16.2

a

$$
\begin{aligned}
\eta_i &= \beta_0 + \beta_1 x_i \\
\beta &= \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} \\
X &= \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \\
\eta &= X\beta \\
\theta &= g^{-1}(\eta) = \text{logit}^{-1}(\eta) = \text{logit}^{-1}(X\beta) \\
y_i &\sim Binom(n_i, \theta_i)
\end{aligned}
$$

Posterior distribution is proportional to :

$$
\begin{aligned}
p(\alpha, \beta|y) &\propto \prod_i \text{logit}^{-1}(\alpha + \beta x_i)^{y_i}(1 - \text{logit}^{-1}(\alpha + \beta x_i))^{n_i - y_i} \\
&\propto \prod_i (\frac{e^{\alpha+\beta x_i}}{1 + e^{\alpha+\beta x_i}})^{y_i} (\frac{1}{1 + e^{\alpha+\beta x_i}})^{n_i - y_i} \\
&\propto \prod_i e^{y_i(\alpha+\beta x_i)}(1 + e^{\alpha+\beta x_i})^{-n_i} \\
\log(p(\alpha, \beta|y) &= \sum_i y_i(\alpha + \beta x_i) - n_i \log(1 + e^{\alpha+\beta x_i}) + C
\end{aligned}
$$

```
xs <- c(-0.86, -0.30, -0.05, 0.73)
ys <- c(0, 1, 3, 5)
ns <- c(5, 5, 5, 5)

LogConditionalProb <- function(a, b) {
    eta <- a+b*xs
    llik <- ys*(eta)-ns*log(1+exp(eta))
    return(sum(llik))
}

ProposalAlpha <- function(x) rnorm(1, x, 1.8)
ProposalBeta <- function(x) rnorm(1, x, 10)


GibbsMetropolis <- function(ProposalAlpha, ProposalBeta, LogProb, N, start.values=c(0.5, 7
{
    alpha <- start.values[1]
    beta <- start.values[2]

    alphas <- numeric(2*N)
    betas <- numeric(2*N)

    acc.alpha <- numeric(2*N)
```

```
    acc.beta <- numeric(2*N)

    for(i in 1:(2*N))
    {
        alpha.star <- ProposalAlpha(alpha)
        beta.star <- ProposalBeta(beta)

        pr.alpha <- min(1, exp(LogProb(alpha.star, beta)-LogProb(alpha, beta)))
        alpha <- sample(c(alpha.star, alpha), size=1, prob=c(pr.alpha, 1-pr.alpha))

        pr.beta <- min(1, exp(LogProb(alpha, beta.star)-LogProb(alpha, beta)))
        beta <- sample(c(beta.star, beta), size=1, prob=c(pr.beta, 1-pr.beta))

        alphas[i] <- alpha
        betas[i] <- beta
        acc.alpha[i] <- pr.alpha
        acc.beta[i] <- pr.beta
    }

    print("Alpha Acc:")
    print(mean(acc.alpha))

    print("Beta Acc:")
    print(mean(acc.beta))

    m <- matrix(c(alphas[(N+1):(2*N)], betas[(N+1):(2*N)]), 2, N, byrow=T)
    return(m)
}

samples1 <- GibbsMetropolis(ProposalAlpha, ProposalBeta, LogConditionalProb, 10000, c(0, 1
samples2 <- GibbsMetropolis(ProposalAlpha, ProposalBeta, LogConditionalProb, 10000, c(2, 1

alphas = c(samples1[1, ], samples2[1,])
betas = c(samples1[2, ], samples2[2, ])

SummaryStats <- function(name, samples) {
    print(name)
    print(mean(samples))
    print(quantile(samples, c(0.025, 0.975)))
}

SummaryStats("alpha", alphas)
SummaryStats("beta", betas)

LD50 <- -alphas/betas
SummaryStats("LD50", LD50)

# > source("exercise162.r")
# [1] "Alpha Acc:"
# [1] 0.4066657
# [1] "Beta Acc:"
# [1] 0.2535952
# [1] "Alpha Acc:"
# [1] 0.4101632
```

```
# [1] "Beta Acc:"
# [1] 0.2534394
# [1] "alpha"
# [1] 0.7755103
#       2.5%       97.5%
# -0.7453023   2.5547396
# [1] "beta"
# [1] 8.444438
#       2.5%       97.5%
#  2.865092  17.223470
# [1] "LD50"
# [1] -0.08526614
#       2.5%       97.5%
# -0.2765251   0.1540025
```

17

a

Draw x_a and x_b from $f(x)$ and label such that

$$f(x_a)q(x_a|x_b) \leq f(x_b)q(x_b|x_a)$$

$$\begin{aligned}
p(x^{t-1} = x_a, x^t = x_b) &= f(x_a)q(x_b|x_a) \\
p(x^{t-1} = x_b, x^t = x_a) &= f(x_b)q(x_a|x_b)\frac{q(x_b|x_a)}{q(x_a|x_b)}\frac{f(x_a)}{f(x_b)} \\
p(x^{t-1} = x_b, x^t = x_a) &= f(x_a)q(x_b|x_a)
\end{aligned}$$

b

In order to be practically effective:

- The 'steps' in the proposal distribution must not be too small compared to the parameter space
- The acceptance ratio must not be too small

c

Where $q(x|y) = q(y|x)$ the proposal distribution is symmetric and we can use Metropolis
Where $q(x|y) = q_0(x)$, the proposals are sampled from the same distribution. When q_0 is very different than f, the acceptance rate is low and convergence slow.

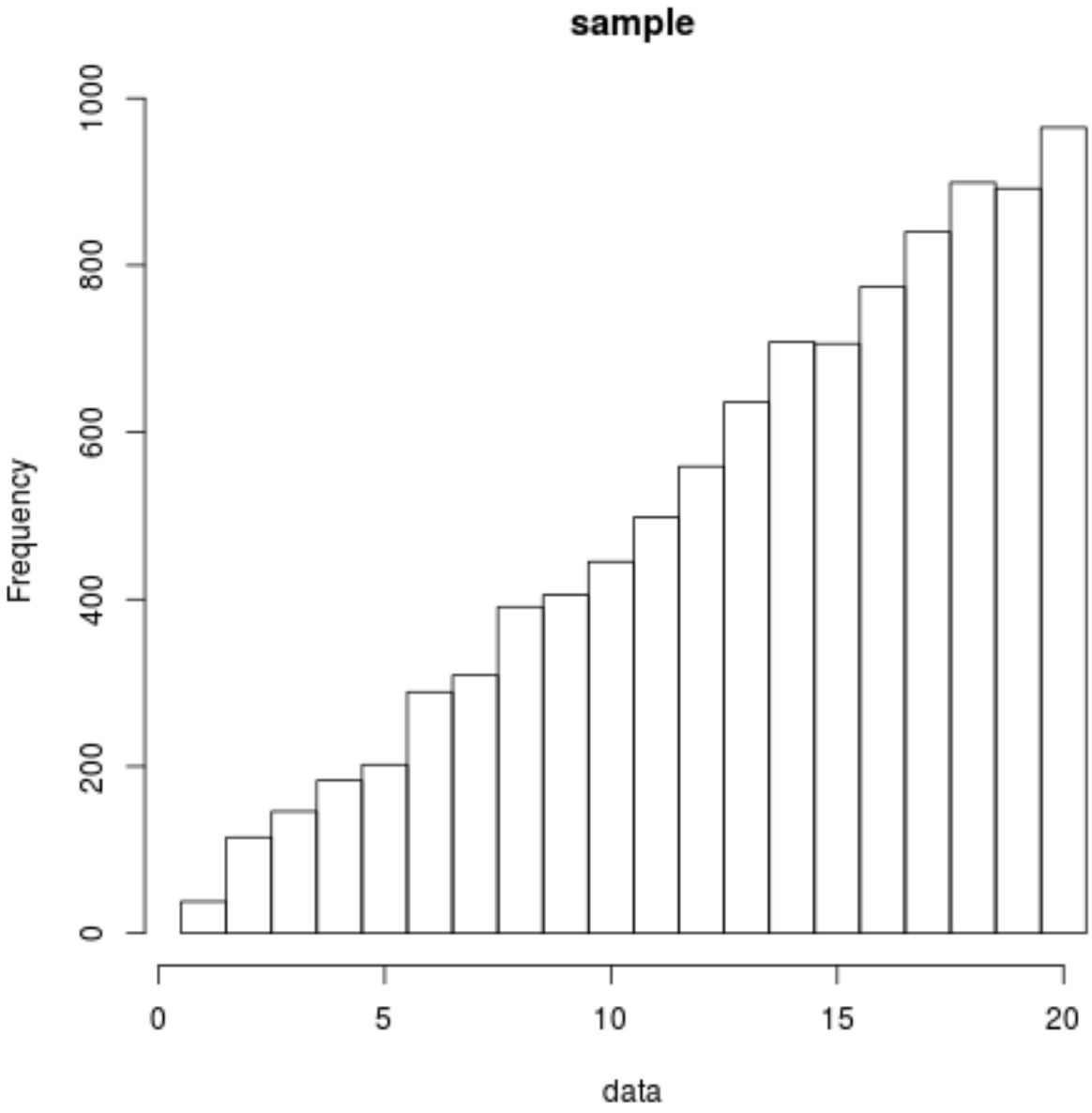
d

```
RandomU <- function(x) runif(1, x/3, x*3)
LogDistU <- function(y, x) log(dunif(y, x/3, x*3))
LogDistG <- function(x) log(dgamma(x, shape=3/2, rate=1))

MetropolisHastings <- function(RandomQ, LogDistQ, LogF, N, x=1.5){
    xs <- numeric(2*N)
    acc <- numeric(2*N)
```

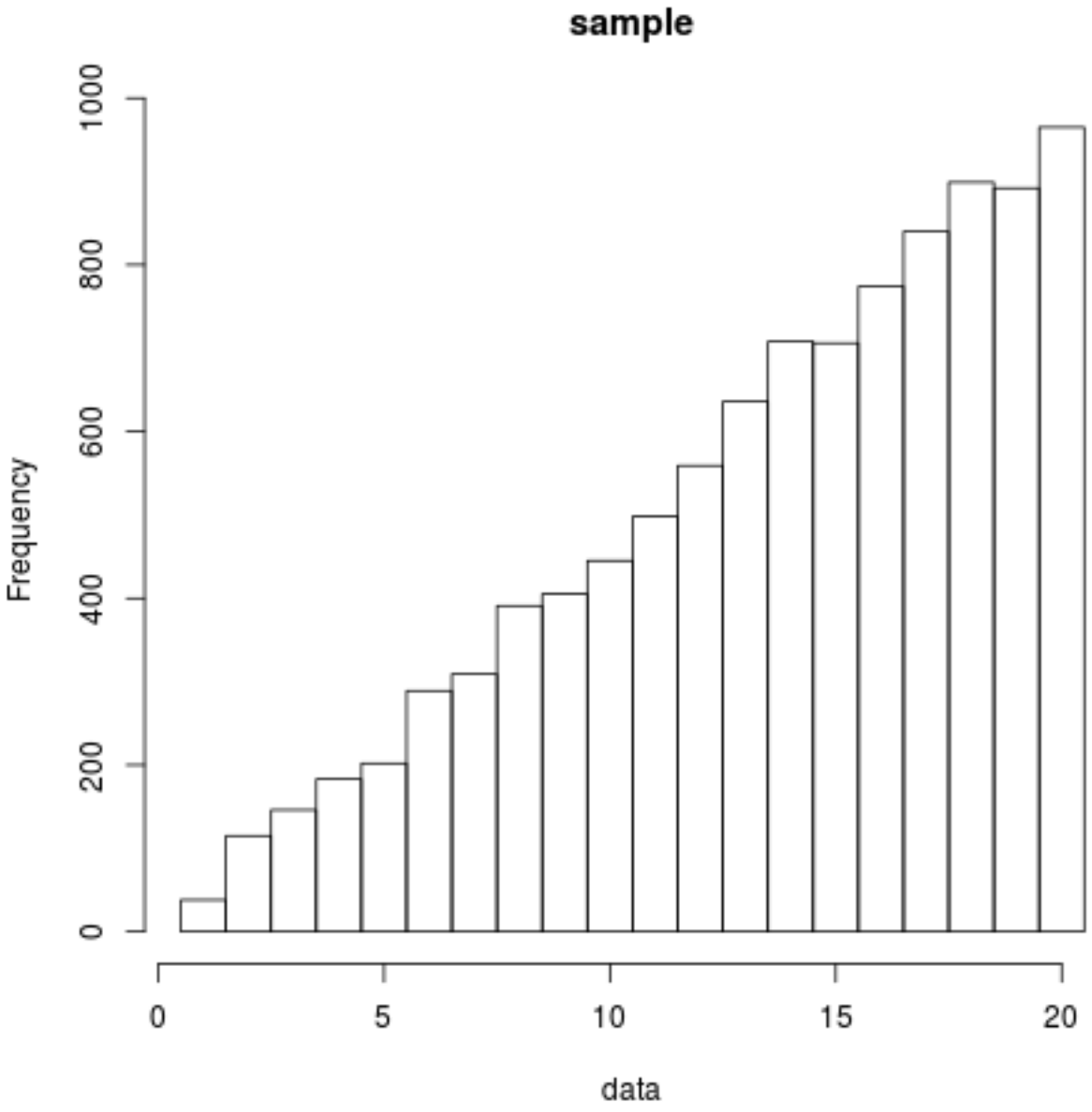
```
    for(i in 1:(2*N)) {
        y <- RandomQ(x)
        pr <- min(1, exp(LogDistQ(x, y)+LogF(y)-LogDistQ(y, x)-LogF(x)))
        x <- sample(c(y, x), size=1, prob = c(pr, 1-pr))
        xs[i] = x
        acc[i] = pr
    }
    print(mean(acc))
    return(xs[(N+1):(2*N)])
}



 xs = MetropolisHastings(RandomU, LogDistU, LogDistG, 10000)

 print("Mean")
 print(mean(xs))
 print("Var")
 print(var(xs))

# source("nils17.r")
# [1] "Mean"
# [1] 1.511603
# [1] "Var"
# [1] 1.517227
```

18

a




```
        Chi-squared test for given probabilities

data:  ns
X-squared = 14.3076, df = 19, p-value = 0.7655
```

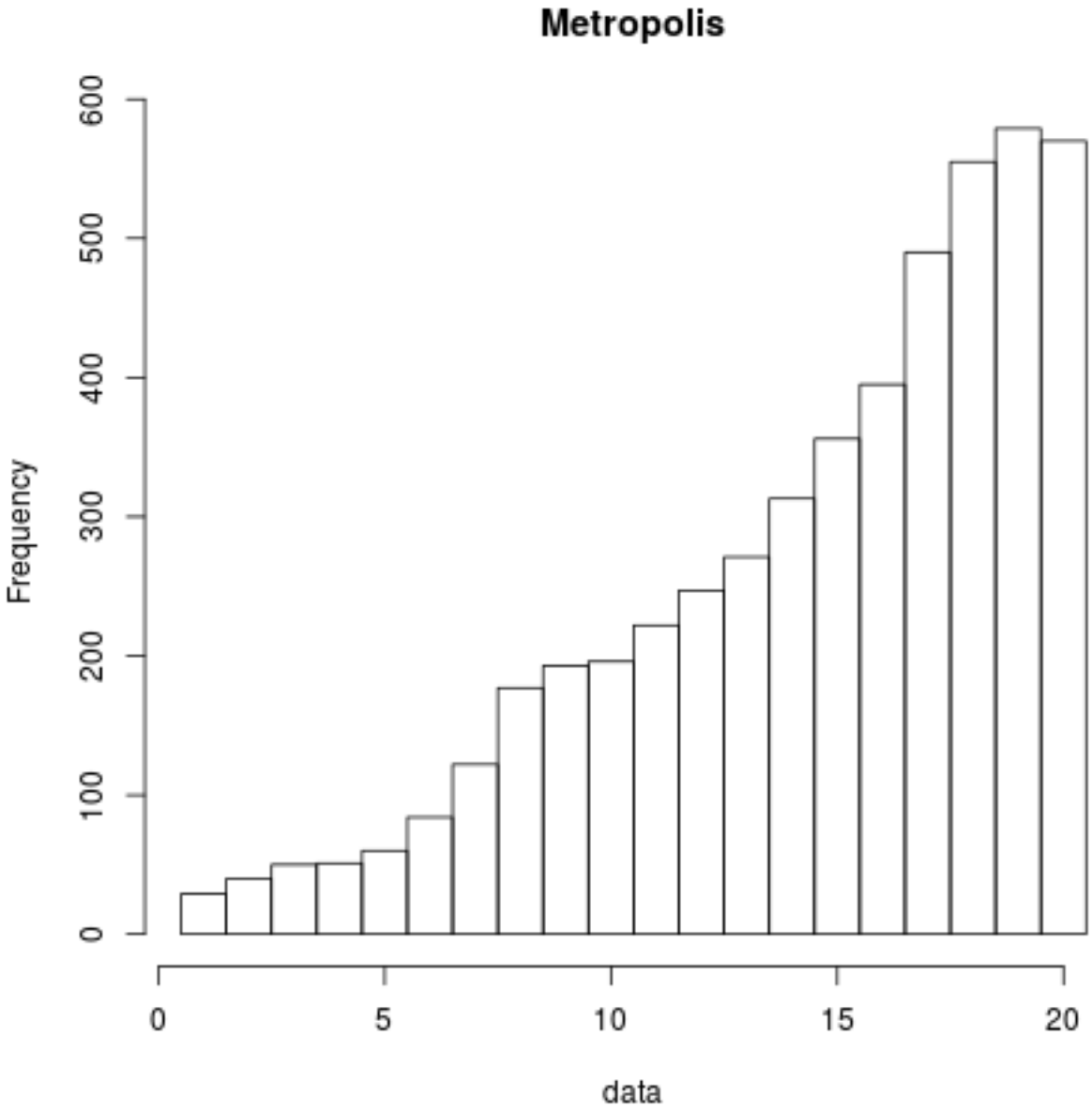


b

c

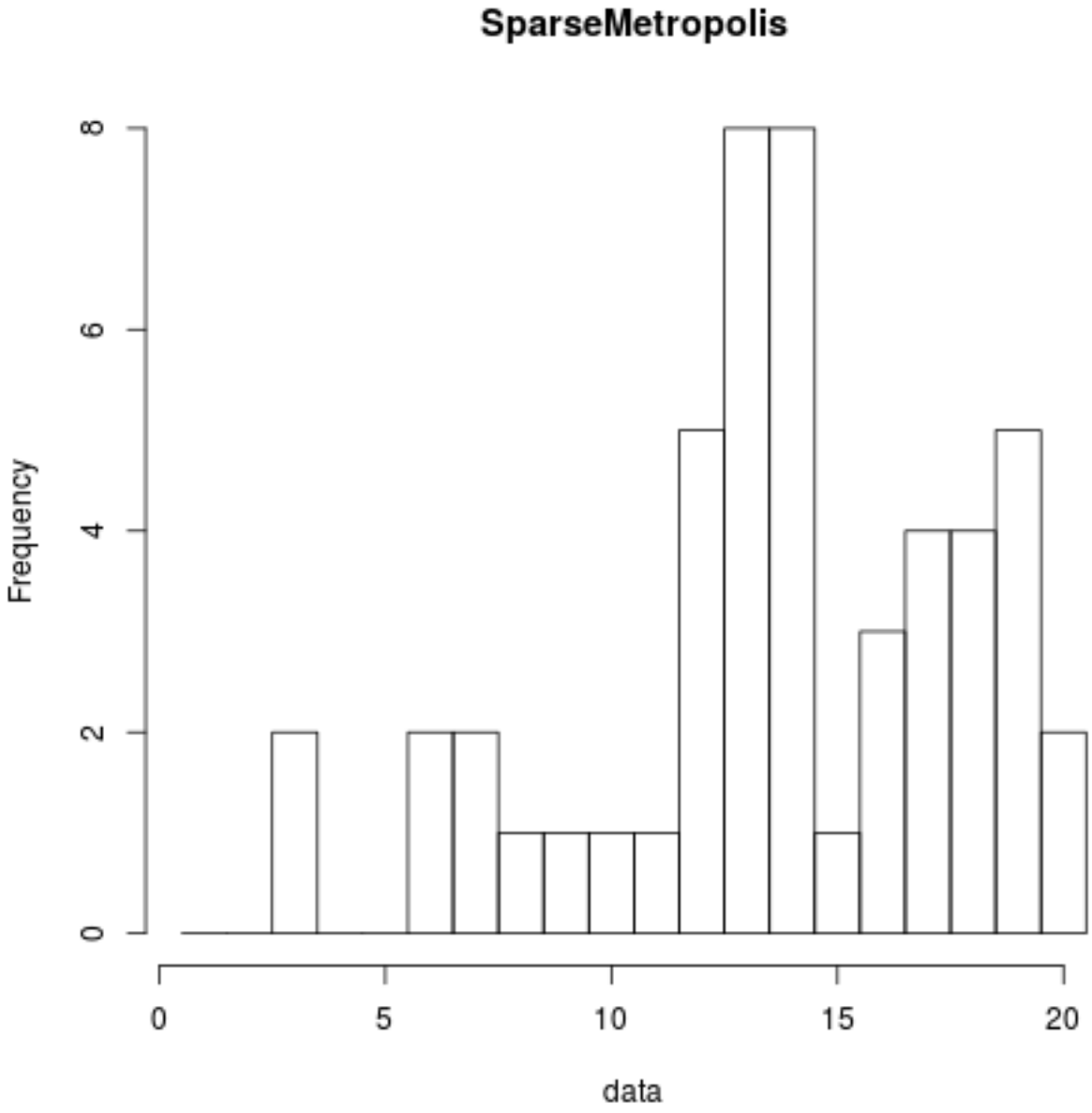
Fermi/Teller

d




```
[1] "Metropolis"

        Chi-squared test for given probabilities

data:  ns
X-squared = 61.6538, df = 19, p-value = 2.113e-06
```

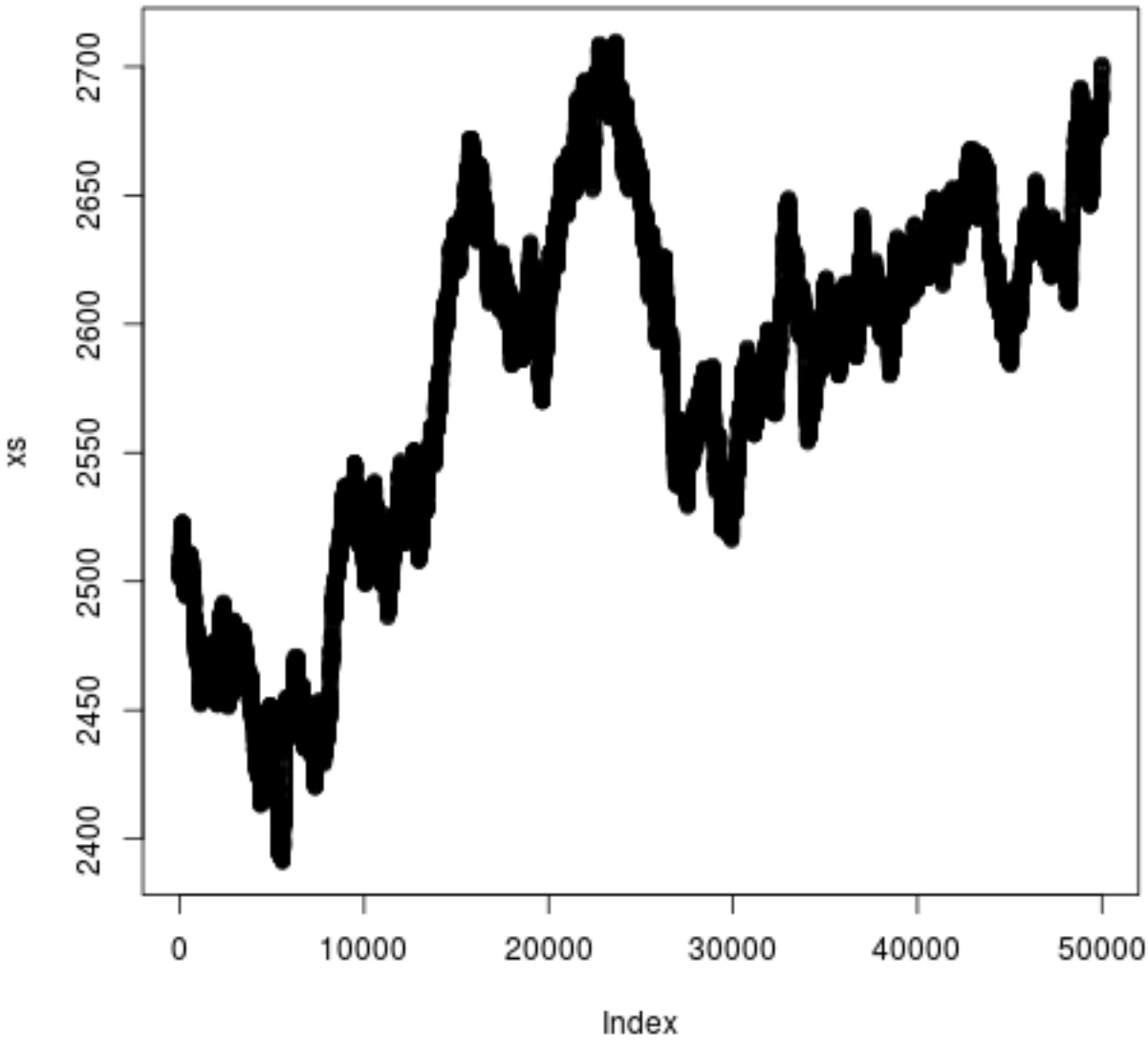
e

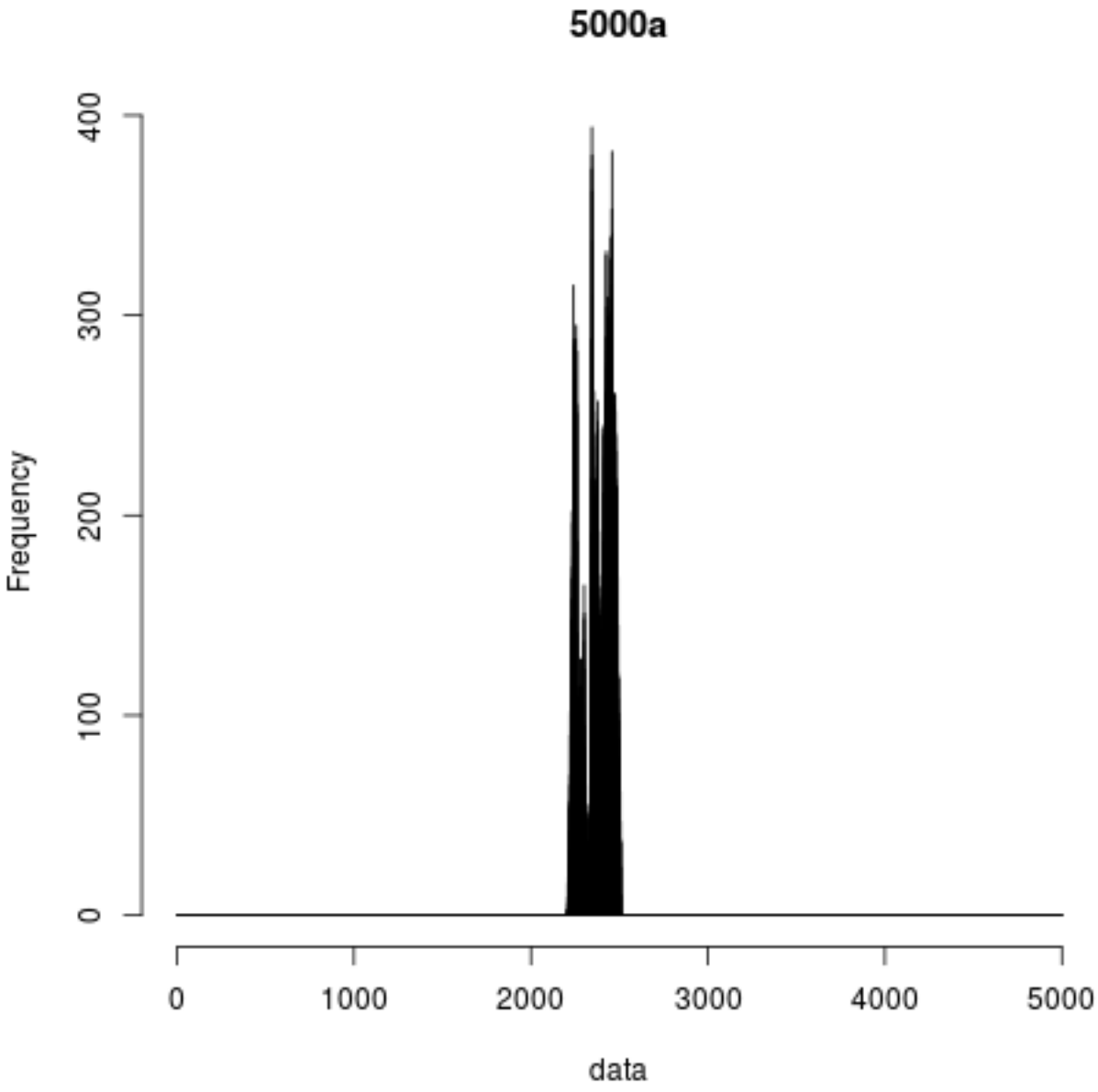



```
[1] "SparseMetropolis"

        Chi-squared test for given probabilities

data:  ns
X-squared = 27.9155, df = 19, p-value = 0.08507
```

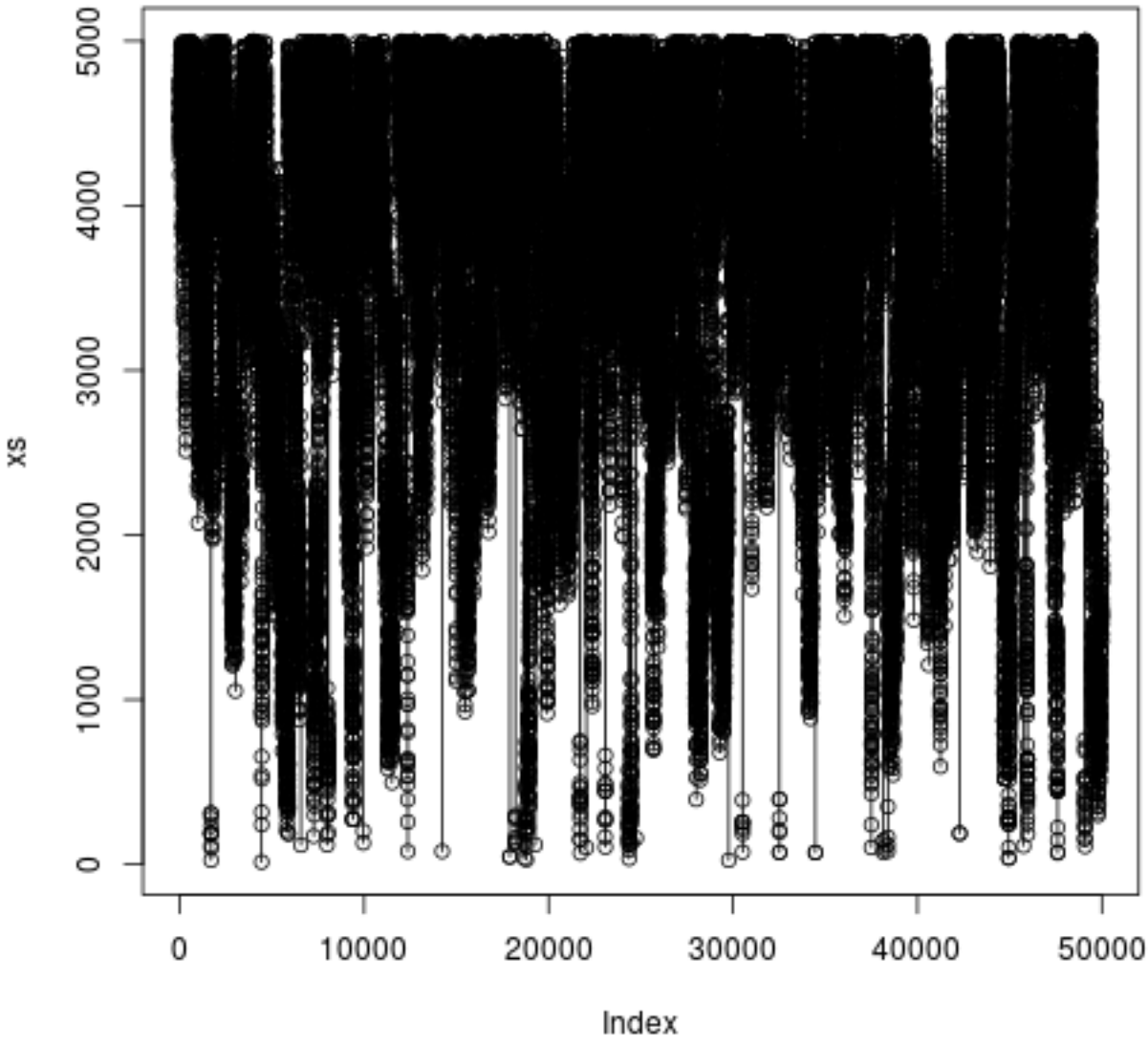
f

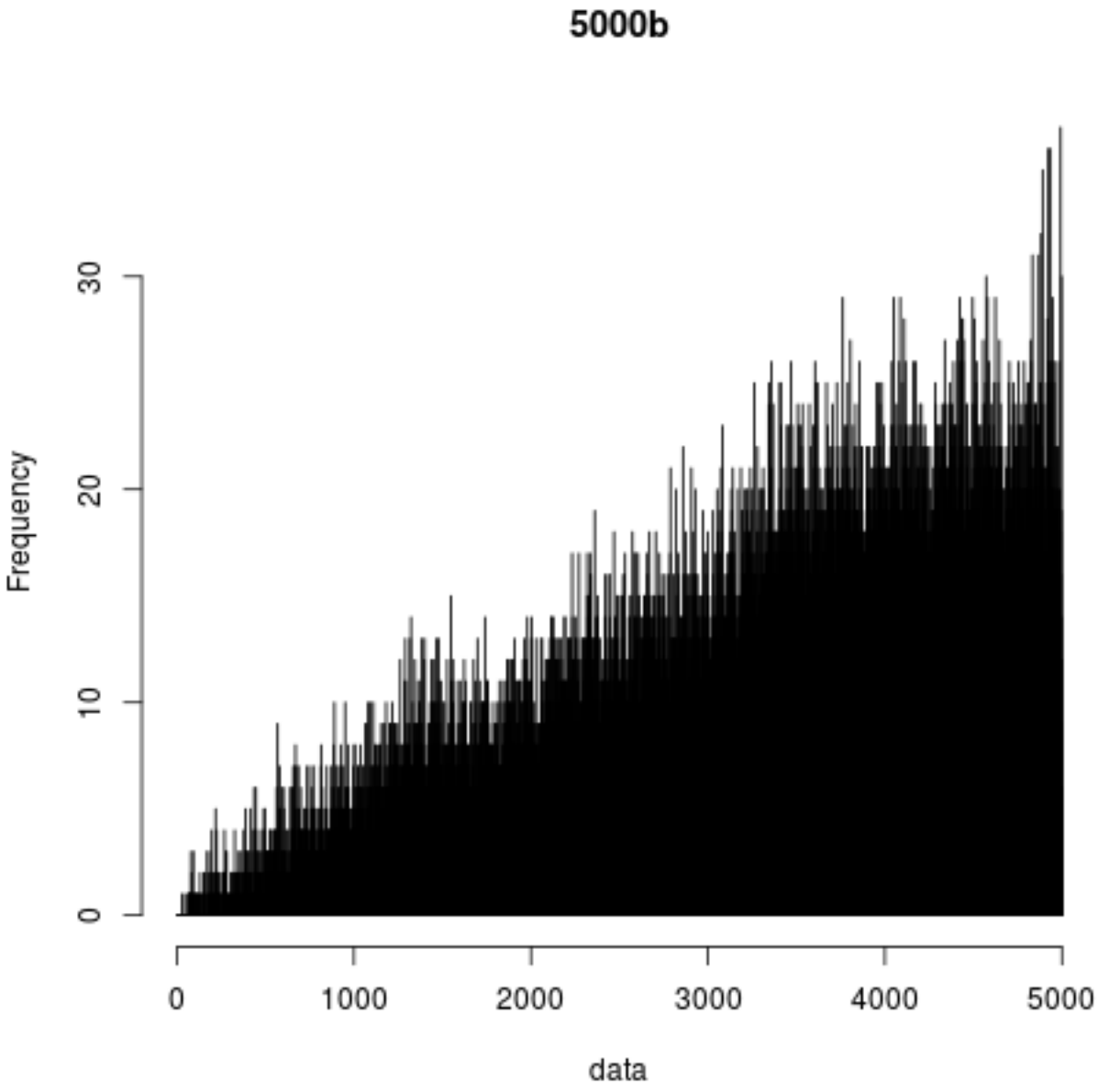
When M is large, the chain moves too slowly. Could use a larger uniform proposal function.


xs
2400
2450
2500
2550
2600
2650
2700
0
10000
20000
30000
40000
50000
Index

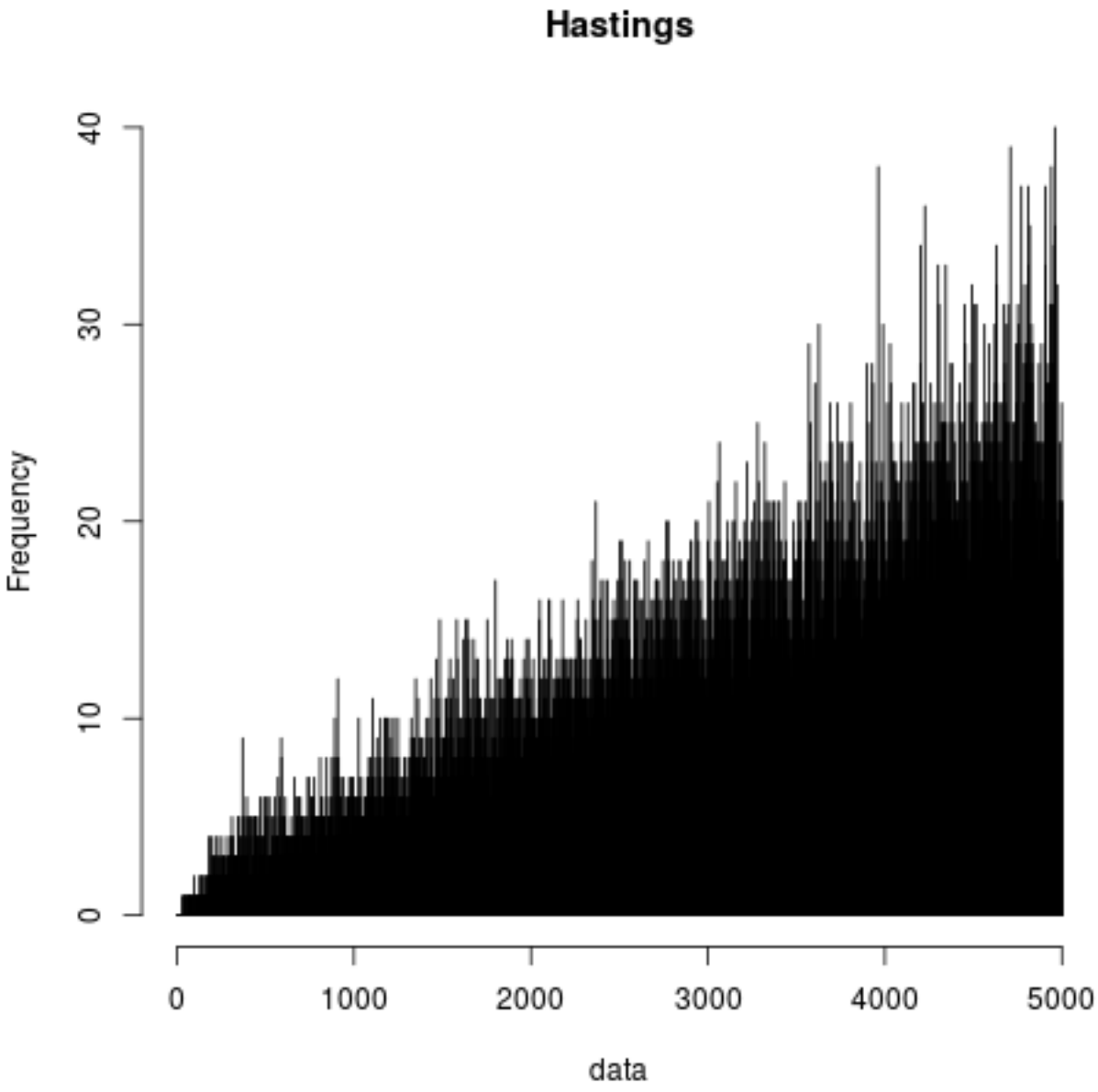


Using a larger window


xs
0
1000
2000
3000
4000
5000
0
10000
20000
30000
40000
50000
Index



Using asymmetric proposal function




```
library(plyr)

testfit <- function(data, M, name="test") {
    js = 1:M
    T = M*(M+1)/2
    ps = js/T
    counts = count(data)
    ns = numeric(M)
    ns[counts$x] = counts$freq
    # png(paste(name, "hist.png", sep = ""))
    hist(data, 0.5:(M+0.5), main=name)
    # dev.off()
```

```
    print(name)
    print(chisq.test(ns, p=ps))
    return(ns)
}

suba <- function() {
    M = 20
    js = 1:M
    T = M*(M+1)/2
    ps = js/T
    s = sample(js, 10000, replace=T, prob=ps)
    testfit(s, M, "sample")
}

Metropolis <- function(randomQ, f, N, M=20) {
    x <- M/2
    xs <- numeric(2*N)
    for(i in 1:(2*N)) {
        y <- randomQ(x, M)
        pr <- min(1, f(y)/f(x))
        x <- sample(c(y, x), size=1, prob = c(pr, 1-pr))
        xs[i] <- x
    }
    return(xs[(N+1):(2*N)])
}

rqD <- function(x, M=20) sample(c((x+1) %% M, (x-1) %% M), size=1, prob=c(0.5, 0.5))

fD <- function(x) x+1

subd <- function()
{
    N = 50000
    xs = Metropolis(rqD, fD, N)+1
    testfit(xs, 20, "Metropolis")
}

sube <- function()
{
    N = 5000
    xs = Metropolis(rqD, fD, N)+1
    pruned = xs[(1:(N/100))*100]
    testfit(pruned, 20, "SparseMetropolis")
}

subf <- function() {
    N = 50000
    M = 5000
    xs = Metropolis(rqD, fD, N, M)+1
    png("5000plot.png")
    plot(xs, type="o")
    dev.off()
    testfit(xs, M, "5000a")
```

```
    new.proposal <- function(x, M) {
        v = sample((-M/20):(M/20),1)
        return((x+v) %% M)
    }
    xs = Metropolis(new.proposal, fD, N, M)+1
    #png("5000plot2.png")
    plot(xs, type="o")
    #dev.off()
    testfit(xs, M, "5000b")

    maxLimit = 5000
    asymmetric.proposal <- function(x) {
        min.level <- floor(x/2)
        max.level <- min(maxLimit-1, x*2)
        v = sample(min.level:max.level, 1)
        return(v)
    }
    asymmetric.density <- function(y, x) {
        min.level <- floor(x/2)
        max.level <- min(maxLimit-1, x*2)
        return(log(1/(max.level-min.level+1)))
    }
    prob.func <- function(x) log(x+1)
    source("nils17.r")
    xs <- MetropolisHastings(
        asymmetric.proposal,
        asymmetric.density,
        prob.func,
        N,
        x=floor(M/2))
    testfit(xs, 5000, "Hastings")
}

#suba()
# subd()
#sube()
subf()
```