <- 2
b <- 1
QL <- mu - b*sign(u)*log(1-2*abs(u))
hist(Xnew, breaks = 30, prob = T, col = "gray",
     main ="Minus the burnin sample")
# true laplace density curve
lines(QL,f(QL,b,mu),col="red")
```



The Gibbs sampler

- The Gibbs sampler is often applied when the target is a multivariate distribution.
- Suppose that all the univariate conditional densities are fully specified and it is reasonably easy to sample from them.
- The chain is generated by sampling from the marginal distributions of the target distribution, and every candidate point is therefore accepted.
- Let $X = (X_1, \ldots, X_n)$ be a random vector in $\mathbb{R}^d$. Define the $d-1$ dimensional random vectors

$$X_{(-j)} = (X_1, \ldots X_{j-1}, X_{j+1}, \ldots, X_d),$$

  and denote the corresponding univariate conditional density of X_j given $X_{(-j)}$ by $f(X_j|X_{(-j)})$.
- The Gibbs sampler generates the chain by sampling from each of the d conditional densities $f(X_j|X_{(-j)})$.

  The Gibbs Sample Algorithm

Denote X_t by $X(t)$.

1. Initialize $X(0)$ at time $t = 0$.

2. For each iteration, indexed $t = 1, 2, \ldots$ repeat:

   a. Set $x_1 = X_1(t-1)$.
   b. For each coordinate $j = 1, 2, \ldots, d$
      i. Generate $X_j^*(t)$ from $f(X_j|x_{(-j)})$.
      ii. Update $x_j = X_j^*(t)$.
   c. Set $X(t) = (X_1^*(t), \ldots, X_d^*(t))$ (every candidate is accepted).
   d. Increment t.

Example 5: Gibss sampler: Bivariate distribution

- Generate a bivariate normal distribution with mean vector (μ_1, μ_2), variances σ_1^2,σ_2^2, and correlation ρ, using Gibbs sampling.
- In the bivariate case, $X = (X_1, X_2)$, $X_{(-1)} = X_2$,$X_{(-2)} = X_1$. The conditional densities of a bivariate normal distribution are univariate normal with parameters

$$\mu_{2|1} = E[X_2|x_1] = \mu_2 + \rho\frac{\sigma_2}{\sigma_1}(x_1 - \mu_1),$$
$$\sigma_{2:1}^2 = Var(X_2|x_1) = (1-\rho^2)\sigma_2^2,$$

  and the chain is generated by sampling from

$$f(x_1|x_2) \sim Normal(\mu_{2|1}, \sigma_{1|2}^2),$$
$$f(x_2|x_1) \sim Normal(\mu_{1|2}, \sigma_{2|1}^2).$$

  Gibbs Sampler for Bivariate Distribution

For a bivariate distribution (X_1, X_2), at each iteration the Gibbs sampler

1. Set $(x_1, x_2) = X(t-1)$;
2. Generates $X_1^*(t)$ from $f(X_1|x_2)$;
3. Updates $x_1 = X_1^*(t)$;
4. Generates $X_2^*(t)$ from $f(X_2|x_1)$;
5. Sets $X(t) = (X_1^*(t), X_2^*(t))$.

```r
#initialize constants and parameters
N <- 5000               #length of chain
burn <- 1000            #burn-in length
X <- matrix(0, N, 2)    #the chain, a bivariate sample

rho <- -.75            #correlation
mu1 <- 0
mu2 <- 2
sigma1 <- 1
sigma2 <- .5
s1 <- sqrt(1-rho^2)*sigma1
s2 <- sqrt(1-rho^2)*sigma2
```

```r
###### generate the chain #####
set.seed(17)
X[1, ] <- c(mu1, mu2)            #initialize
for (i in 2:N) {
    x2 <- X[i-1, 2]
    m1 <- mu1 + rho * (x2 - mu2) * sigma1/sigma2
    X[i, 1] <- rnorm(1, m1, s1)
    x1 <- X[i, 1]
    m2 <- mu2 + rho * (x1 - mu1) * sigma2/sigma1
    X[i, 2] <- rnorm(1, m2, s2)
}
b <- burn + 1
x <- X[b:N, ]

# compare sample statistics to parameters
colMeans(x)
```

```
## [1] 0.002167 2.001962
```

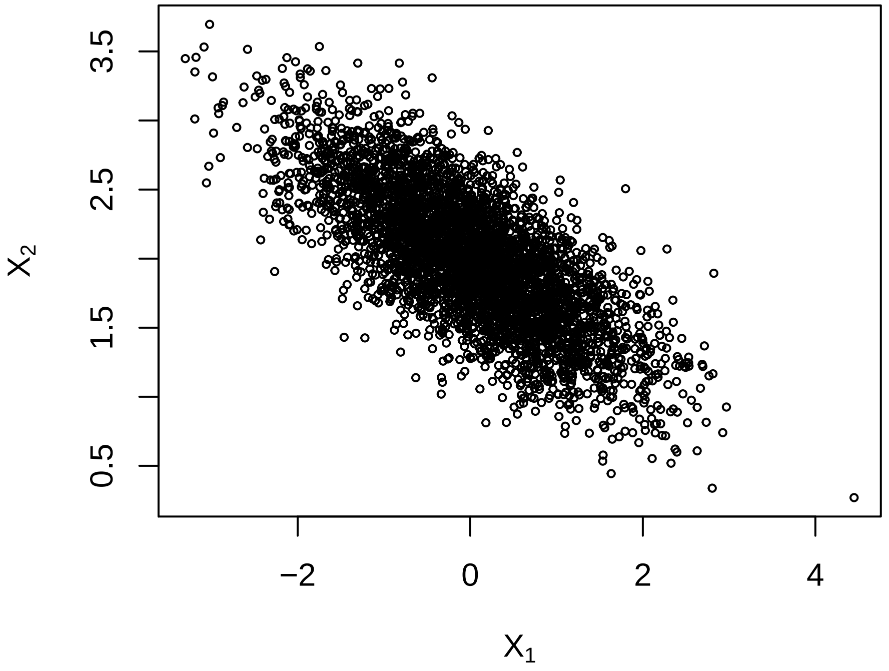
```r
cov(x)
```

```
##          [,1]    [,2]
## [1,]  0.9622 -0.3567
## [2,] -0.3567  0.2412
```

```r
cor(x)
```

```
##          [,1]    [,2]
## [1,]  1.0000 -0.7405
## [2,] -0.7405  1.0000
```

```r
plot(x, main="", cex=.5, xlab=bquote(X[1]),
           ylab=bquote(X[2]), ylim=range(x[,2]))
```



The sample means, variances, and correlation are close the true parameters, and the plot displays symmetry of the the bivarate normal with negative correlation.

Running several chains

- In previous examples, we have seen that various Metropolis-Hastings algorithms generated chains that have not converged to the target distribution.
- In general, for arbitrary M-H sampler the number of iterations that are sufficient for approximate convergence to the target distribution or what length burn-in sample sample is required are unknown.
- Gelman and Rubin's (1992) paper titled "A single sequence from the Gibbs Sampler gives a falls sense of security" provides examples of slow convergence that cannot be detected by examining a single chain.
- A single chain may appear to have converged because the generated values have a small variance within a local part of the support set of the target distribution, but in reality the chain has not explored all of the support set.
- By examing several parallel chains, slow convergence should be more evident, particularly if the initial values of the chain are overdispersed with respect to the target distribution.

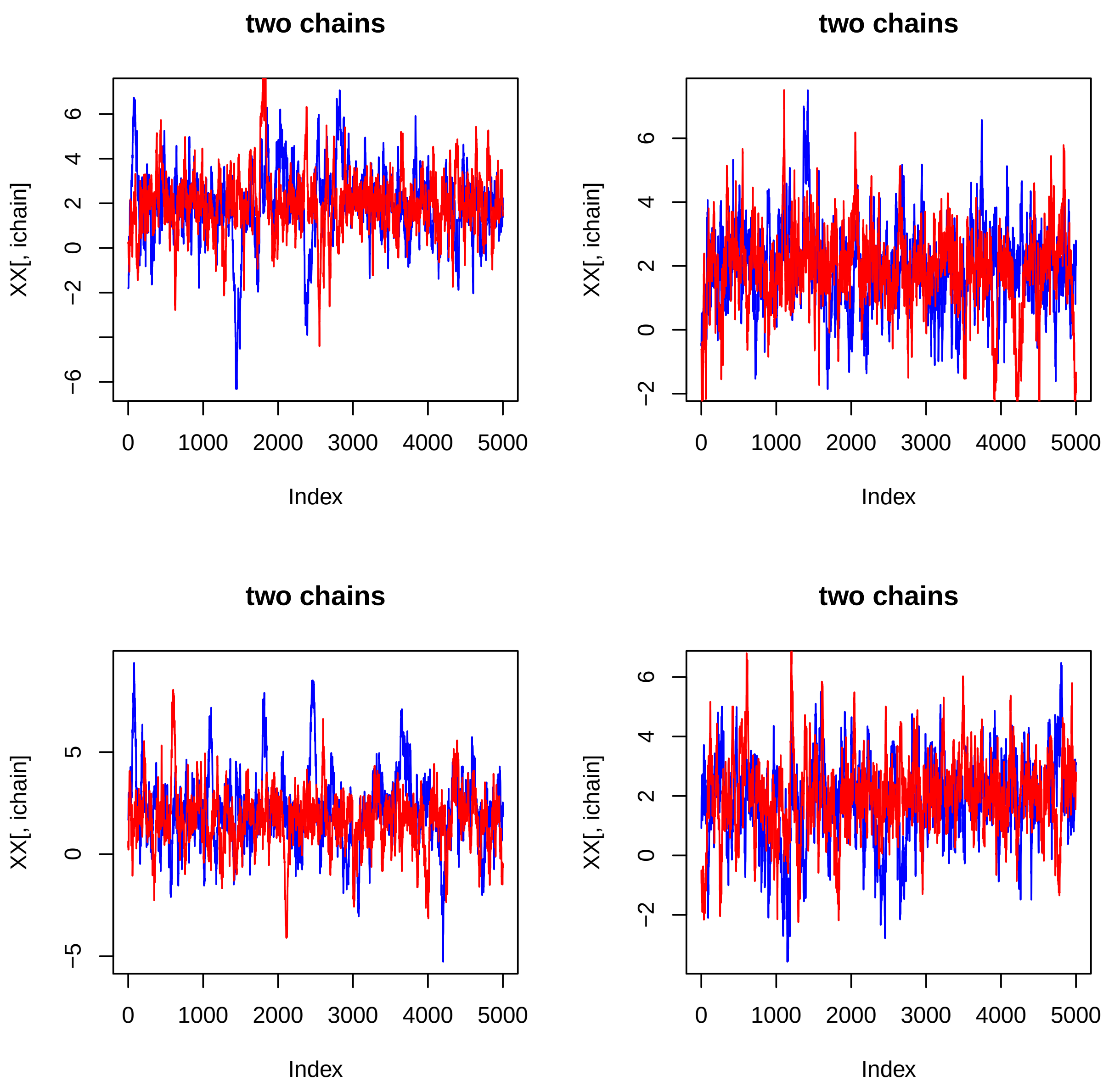
Example 5: Running several chains

- Recall the previous example on laplace distribution. Consider simulating a sequence $X_1, \ldots, X_m$ such that each X_i follows a laplace distribution with pdf

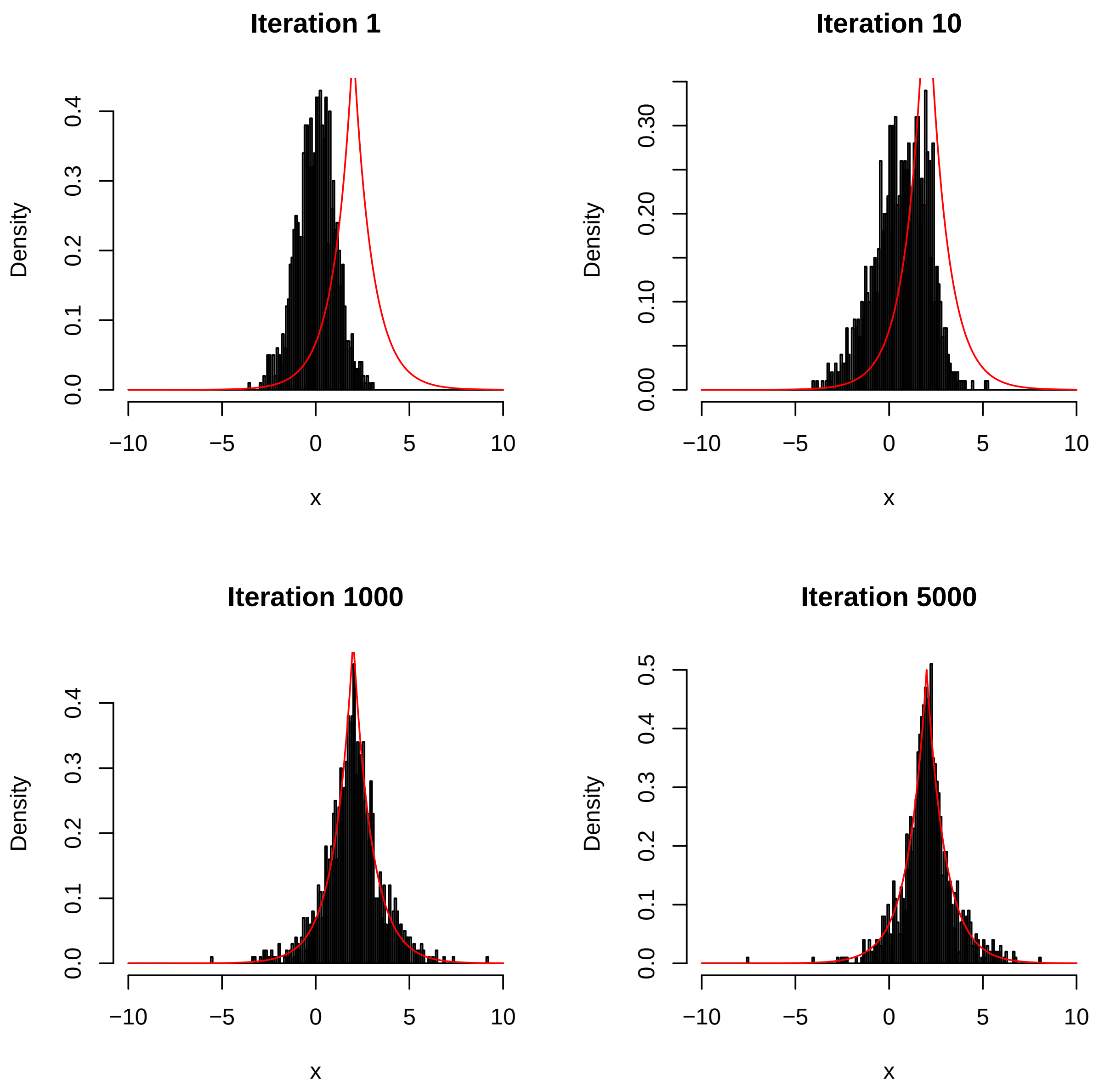
$$f(x) = \frac{1}{2}\exp(-|x-2|).$$

- For this example, we will consider the general Metropolis-Hastings algorithm with $N(\mu, \sigma^2 = 0.25)$ as proposal distribution (see discussion on Metropolis Independence Sampler).

```r
# M-H multiple chain
# define db exp density
f <- function(x, b=1, mu=2){exp(-abs(x - mu)/b)/(2*b)}

niterations <- 5000   # length of chain
nchains <- 1000          # number of parallel chains

XX <- matrix(nr = niterations, nc = nchains, data = NA)
XX[1,] <- rnorm(n = nchains)  # initial values

for(i in 2:niterations) # Iterate random update
{
 # propose candidate values
 Y <- rnorm(n=nchains,mean=XX[i-1,],sd=.5)
 # compute general Metropolis-Hastings ratios
 num <- f(Y) * dnorm(x=XX[i-1,],mean=Y,sd=.5)
 den <- f(XX[i-1,]) * dnorm(x=Y,mean=XX[i-1,],sd=.5)
 R <- num/den
 # compute acceptance probabilities
 p = rep(1,nchains)
 p[R<1] = R[R<1]   # store the prob'y < 1
 # toss an unfair coin to decide if candidates are accepted
 accept <- as.logical(rbinom(n=nchains,size=1,prob=p))
 XX[i,accept] = Y[accept]
 XX[i,!accept] = XX[i-1,!accept]
}

par(mfcol = c(2,2))
for(ichain in 1:4) {
  plot(XX[,ichain],type="l", col="blue",
    main ="two chains")
  lines(XX[,(ichain+4)],col="red")
}
```



```
par(mfrow=c(2,2))
for(iter in c(1,10,1000,5000)) {
  hist(XX[iter,],breaks=seq(-10,10,0.1),
    main=paste("Iteration",iter),xlab="x",prob=TRUE)
  lines(seq(-10,10,0.01),f(seq(-10,10,0.01)),col=2)
}
```



```r
par(mfcol = c(1,1))

for(iter in c(1,10,1000,5000))
{print(paste("Iteration",iter,"  ;
  Variance=",var(XX[iter,]))) }
```

```
## [1] "Iteration 1   ; \n  Variance= 1.01899023414691"
## [1] "Iteration 10   ; \n  Variance= 2.02627877307596"
## [1] "Iteration 1000   ; \n  Variance= 2.23941908890575"
## [1] "Iteration 5000   ; \n  Variance= 1.95416824772201"
```

A strategy in Bayesian computing

- The topic for this section is lifted from Bayesian Computation with R 2nd edition by James Albert.
- For a particular Bayesian inference problem, we assume that one has defined the log posterior density by an R function.
- A good approach for summarizing this density is to set up a Markovchain simulation algorithm.
- The Metropolis-Hastings random walk, independence chains, and Gibbs sampling algorithm are attractive Markov chains snce they are easy to program and require relatively little prior input.
- But these algorithms do require some initial guessess at the location and spread of the parameter vector θ.
- These initial guesses can be found by non-Bayesian methods such as the method of moments or maximum likelihood.
- Alternatively, one can obtain an approximation to the posterior distribution by finding the mode using some optimization algorithm.
- For example, Nelder and Mead's method gives the posterior mode and an approximation to the variance-covariance matrix that can be used in specifying the proposal densities in the Metropolis-Hastings algorithm.
- In the next example, we use the function `{laplace} in the {LearnBayes}, package accompanying the BCWR book, to locate the posterior density.`
- `The laplace function will yield estimates of the posterior mode and covariance matrix that can be used to construct efficient Metropolis-Hastings algorithms for simulating from the exact joint posterior distribution.`

  Once one hast decided that the simulated stream of values represents an approximate sample from the posterior, then one can summarize this sample in different ways to perform inferences about θ.

Example 6: Analysis of the Stanford Heart Transplant Data

- The main goal of the study involving this data is to decide if heart transplantation extends a patient's life.
- A popular moode used to describe this data is the Pareto Model which assumes individual patients in the nontransplant group have exponential lifetime distributions with mean $1/\theta$, where θ is assumed to vary between patients and is drawn from a gamma distribution with density

$$f(\theta) = \frac{\lambda^p}{\Gamma(p)}\theta^{p-1}\exp(-\lambda\theta).$$

- Patients in the transplant group have similar exponential life distribution, where the mean $1/(\theta\tau)$.
- This model assumes that the patient's risk of death changes by an unknown constant factor $\tau > 0$. If $\tau = 1$, then there is no increased risk by having a transplant operation.

Model for Stanford Heart Transplant Data

- Suppose the surival times $\{x_i\}$ are observed for N nontransplant patients. For n of these patients, x_i represents the actual survival time (in days); the remaining $N - n$ patients were still alive at the end of the study.
- For the M patients that have a heart transplant, let y_i and z_j denote the time to transplant and survival time; m of these patients died during the study.
- The unknown parameter vector is (τ, λ, p), with the likelihood function given by

$$L(\tau,\lambda,p) \propto \prod_{i=1}^{n} \frac{p\lambda^p}{(\lambda+x_i)^{p+1}} \prod_{i=n+1}^{N} \left(\frac{\lambda}{\lambda+x_i}\right)^p \times \prod_{j=1}^{m} \frac{\tau p \lambda^p}{(\lambda+y_j+\tau z_j)^{p+1}} \prod_{j=m+1}^{M} \left(\frac{\lambda}{\lambda+y_j+\tau z_j}\right)^p,$$

  where all the parameters are positive.
- Suppose we place a uniform prior on (τ, λ, p), so the posterior density is proportional to the likelihood.
- For convenience, let us transform the parameters by

$$\theta_1 = \log\tau, \quad \theta_2 = \log\lambda, \quad \theta_3 = \log p.$$

- The posterior density of $\theta = (\theta_1, \theta_2, \theta_3)$ is given by

$$g(\theta|data) \propto L(\exp(\theta_1), \exp(\theta_2), \exp(\theta_3)) \prod_{i=1}^{3} \exp(\theta_i)$$

- The dataset `standfordheart` in the `LearnBayes` package contains the data for 82 patients; for each patient, there are four variables: `survtime` , the survival time; `transplant`, a variable that is 1 or 0 depending on whether the patient had a transplant or not; `timetotransplant`, the time a transplant patient waits for the operation; and `state`, a variable that indicates if the survival time was censored (0 if the patient died and 1 if he was still alive).

```
library(LearnBayes)
```

```
##
## Attaching package: 'LearnBayes'

## The following object is masked from 'package:VGAM':
##
##     laplace
```

```
data(stanfordheart)

start <- c(0,3,-1)

laplacefit <- laplace(transplantpost, start, stanfordheart)
laplacefit
```

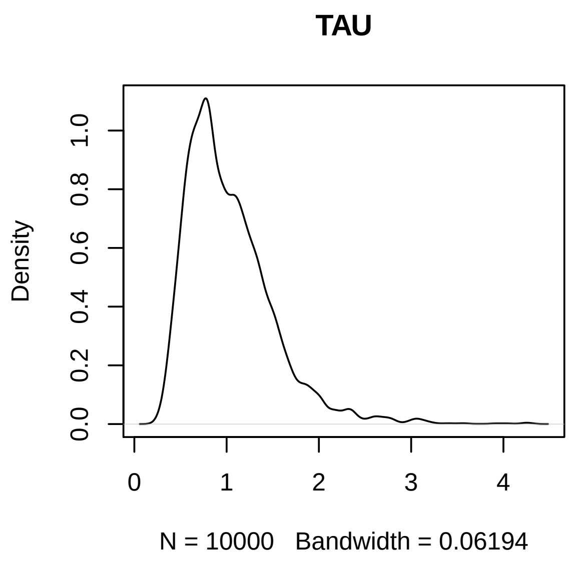
```
## $mode
## [1] -0.09211  3.38385 -0.72334
##
## $var
##           [,1]      [,2]     [,3]
## [1,]  0.172789 -0.009282 -0.04995
## [2,] -0.009282  0.214737  0.09301
## [3,] -0.049952  0.093013  0.06892
##
## $int
## [1] -376.3
##
## $converge
## [1] TRUE
```

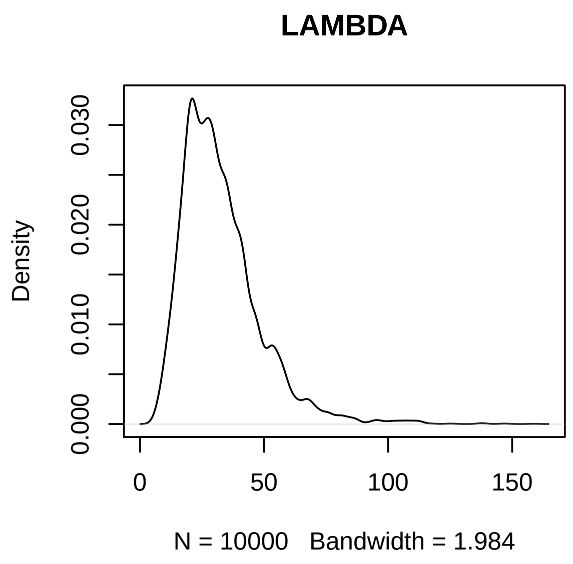
```
proposal <- list(var = laplacefit$var, scale=2)
s <- rwmetrop(transplantpost, proposal, start,10000, stanfordheart)
s$accept
```

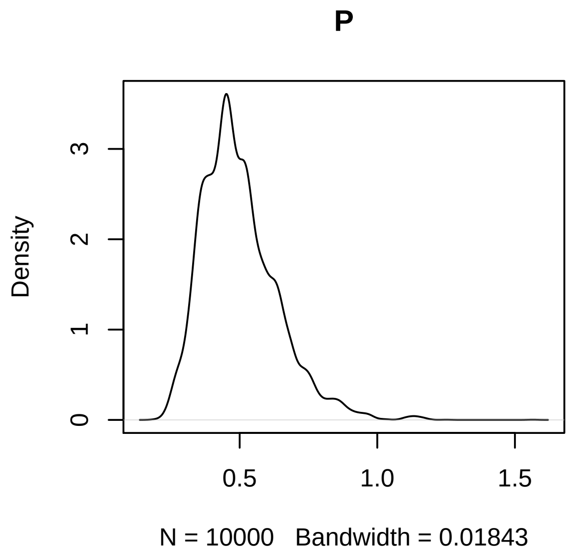
```
## [1] 0.1824
```

```
par(mfrow = c(2,2))
tau <- exp(s$par[, 1])
plot(density(tau), main="TAU")
lambda <- exp(s$par[, 2])
plot(density(lambda), main="LAMBDA")
p <- exp(s$par[, 3])
plot(density(p), main="P")

apply(exp(s$par), 2, quantile,c(.05,.5,.95))
```

```
##       [,1]  [,2]   [,3]
## 5%  0.4615 13.57 0.3185
## 50% 0.9389 29.54 0.4769
## 95% 1.8971 60.93 0.7525
```


TAU
Density
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
N = 10000 Bandwidth = 0.06194


LAMBDA
Density
0.000
0.010
0.020
0.030
0
50
100
150
N = 10000 Bandwidth = 1.984


P
Density
0
1
2
3
0.5
1.0
1.5
N = 10000 Bandwidth = 0.01843