STAT 428 Homework 5

Yiming Gao (NetID: yimingg2)

2017/3/28

Contents

Exercise 1 (Rizzo 9.3)

Use the Metropolis-Hastings sampler to generate random variables from a standard Cauchy 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 density function of Cauchy(θ, η) is

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

The standard Cauchy has Cauchy($\theta = 1$, $\eta = 0$) density, i.e.

$$f(x) = \frac{1}{\pi(1 + x^2)}, \quad -\infty < 0 < \infty$$

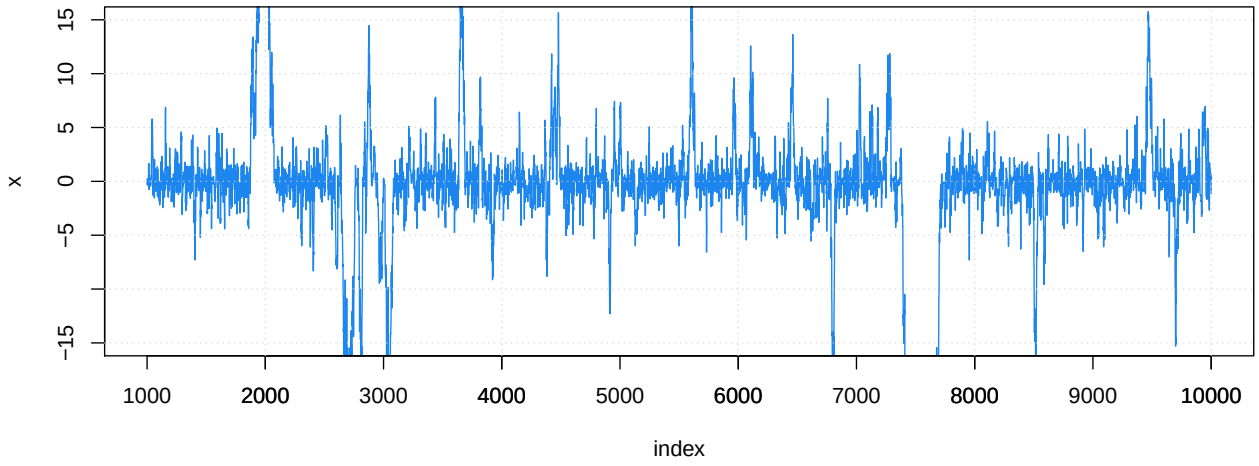
We use the normal distribution as the proposal density with mean given by the previous value in the chain and the standard deviation given by $\sigma = 2$. We will discard the first 1000 of the chain.

```
set.seed(27)
# cauchy density
f1 <- function(x) {
  1/(pi*(1 + x^2))
}

# initialize parameters
m = 10000
b = 1001 # burn-in period
x = numeric(m)
x[1] = rnorm(1, 0, 2)
```

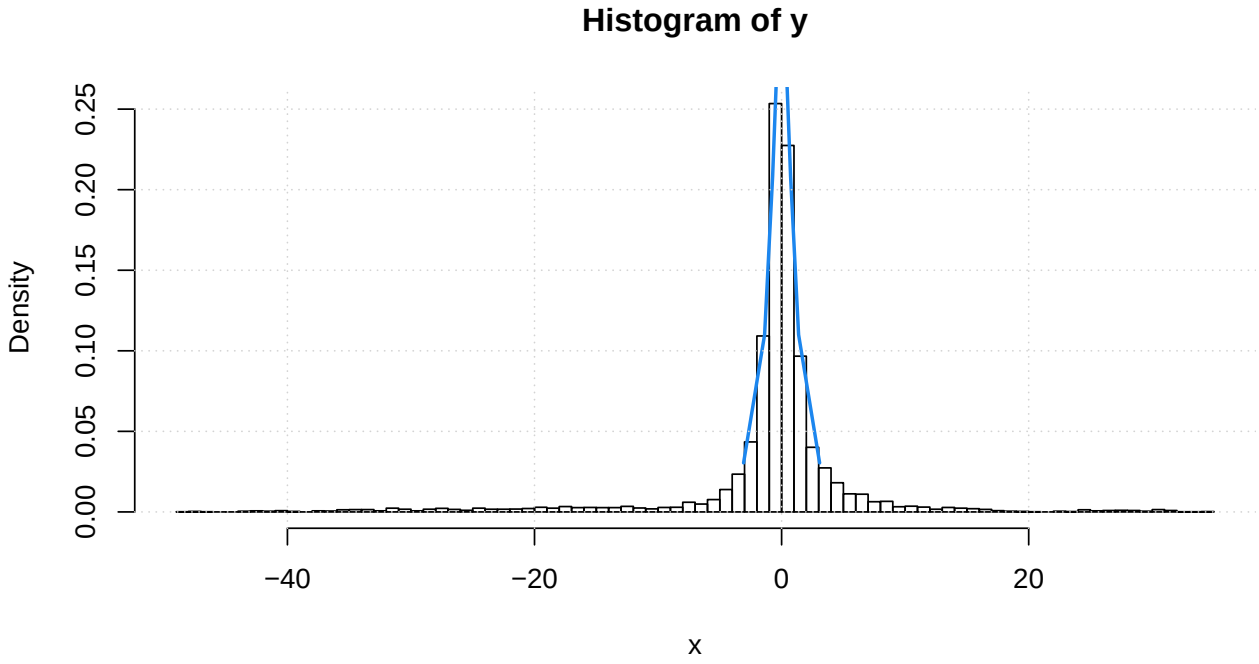
```r
k = 0 # count the number of times y is rejected
u = runif(m)

# run the chain
for (i in 2:m) {
  xt = x[i-1]
  y = rnorm(1, xt, 2)
  num = f1(y) * dnorm(xt, y, 2)
  den = f1(xt) * dnorm(y, xt, 2)
  if (u[i] <= num/den) {
    x[i] = y
    }
  else {
    x[i] = xt
    k = k + 1 # y is rejected
  }
}

index = b:m
y = x[index]
prob = seq(0.1, 0.9, 0.1)
# standard cauchy
std_cauchy = qcauchy(prob)
# simulation
sim_cauchy = quantile(y, prob)
rbind(std_cauchy, sim_cauchy)
```

```
##                  10%       20%        30%        40%         50%       60%
## std_cauchy -3.077684 -1.376382 -0.7265425 -0.3249197  0.00000000 0.3249197
## sim_cauchy -3.663363 -1.561969 -0.8514966 -0.4273773 -0.07995834 0.2658587
##                  70%      80%      90%
## std_cauchy 0.7265425 1.376382 3.077684
## sim_cauchy 0.6780318 1.338245 3.296913
```

We notice that the generated data has similar quantiles to standard cauchy distribution. Then we plot the data.



The plot below shows the histogram of the generated sample after burnin period.



The chain plot and associated histogram of the generated sample after discarding first 1000 observations show us there is no pattern in the data, which is what we expect.

Exercise 2 (Rizzo 9.4)

The standard Laplace distribution has target density

$$f(x) = \frac{1}{2}e^{-|x|}, \quad x \in R$$

```r
# Laplace density
f2 <- function(x) {
  0.5 * exp(-abs(x))
}

# write a function
rw.Metropolis <- function(sigma, x0, N) {
  x = numeric(N)
  x[1] = x0
  u = runif(N)
  k = 0 # number of accepted proposals
  for (i in 2:N) {
    y = rnorm(1, x[i-1], sigma) # proposal
    ratio = f2(y) / f2(x[i-1])
    accept = (u[i] <= ratio)
    x[i] = y*accept + x[i-1]*(1-accept)
    k = k + accept
  }
  return(list(x = x, k = k))
}

# Let's try with 4 different choices for sigma
N = 5000
sigma = c(0.05, 0.5, 2, 16)
x0 = 10

rw1 = rw.Metropolis(sigma[1], x0, N)
rw2 = rw.Metropolis(sigma[2], x0, N)
rw3 = rw.Metropolis(sigma[3], x0, N)
rw4 = rw.Metropolis(sigma[4], x0, N)

# how many we accepted in each case
print(c(rw1$k, rw2$k, rw3$k, rw4$k))
```

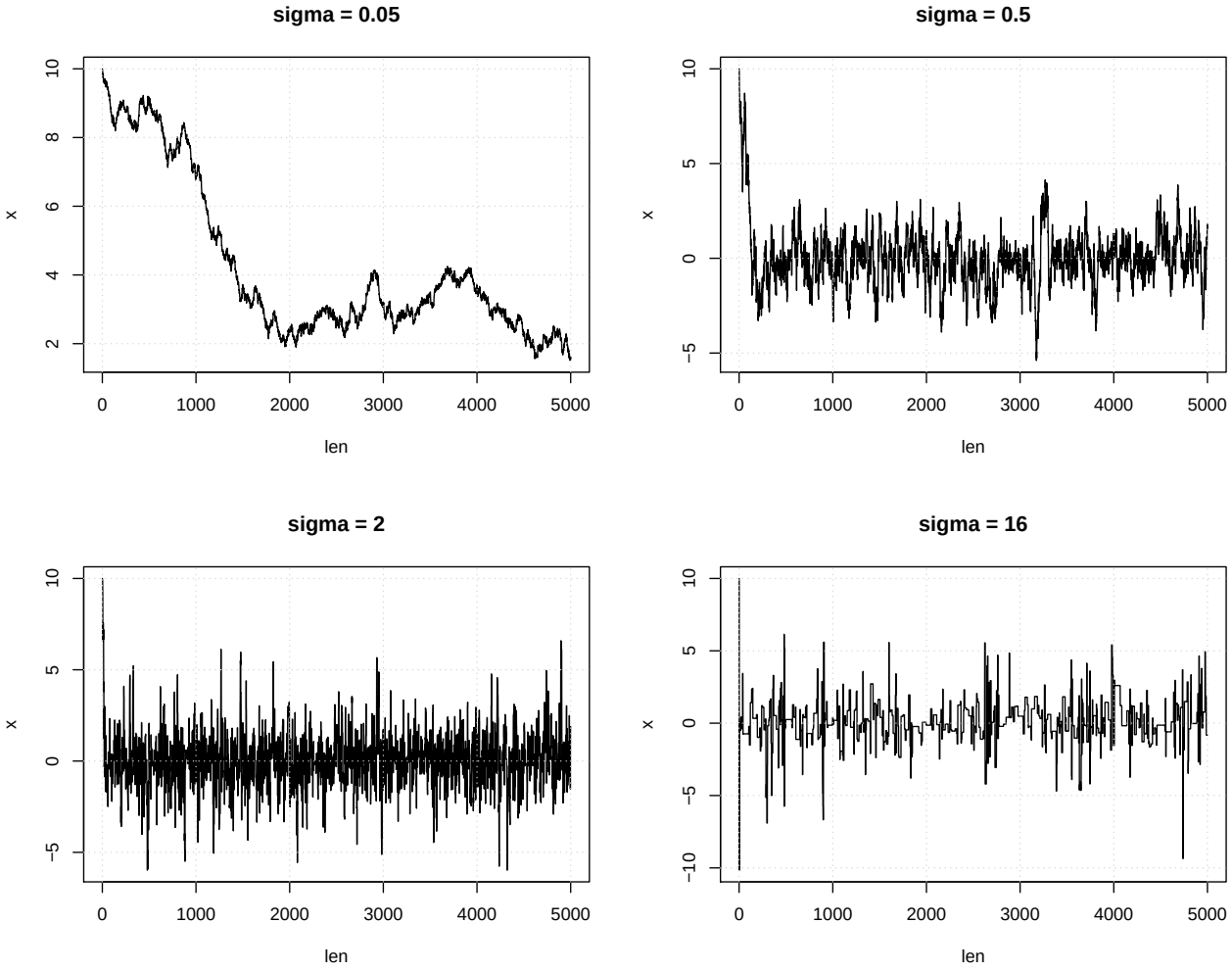
```
## [1] 4911 4141 2644  473
```

```r
print(c(rw1$k, rw2$k, rw3$k, rw4$k)/5000)
```

```
## [1] 0.9822 0.8282 0.5288 0.0946
```

The third chain has an acceptance rate around 0.5, so it might be the most efficient. Then we plot

four chains.



Now let's see how closely the draws match the quantiles of the Laplace density.

```
burnin = 500
prob = c(0.05, seq(0.1, 0.9, 0.1), 0.95)
Q = rmutil::qlaplace(prob) # Laplace quantiles
rw = cbind(rw1$x, rw2$x, rw3$x, rw4$x)
mc = rw[-burnin, ]
Qrw = apply(mc, 2, function(x) quantile(x, prob))
colnames(Qrw) = c("rw1", "rw2", "rw3", "rw4")
print(round(cbind(Q, Qrw), 3))
```

```
##          Q   rw1    rw2    rw3    rw4
## 5%  -2.303 2.029 -2.374 -2.261 -2.057
## 10% -1.609 2.208 -1.772 -1.663 -1.426
## 20% -0.916 2.493 -1.030 -0.937 -0.716
```

```
## 30% -0.511 2.750 -0.605 -0.511 -0.465
## 40% -0.223 2.942 -0.300 -0.223 -0.219
## 50%  0.000 3.190 -0.076 -0.001 -0.028
## 60%  0.223 3.614  0.144  0.280  0.315
## 70%  0.511 4.105  0.439  0.588  0.527
## 80%  0.916 7.030  0.873  0.998  1.105
## 90%  1.609 8.490  1.600  1.663  1.684
## 95%  2.303 8.846  2.410  2.426  2.400
```

We notice that the **rw3** chain has closest quantile values to theoretical values.

Exercise 3 (Rizzo 9.6)

The group sizes of 197 animals in four categories are (125, 18, 20, 34). Assume that the probabilities of the corresponding multinomial distribution are

$$(\frac{1}{2}+\frac{\theta}{4}, \frac{1-\theta}{4}, \frac{1-\theta}{4}, \frac{\theta}{4})$$

We want to estimate the posterior distribution of θ given the observed sample. We will use similar method for the Investment Model example. The likelihood function is a multinomial

$$f(x_1, x_2, x_3, x_4|\theta) = \frac{197!}{x_1!x_2!x_3!x_4!}(\frac{1}{2}+\frac{\theta}{4})^{x_1}(\frac{1-\theta}{4})^{x_2}(\frac{1-\theta}{4})^{x_3}(\frac{\theta}{4})^{x_4}$$

We use Metropolis sampler to have a target distribution equal to the posterior probability density function of θ. The given sample is $X_{obs} = (125, 18, 20, 34)$.

```
# initialize parameters
theta = 0.5 # actual value of theta
w = 0.25 # width of uniform support of proposal
m = 5000 # length of chain
sample = c(125, 18, 20, 34)
x = numeric(m)

# define a function proportional to the target density
prob <- function(y, count) {
  # compute (without the constant) the target density
  if (y < 0 || y > 1)
    return(0)
  else
    return((1/2 + y/4)^count[1] * ((1-y)/4)^count[2] * ((1-y)/4)^count[3] * (y/4)^count[4])
}
```

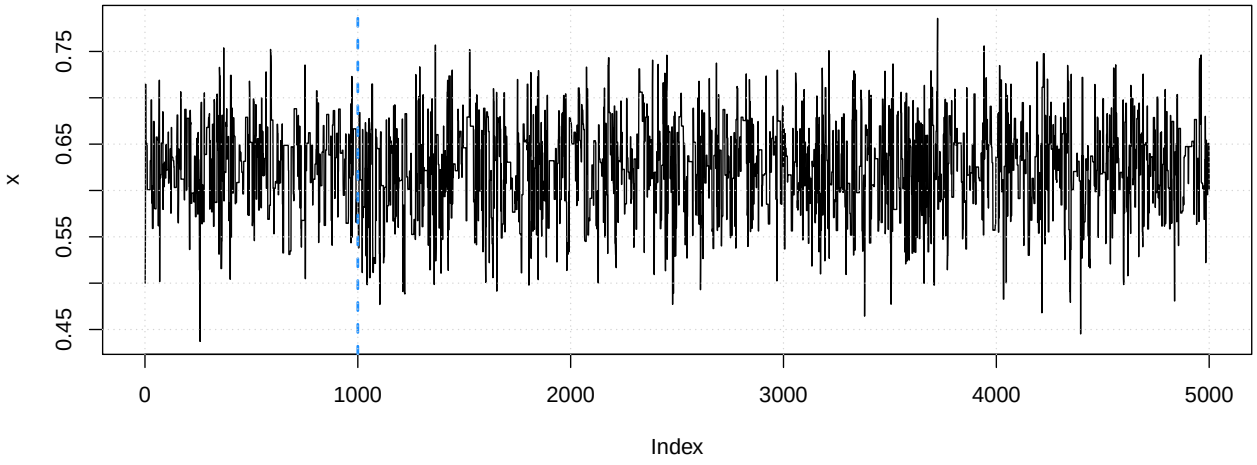
```
# set up the chain
u = runif(m)
v = runif(m, -w, w) # proposal increments
x[1] = 0.5 # initial value
for (i in 2:m) {
  y = x[i-1] + v[i]
  if (u[i] <= prob(y, sample) / prob(x[i-1], sample))
    x[i] = y
  else
    x[i] = x[i-1]
}
```

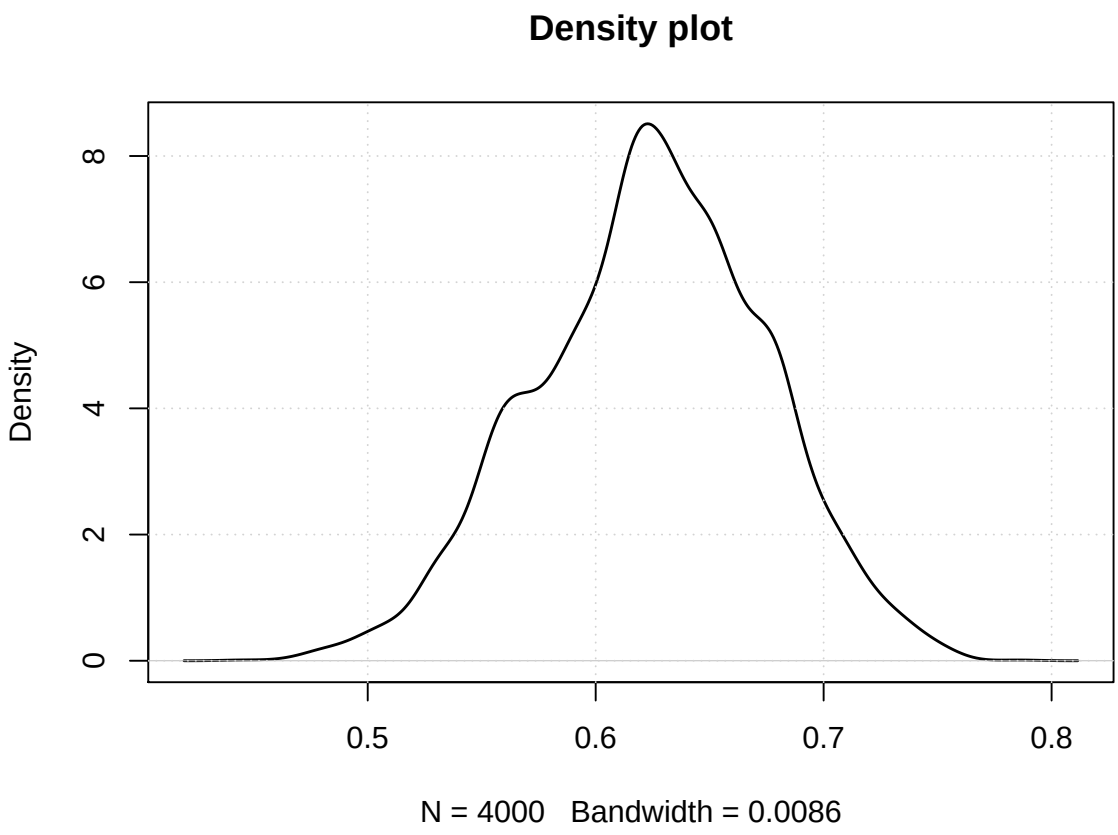
We can find the posterior mean.

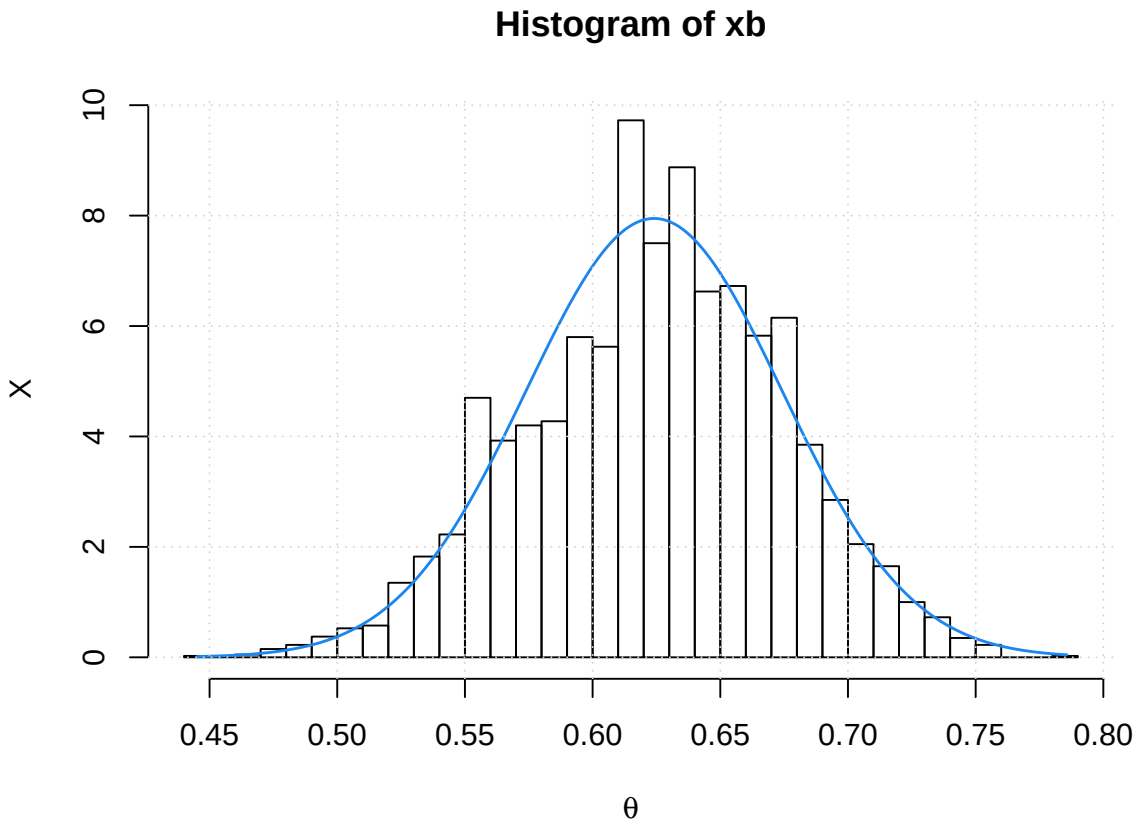
```
print(mean(x))
```

```
## [1] 0.6244409
```

Then we plot the chain and density estimate, as well as the histogram in the following cells.




Density plot
Density
0
2
4
6
8
0.5
0.6
0.7
0.8
N = 4000 Bandwidth = 0.0086


Histogram of xb
X
0
2
4
6
8
10
0.45
0.50
0.55
0.60
0.65
0.70
0.75
0.80
θ

Exercise 4 (Rizzo 9.8)

Consider the bivariate density

$$f(x,y) \propto \binom{n}{x} y^{x+a-1}\,(1-y)^{n-x+b-1}, \quad x = 0,1,\ldots,n,\ 0 \le y \le 1$$

For some fixed a, b, n, the conditional distributions are Binomial(n, y) and Beta$(x+a, n-x+b)$. We will generate a chain with target joint density `f(x,y)` using Gibbs sampler.

```
gibbs <- function(n, a, b) {
  N = 5000
  X = matrix (0, N, 2) # store chain

  # run the chain
  X[1, ] = c(floor(n/2), a/(a+b)) # initial value
  for (i in 2:N) {
    y = X[i-1, 2]
    X[i, 1] = rbinom(1, n, y)
    x = X[i, 1]
    X[i, 2] = rbeta(1, x+a, n-x+b)
  }
  chain = X[1001:N, ]
  return(chain)
}
```

The function `gibbs` returns the chain generated from the target joint density. Then we look at an example when $a = 1, b = 2, n = 100$.

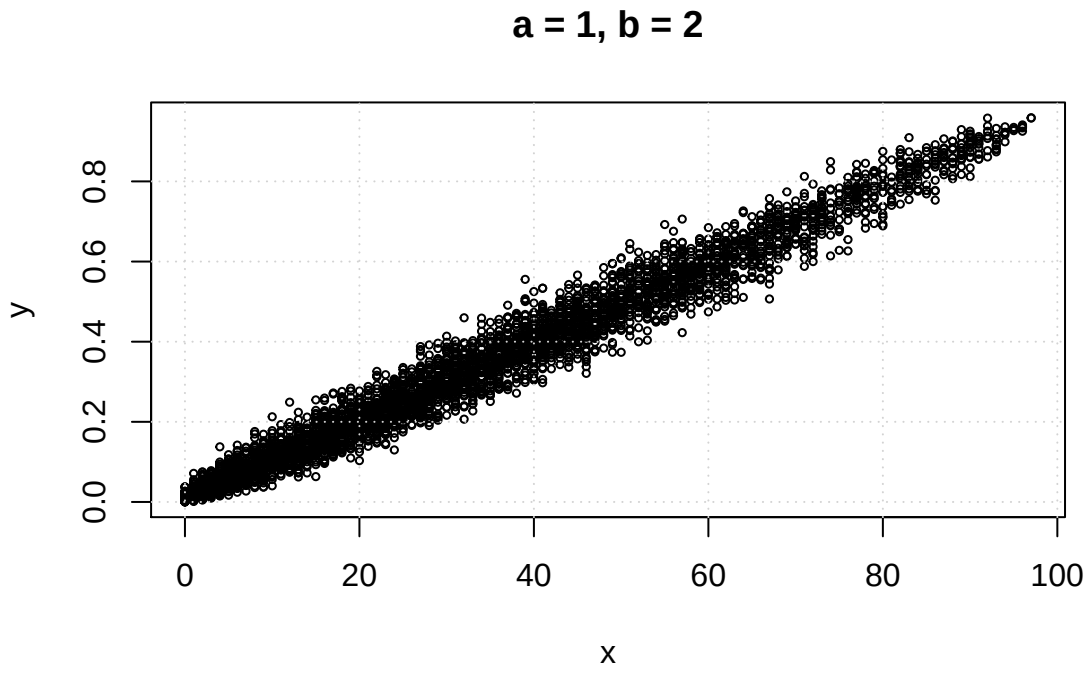
```
sample = gibbs(100, 1, 2)

# mean and variance of the chain after deleting 1000 burn-in draws
## mean
apply(sample, 2, mean)
```

```
## [1] 34.5462500  0.3442859
```

```
## sd
apply(sample, 2, sd)
```

```
## [1] 24.2243486  0.2376893
```

Then we plot the chain.



Exercise 5 (Rizzo 9.11)

Refer to Example 9.5, use Gelman-Rubin method to monitor convergence of the chain, and run the chain until the chain has converged approximately to the target distribution according to $\hat{R} < 1.2$. Also use the `coda` package to check for the convergence of the chain by the Gelman-Rubin method.

First we build up `Gelman.Rubin` function.

```
library(coda)
# define Gelman-Rubin method
Gelman.Rubin <- function(psi) {
  psi = as.matrix(psi)
  n = ncol(psi)
  k = nrow(psi)

  psi.means = rowMeans(psi) # row means
  B = n * var(psi.means) # between variance est.
  psi.w = apply(psi, 1, "var") # within variances
  W = mean(psi.w) # within variance est.
  v.hat = W*(n-1)/n + (B/n) # upper variance est.
  r.hat = v.hat/W  # Gelman-Rubin statistics
  return(r.hat)
}
```

Then we initialize some parameters and write the target density function `prob`.

```
b = 0.2 # actual value of beta
w = 0.25 # width of the uniform support set
m = 5000 # length of the chain

# generate the observed frequencies of winners
i = sample(1:5, size = 250, replace = TRUE,
           prob = c(1, 1-b, 1-2*b, 2*b, b))
win = tabulate(i)

# compute the target density
prob <- function(y, win) {
  if (y < 0||y >= 0.5)
    return(0)
    return((1/3)^win[1] * ((1-y)/3)^win[2] *
             ((1-2*y)/3)^win[3] * ((2*y)/3)^win[4] *
             (y/3)^win[5])
}
```

Since $0 \leq 1 - 2\beta \leq 1$, β is an unknown parameter with a prior distribution set to the uniform distribution on the interval $(0, 0.5)$. Here we use four initial values and do the iteration starting with $\hat{R} = 10$. We continue running the chain until $\hat{R} < 1.2$.

```
k = 4  #number of chains to generate
x = as.matrix(c(0.1, 0.2, 0.3, 0.4)) # beta should be in (0, 0.5)

invest.chain <- function(x) {
  xi = numeric(nrow(x))

  for (i in 1:length(xi)) {
    v = runif(1, -w, w) # proposal increment
    xt = x[i, ncol(x)]
    y = xt + v
    ratio = prob(y, win)/prob(xt, win)
    u = runif(1)
    if (u <= ratio) xi[i] = y else {
      xi[i] = xt  # y is rejected
    }
  }
  return(cbind(x, xi))
```

```
}

r.hat = 10
while(r.hat >= 1.2) {   #continuing the chain till it converges
  x = invest.chain(x)
  psi = t(apply(x, 1, cumsum))

  for (i in 1:nrow(psi)) {
    psi[i, ] = psi[i, ] / (1:ncol(psi))}
  r.hat = Gelman.Rubin(psi)
}

# When the chain converges
conv = ncol(x)
conv
```

```
## [1] 270
```

```
# What is the value of r.hat when it converges
r.hat
```

```
## [1] 1.199759
```

The `Gelman.Rubin` function gives the G-R convergence diagnostic statistic for the MCMC chain. We notice that our chains converge at 270, by the constraint $\hat{R} < 1.2$.

```
# Use coda package to check convergence by the G-R method
x.mcmc = mcmc.list(as.mcmc(x[1, ]), as.mcmc(x[2, ]),
                   as.mcmc(x[3, ]), as.mcmc(x[4, ]))

z_value = round(c(geweke.diag(x.mcmc)[[1]]$z, geweke.diag(x.mcmc)[[2]]$z,
            geweke.diag(x.mcmc)[[3]]$z, geweke.diag(x.mcmc)[[4]]$z), 4)

z_value
```

```
##    var1    var1    var1    var1
## -0.3101 -0.0003  1.0959 -0.7287
```

We also use `geweke.diag` function in `coda` package to verify the convergence. The test statistic is a standard Z-score: the difference between the two sample means divided by its estimated standard error. In this problem, Z-statistic values are above, all of which are greater than critical value `-1.96` and hence indicate convergence.

Then we use `geweke.plot` function to create four plots.

```
par(mfrow=c(2,2))
geweke.plot(x.mcmc, auto.layout = FALSE)
```