# Bayesian analysis

*Daniel Buscombe*

## Bayesian Thinking

Bayesian inference is an extremely powerful set of tools for modeling any random variable

We provide our understanding of a problem and some data, and in return get a quantitative measure of how certain we are of a particular fact. This approach to modeling uncertainty is particularly useful when:

1. Data is limited
2. We're worried about overfitting (our model is excessively complex)
3. We have reason to believe that some facts are more likely than others, but that information is not contained in the data we model on
4. We're interested in precisely knowing how likely certain facts are, as opposed to just picking the most likely fact

For two events $A$ and $B$, the conditional probability of $A$ given $B$ is

$$Pr(A|B) = \frac{Pr(A \cap B)}{Pr(B)}$$

where $A \cap B$ means "the intersection of $A$ and $B$", or, that $A$ and $B$ both occur. Let $A^c$ be the complement of $A$, that is, of outcomes that are not $A$.

Baye's theorem allows us to compute $Pr(A|B)$ from $Pr(B|A)$, $Pr(B|A^c)$ and $Pr(A)$ via

$$Pr(A|B) = \frac{Pr(B|A)Pr(A)}{Pr(B|A)Pr(A) + Pr(B|A^c)Pr(A^c)}$$

From the definition of conditional probability, the numerator is $Pr(B|A)Pr(A) = Pr(A \cap B)$

and the denominator is

$Pr(B|A)Pr(A) + Pr(B|A^c)Pr(A^c)$

$= Pr(A \cap B) + P(A^c \cap B)$

$= Pr(A \cap B)$ or $Pr(A^c \cap B)$

$= Pr(B)$

so Baye's theorem is usually written (as long as $Pr(B) > 0$)

$$Pr(A|B) = \frac{Pr(B|A)Pr(A)}{Pr(B)}$$

Think of $A$ as some proposition about the world, and $B$ as some data or evidence.

## It's ok to be uncertain about what is true ...

### because we can use data as evidence that certain facts are more likely than others

$A$ represents the proposition that it rained today, and $B$ represents the evidence that the grass outside is wet:

$$Pr(rain|wet\ grass) = \frac{Pr(wet\ grass|rain)Pr(rain)}{Pr(wet\ grass)}$$

Let's break this down.

$Pr(rain|wet\ grass)$ asks, "What is the probability that it rained given that there is wet grass outside?"

Before looking at the ground, what is the probability that it rained, $Pr(rain)$? Think of this as the plausibility of an assumption about the world.

We then ask how likely the observation that the grass is wet outside is under that assumption, $Pr(wet\ grass|rain)$?

This updates our initial beliefs about the proposition (that it rained today) with some observation (that the grass is wet).

This is *Bayesian inference*, where our initial beliefs are represented by the **prior distribution** ($Pr(rain)$), and our final beliefs are represented by the **posterior distribution** $Pr(rain|wet\ grass)$.

The denominator simply asks, "What is the total plausibility of the evidence?", whereby we have to consider all assumptions to ensure that the posterior is a proper probability distribution.

## Frequentists versus Bayesians

### Frequentist

Probability only has meaning in terms of a limiting case of repeated measurements.

You are standing in a wind tunnel and you measure wind speed, then measure it again, and again, and again, each time the result will be slightly different due to the statistical error of the measuring device. In the limit of many measurements, the frequency of any given value indicates the probability of measuring that value.

For frequentists, probabilities are fundamentally related to frequencies of events. This means, for example, that in a strict frequentist view, it is meaningless to talk about the probability of the true wind speed: the true wind speed is, by definition, a single fixed value, and to talk about an extended frequency distribution for a fixed value is nonsense.

The frequentist approach evaluates procedures based on sampling from a particular model (the likelihood) repeatedly. The likelihood defines the distribution of the observed data conditional on the unknown parameter(s).

- A parameter is fixed but an unknown constant.
- The probabilities are always interpreted as long-run relative frequencies.
- Statistical procedures are judged by how well they perform in the long-run over some infinite number of repetitions of the experiment

### Bayesian

For Bayesians, the concept of probability is extended to cover degrees of certainty about statements. A Bayesian might claim to know the wind speed, $U$ with some probability $Pr(U)$: that probability can certainly be estimated from frequencies in the limit of a large number of repeated experiments, but this is not fundamental

For Bayesians, probabilities are fundamentally related to their own knowledge about an event. This means, for example, that in a Bayesian view, we can meaningfully talk about the probability that the true wind speed lies in a given range. That probability codifies our knowledge of the value based on prior information and available data.

The Bayesian approach requires sampling a model (likelihood) and also knowing a prior distribution on all unknown parameters in the model. In the easiest case, where we only have one unknown parameter, the likelihood and prior are combined in such a way that we compute the distribution of the unknown parameter given the data (posterior distribution).

- A parameter is considered to be a random variable and has a distribution.
- The prior distribution placed on the unknown parameter quantifies our beliefs regarding the unknown parameter.
- We use the laws of probability to make inferences about the unknown parameter of interest.
- We update our beliefs about the unknown parameter after getting data (likelihood). This yields the posterior distribution which reweights things according to the prior distribution and the data (likelihood).

## Simple Bayesian Inference

Doctors recommend eight hours of sleep for an average adult. What proportion of American college students get at least eight hours of sleep?

Let $p$ be the (unknown) proportion of students who sleep at least 8 hours. In the Bayesian viewpoint, your beliefs about the uncertainty in this proportion are represented by a **prior probability distribution** placed on this parameter. This distribution reflects your subjective prior opinion about plausible values of $p$.

Based on reading articles on the topic, we think that college students generally get less than eight hours of sleep and so $p$ is likely smaller than 0.5. Our best guess at the value of $p$ is 0.3, but we think it is plausible that this proportion could be any value in the interval from 0 to 0.5

Our prior density for $p$ is $g(p)$. A success is sleeping 8 hours or more. A random sample of $s$ successes and $f$ failures has a likelihood function

$L(p) \propto p^s(1-p)^f$

the posterior density for $p$ by multiplying the prior density by the likelihood

Posterior $\propto$ Likelihood $\times$ Prior

$g(p|data) \propto g(p)L(p)$

Posterior = Prior $\times$ evidence / constant

We'll calculate posterior distribution calculations using 2 different choices of the prior density $g$

### 1. Discrete prior

Let's start with a list of plausible proportions

```
p = seq(0.05, 0.95, by = 0.1)
```

assign weights, and then normalize to probabilities

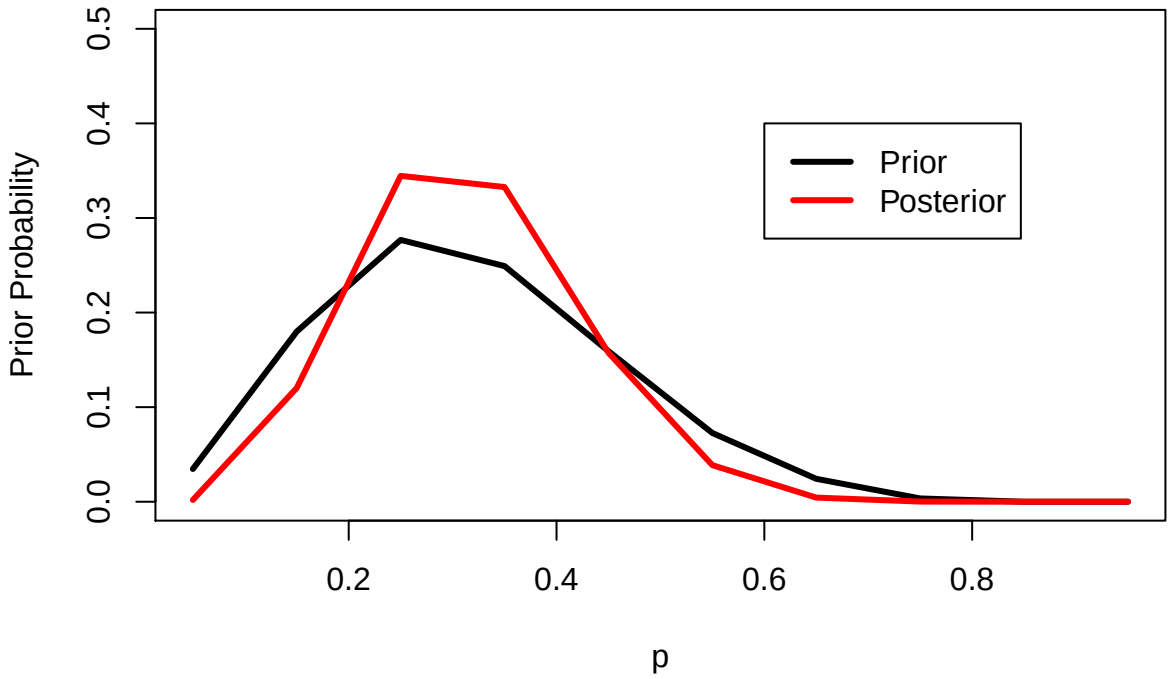
```
prior = c(1, 5.2, 8, 7.2, 4.6, 2.1, 0.7, 0.1, 0, 0)
prior = prior/sum(prior)
```

How many of you had at least 8 hours sleep last night?

Based on the prior information and these observed data, we will estimate the proportion $p$.

```
s = 5 #number of successes
f = 12 #number of failures
data = c(s, f)

post = p^s * (1-p)^f
post = post/sum(post)
```

```
plot(p, prior, type = "l", ylab="Prior Probability", ylim=c(0,0.5), lwd=3)
lines(p, post, col='red', lwd=3)
legend(.6, .4, c("Prior", "Posterior"), col=c("black","red"), lwd=3)
```




The R function *pdisc* in the package LearnBayes computes the posterior probabilities. Inputs the vector of proportion values $p$, the vector of prior probabilities prior, and a data vector data consisting of $s$ and $f$. The output of *pdisc* is a vector of posterior probabilities

```
#install.packages("LearnBayes")
library(LearnBayes)
post = pdisc(p, prior, data)
```

### 2. Beta prior

Since $p$ is a continuous parameter, we can construct density $g(p)$

Suppose we think that $p$ is equally likely to be more than or less than 0.3, but we're 90% confident that $p <$ .9. We use the beta distribution

$g(p) \propto p^{a-1}(1-p)^{b-1}$

we obtain hyperparameters $a$ and $b$ indirectly through statements about the percentiles of the distribution

We need to solve the following equations:

$\int_0^{0.3} \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \theta^{a-1}(1-\theta)^{b-1} d\theta = 0.5$

and

$\int_0^{0.5} \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \theta^{a-1}(1-\theta)^{b-1} d\theta = 0.9$

```
quantile2=list(p=.9,x=.5) #90th percentile is .5
quantile1=list(p=.5,x=.3) #50th percentile is .3
```

use the **beta.select** function in LearnBayes

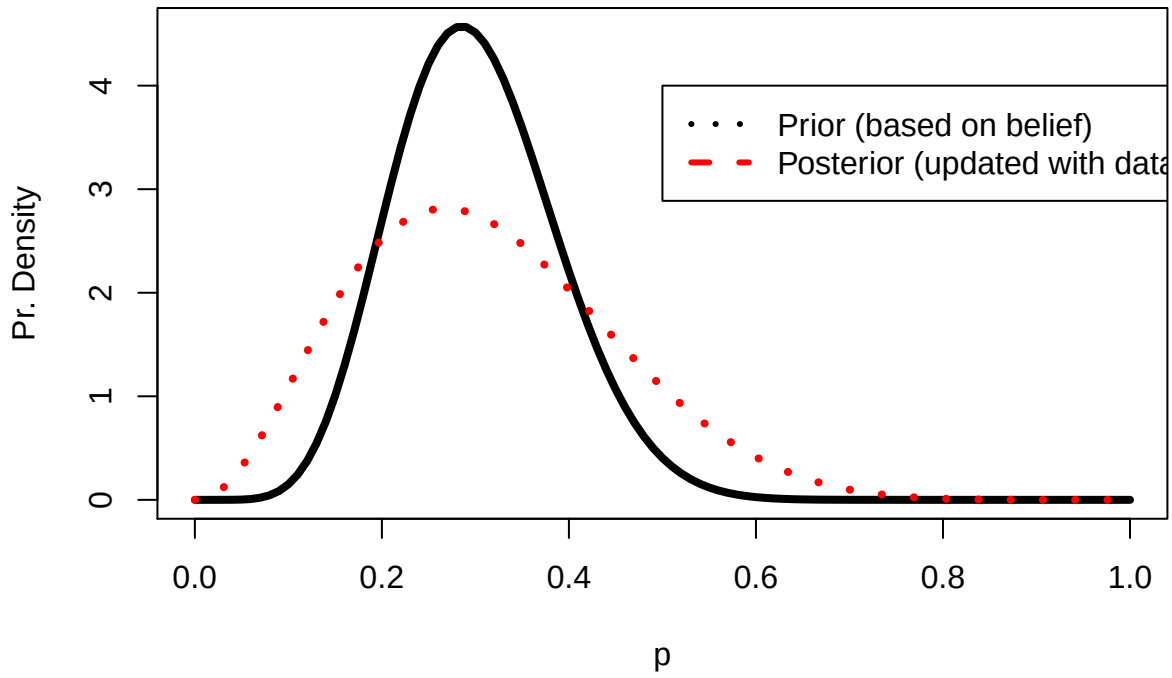
```
beta.select(quantile1,quantile2)
```

```
## [1] 3.26 7.19
```

we see that the prior information is matched with a beta density with $a = 3.26$ and $b = 7.19$. Combining this beta prior with the likelihood function, we say

$g(p|data) \propto p^{a+s-1}(1-p)^{b+f-1}$

```
a  = 3.26
b = 7.19

# use dbeta as the density of the beta distribution with shape parameters a+s and b+f
curve(dbeta(x,a+s,b+f), from=0, to=1, xlab="p",ylab="Pr. Density",lty=1,lwd=4)

#posterior distribution
curve(dbeta(x,a,b),add=TRUE,lty=3,lwd=4, col="red")

# add a legend
legend(.5,4,c("Prior (based on belief)","Posterior (updated with data)"), lty=c(3,2),lwd=c(3,3), col=c(
```




The beta cdf and inverse cdf functions *pbeta* and *qbeta* are useful in computing probabilities and constructing interval estimates for $p$.

For example, is it likely that the proportion of heavy sleepers is greater than .5? This is answered by computing the posterior probability $Pr(p \geq .5|data)$

```
1 - pbeta(0.5, a + s, b + f)
```

```
## [1] 0.01577408
```

This probability is small, so it is unlikely that more than half of the students are heavy sleepers.

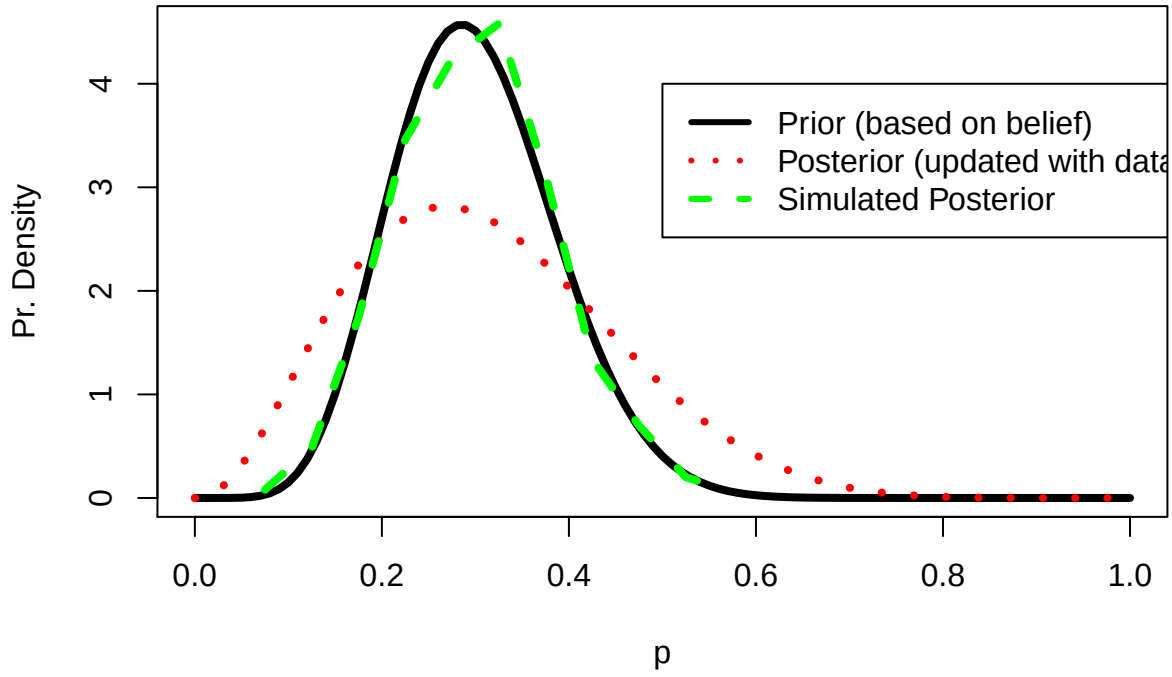
An alternative method of summarization of a posterior density is based on simulation. In this case, we can simulate a large number of values from the beta posterior density and summarize the simulated output.

Using the random beta command *rbeta*, we simulate 1000 random proportion values from the posterior:

```
ps = rbeta(1000, a + s, b + f)
```

Let's update our plot to see how this posterior compares

```
# use dbeta as the density of the beta distribution with shape parameters a+s and b+f
curve(dbeta(x,a+s,b+f), from=0, to=1, xlab="p",ylab="Pr. Density",lty=1,lwd=4)

#posterior distribution
curve(dbeta(x,a,b),add=TRUE,lty=3,lwd=4, col="red")

# get the histogram
h = hist(ps, plot=F)

lines(h$mids, h$density, lty=2,lwd=4, col="green")

# add a legend
legend(.5,4,c("Prior (based on belief)","Posterior (updated with data)","Simulated Posterior"),
       lty=c(1,3,2),lwd=c(3,3,3), col=c("black","red", "green"))
```




**Making predictions based on priors**

We have focused on learning about the population proportion of heavy sleepers $p$. Now we want to predict the number of heavy sleepers $y$ in a future sample of $m = 50$ students.

The predictive density of $y$ is given by

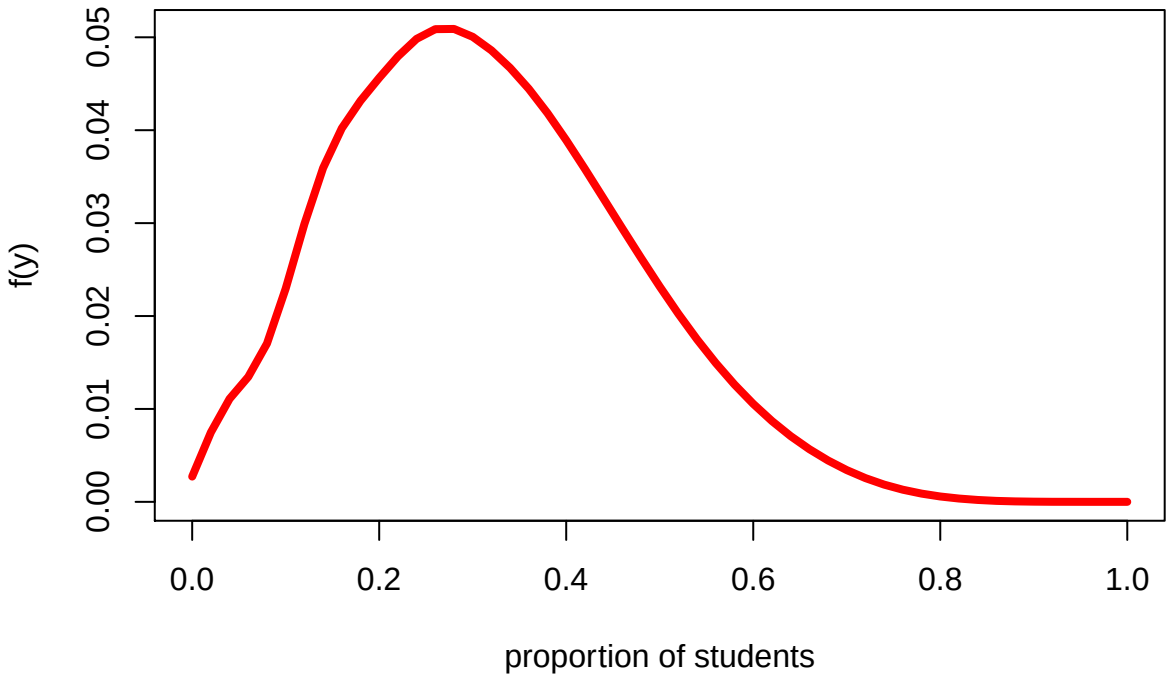
$f(y) = \int f(y|p)g(p)dp$

$g$ is a prior predictive density and $f$ is a posterior predictive density

The function **pdiscp** in the LearnBayes package can be used to compute the predictive probabilities when $p$ is given a discrete distribution.

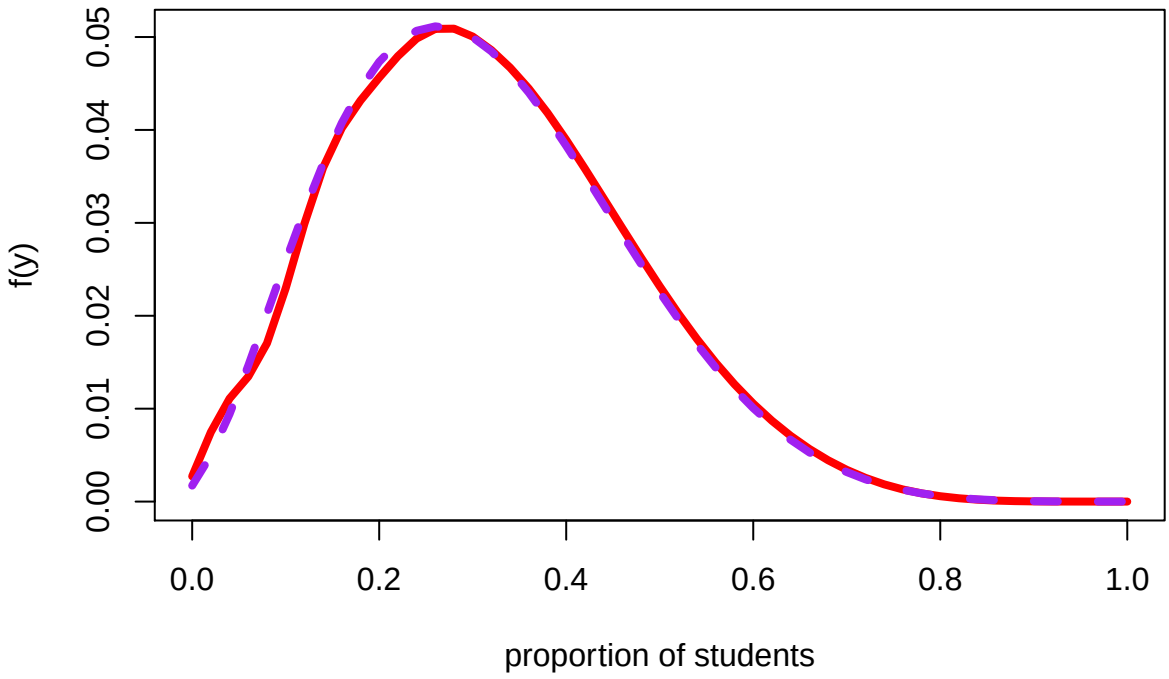
Let's first use our list of plausible proportions

```
m=50 #future sample size m
ys=0:m #number of successes
pred=pdiscp(p, prior, m, ys) #vector of predictive probabilities

# make a plot of predicted probabilities vs proportion of students
plot(ys/m, pred, 'l', lwd=4, col="red", ylab="f(y)", xlab="proportion of students")
```




Suppose instead that we model our beliefs about $p$ using a beta(a, b) prior (rather than plausible proportions). The predictive probabilities using the beta density are computed using the function *pbetap*

```
pred_beta = pbetap(c(a, b), m, ys)

# let's update our plot
plot(ys/m, pred, 'l', lwd=4, col="red", ylab="f(y)", xlab="proportion of students")
lines(ys/m, pred_beta, 'l', lwd=4, lty =2, col="purple")
```




**Predictions for any prior, through simulation**

One convenient way of computing a predictive density for any prior is by simulation. To obtain $y$, we first simulate, say, $p'$ from $g(p)$, and then simulate $y$ from the binomial distribution

We first simulate 1000 draws from the prior and store the simulated values in p:

```
pprime = rbeta(1000, a, b)
```

Then we simulate values of $y$ for these random $p'$ using the *rbinom* function

(remember, the number of successes in $N$ trials)

```
Ntrials = 50
y = rbinom(1000, Ntrials, pprime)
```

let's tabulate these frequencies

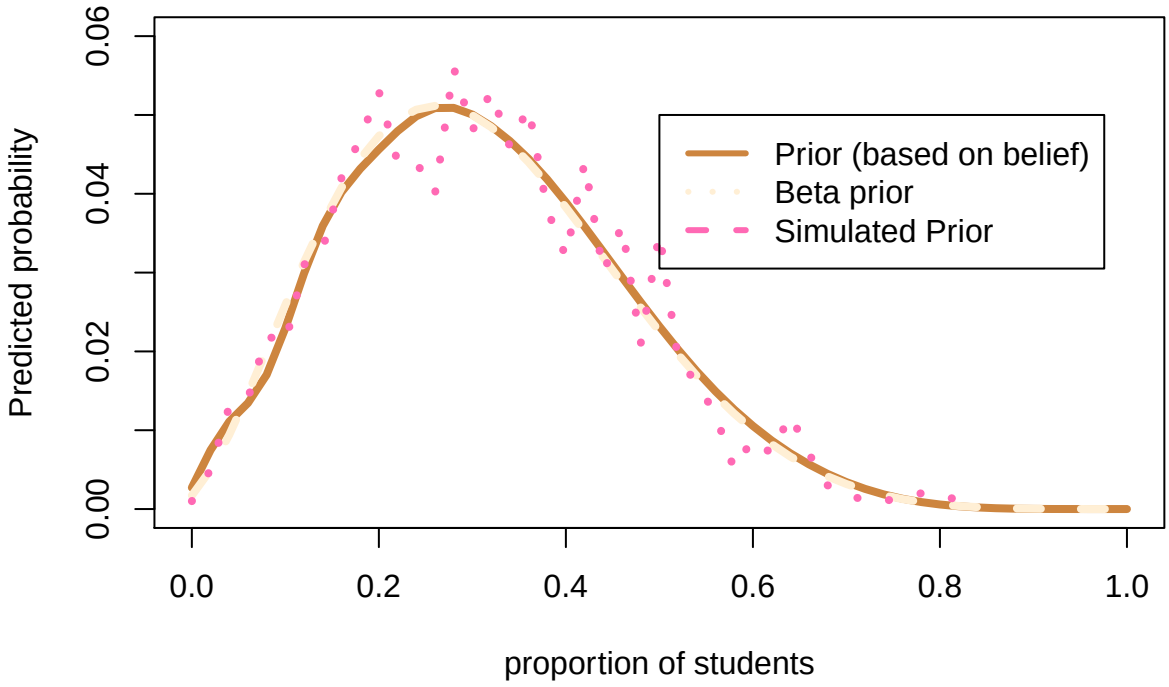
```
freq = table(y)
freq
```

```
## y
##  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
##  1  5 13 14 22 21 31 33 42 47 53 44 44 40 56 48 53 46 51 38 32 44 30 36 21
## 25 26 27 28 29 30 31 32 33 34 35 36 37 39 40 41 42
## 35 19 16 12  5  9  7 12  7  3  2  1  1  2  2  1  1
```

```
#convert frequencies to probabilities
predprob = freq/sum(freq)

ysim=as.integer(names(freq))
```

```
cols = c("peru","papayawhip", "hotpink")

# let's update our plot again
```

```
plot(ys/m, pred, 'l', lwd=4, col=cols[1], ylab="Predicted probability", xlab="proportion of students", y
lines(ys/m, pred_beta, 'l', lwd=4, lty =2, col=cols[2])
lines(ysim/m, predprob, 'l', lwd=4, lty =3, col=cols[3])

# add a legend
legend(.5,.05,c("Prior (based on belief)","Beta prior","Simulated Prior"),
       lty=c(1,3,2),lwd=c(3,3,3), col=cols)
```




We want to summarize the predicted probability distribution by an interval that overs at least 95% of the probability. The R function *discint* in the *LearnBayes* package is useful for this purpose.

```
dist=cbind(ysim,predprob)
discint(dist, .95)
```

```
## $prob
##    28
## 0.953
##
## $set
##  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
##  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
## 27 28 32
## 27 28 32
```

The probability that $y'$ falls in the interval $\{3, 31\}$ is 0.957, so the probability that $y'$ lies in the interval $\{.06, 0.62\}$ is 0.957.

## Bayesian analysis in practice: Markov Chain Monte Carlo

### Intro to MCMC

Bayesian analysis requires evaluating expectations of functions of random quantities as a basis for inference, where these quantities may have posterior distributions which are multivariate or of complex form or often both.

A turning-point in practical Bayesian computation was the development and application of sampling methods such as Markov Chain Monte Carlo (MCMC). MCMC is a class of algorithms which can efficiently characterize posterior distributions through drawing of randomized samples such that the points are distributed according to the posterior

This essentially is a continuous valued generalization of the discrete Markov chain setup described in the previous section

The MCMC sampling strategy sets up an irreducible, aperiodic Markov chain for which the stationary distribution equals the posterior distribution of interest.

A general way of constructing a Markov chain is by using a Metropolis-Hastings algorithm.

### Markov chain recap

Suppose that we have a three-state Markov process. Let $P$ be the transition probability matrix for the chain:

```
P = rbind(c(.5,  .25, .25),
          c(.2,  .1,  .7),
          c(.25, .25, .5))
P
```

```
##      [,1] [,2] [,3]
## [1,] 0.50 0.25 0.25
## [2,] 0.20 0.10 0.70
## [3,] 0.25 0.25 0.50
```

Compute the left eigenvector and normalize to get our steady state probability vector

```
v = eigen(t(P), FALSE)$vectors[,1]
v = v/sum(v) # normalise eigenvector
```

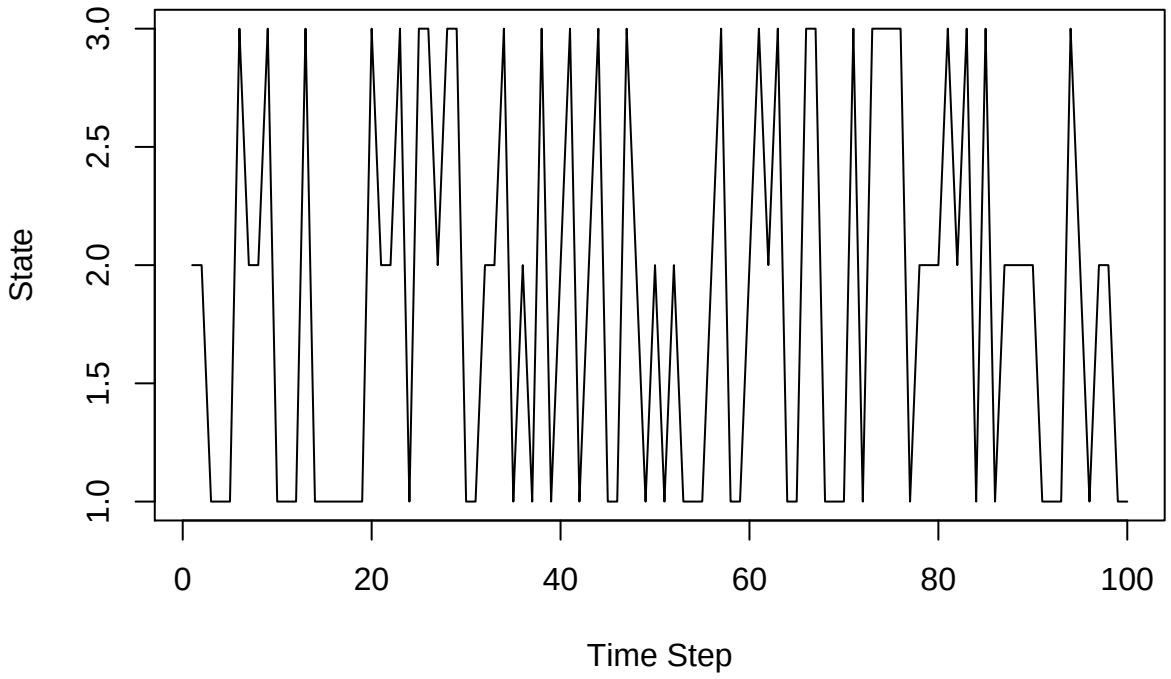
there is a 32 % chance of the chain being in state 1 after a few iterations regardless of where it started. So, knowing about the state of this chain at one point in time gives you information about where it is likely to be for only a few steps

Let's simulate this system. We take a 'current state' $(1, 2, 3)$, the transition matrix, and a number of steps to run. Each step, it looks at the possible places that it could transition to and chooses one

```
run = function(i, P, n) {
    res = integer(n) #preallocate the array
    for (t in seq_len(n))
        res[[t]] = sample(nrow(P), 1, pr=P[i,]) #get random sample with certain probability
    res
}
```

The chain running for 100 steps

```
samples = run(1, P, 100)
plot(samples, type="l", xlab="Time Step", ylab="State")
```

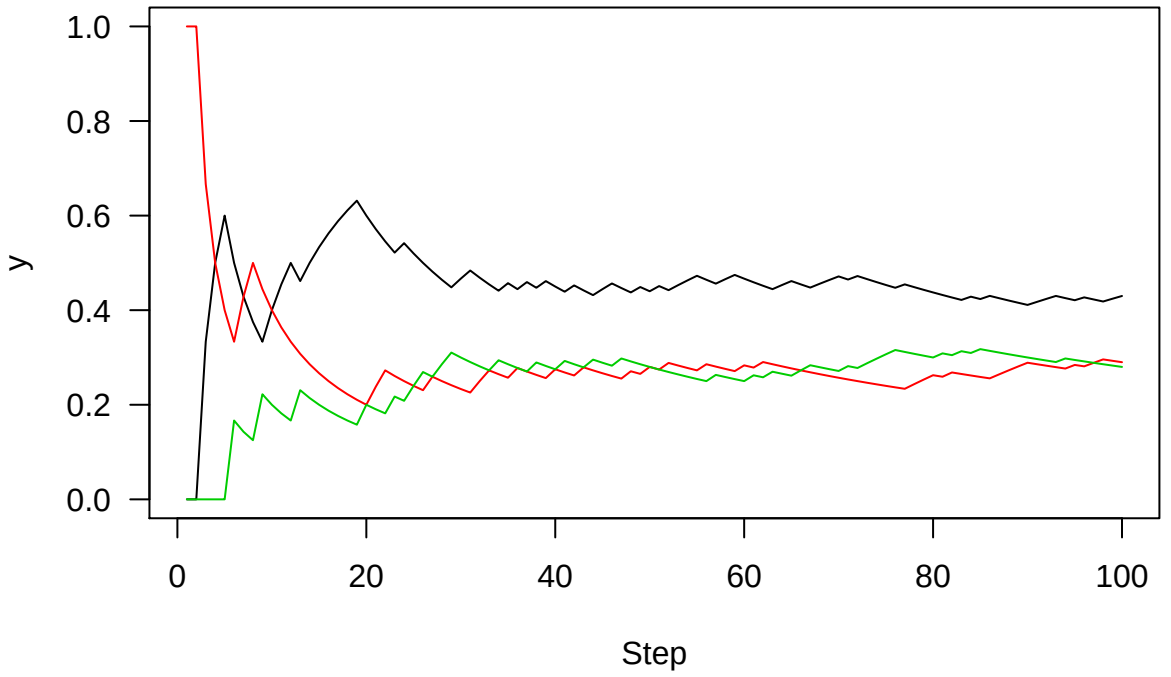



Rather than plotting state, plot the fraction of time that we were in each state over time:

First let's write a quick function for cumulative mean

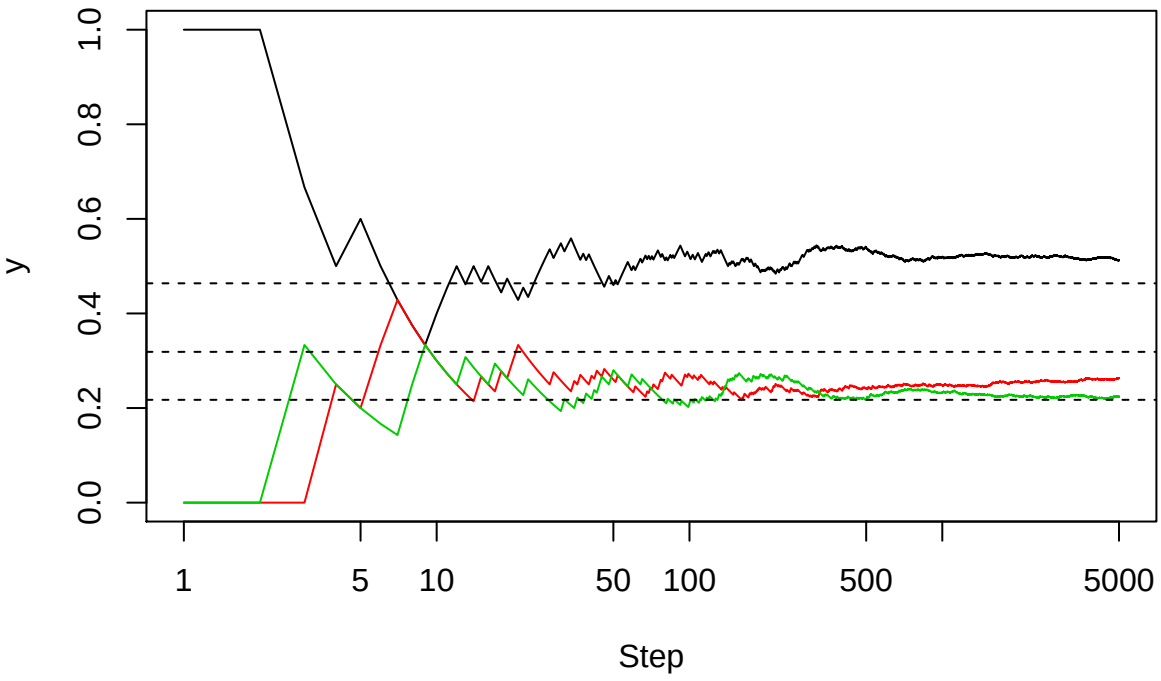
```
cum_mean = function(x){
  cumsum(x)/seq_along(x)
}
```

```
plot(cum_mean(samples == 1), type="l", ylim=c(0, 1),
     xlab="Step", ylab="y", las=1)
lines(cum_mean(samples == 2), col=2)
lines(cum_mean(samples == 3), col=3)
```



You can see it's starting to converge. Let's run it out longer

```
n = 5000
set.seed(1) #for reproducibility
samples = run(1, P, n)
plot(cum_mean(samples == 1), type="l", ylim=c(0, 1), log="x",
     xlab="Step", ylab="y", col=1)
lines(cum_mean(samples == 2), col=2)
lines(cum_mean(samples == 3), col=3)
abline(h=v, lty=2) #draw horizontal lines for steady state v
```




**Metropolis algorithm**

Simplest MCMC algorithm

Used to generate a large sample from posterior distribution of a parameter $p(\theta|x)$

We have some *posterior distribution* that we want to sample from, and we're going to evaluate some function $f(x)$ that is proportional to the probability density of the target distribution

that is, if $p(x)$ is the probability density function itself, $f(x) \propto p(x)$

We also need a probability density function $P(x)$ that we can draw samples from.

The algorithm proceeds as follows.

1. Start in some state $x_t$.
2. Propose a new state $x'$
3. Compute the "acceptance probability"
4. Draw some uniformly distributed random number $u$ from $[0, 1]$
5. if $u < \alpha$ accept the point, setting $xt + 1 = x'$. Otherwise reject it and set $xt + 1 = x_t$.

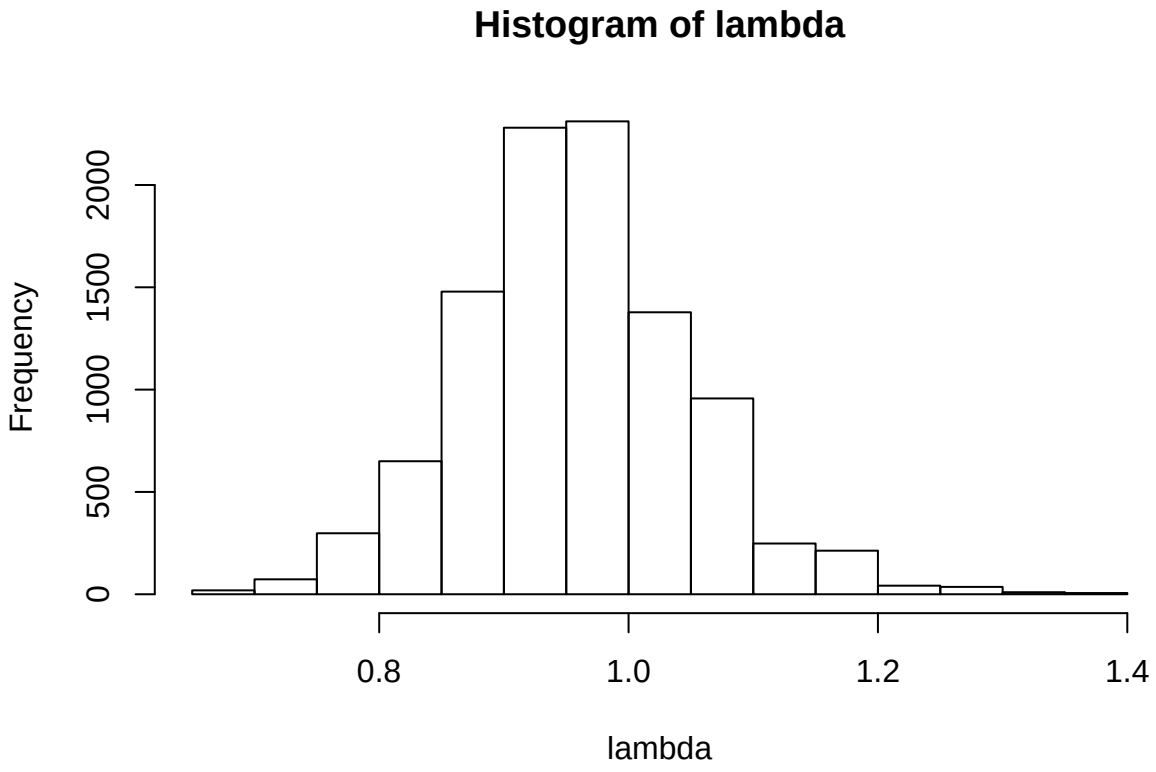
This will generate a series of samples $x0, x1, \ldots$. Note that where the proposed sample is rejected, the same value will be present in consecutive samples.

these are not independent samples from the target distribution; they are dependent samples; that is, sample $x_t$ depends on $x_{t-1}$ and so on. However, because the chain approaches a stationary distribution, this dependence will not matter so long as we sample enough points.

**Example 1: estimate the posterior distribution for the scale parameter from an Exponential distribution with a uniform prior probability distribution.**

We are only calculating $p(x|\lambda)$

```
# get 100 random numbers from the exponential distribution
x = rexp(100)

# we're gonna run the simulation 10,000 times
Nsteps = 10000
# preallocate an output array
lambda = rep(0,Nsteps)

# start with a value of 1
lambda[1] = cur_lambda <- 1  # Starting value
for (i in 1:Nsteps) {
   #propose a new lambda
   prop_lambda = runif(1,0,5)  #sample from uniform distribution
   #define the acceptance probability
   alpha = exp(sum(dexp(x,prop_lambda,log=TRUE)) - sum(dexp(x,cur_lambda,log=TRUE)))
   if (runif(1) < alpha)
     # if accepted, set current lambda as prop lambda
     cur_lambda = prop_lambda
     # and stash this lambda in lambda array
     lambda[i] = cur_lambda
}

hist(lambda)
```



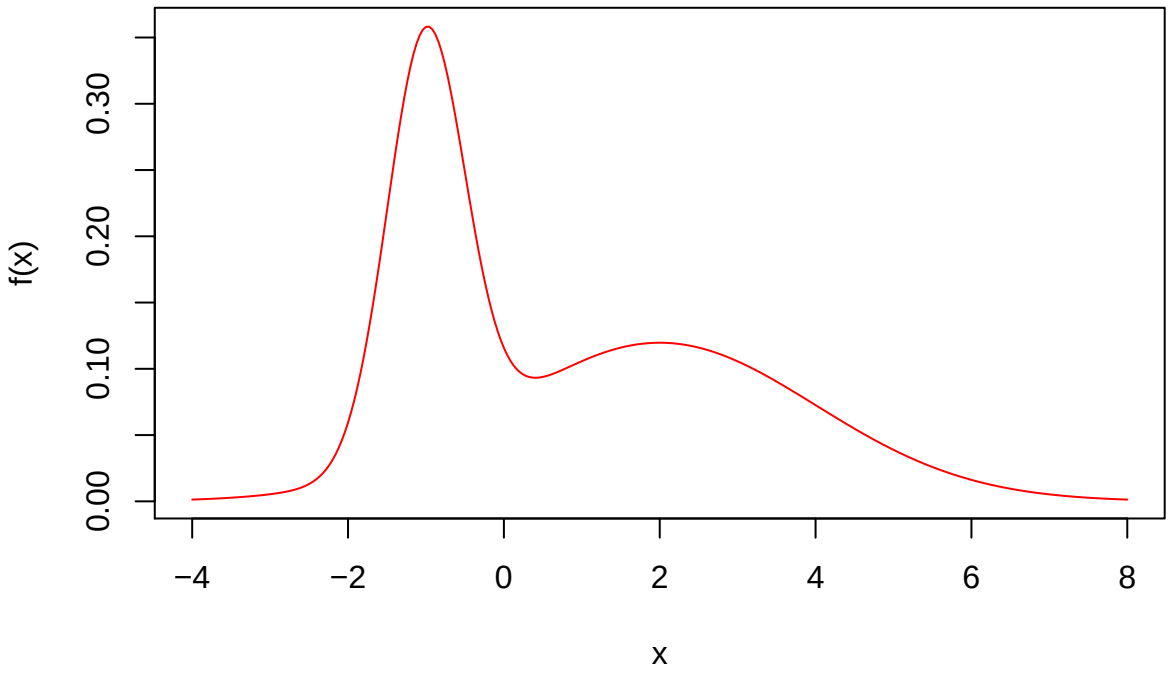

**Example 2: estimate the posterior distribution for the location parameter from an "real" distribution with a uniform prior probability distribution.**

Let's illustrate this by defining a posterior distribution that is a weighted sum of two normal distributions

```
p = 0.4
mu = c(-1, 2)
sd = c(.5, 2)

f = function(x)
    p     * dnorm(x, mu[1], sd[1]) +
    (1-p) * dnorm(x, mu[2], sd[2])
```

Make a plot over some defined range of $x$

```
curve(f(x), col="red", -4, 8, n=301)
```




we're going to draw samples from a normal PDF centred on the current point, $x$, with a standard deviation of 4

```
q = function(x) rnorm(1, x, 4)
```

Here is the Hastings-Metropolis algorithm implemented for 1 step

the probability of acceptance is determined by comparing the values of the function $f(x)$ of the current and candidate sample values with respect to the desired distribution $P(x)$.

```
# we're passing it a value for x, and 2 functions, f, and q
onestep = function(x, f, q) {
    ## Pick new point
    xp = q(x)
    ## Acceptance probability:
    alpha = min(1, f(xp) / f(x))
    ## Accept new point with probability alpha:
    if (runif(1) < alpha)
        x = xp
    ## Returning the point:
    x
}
```

And this next function runs for a number of steps

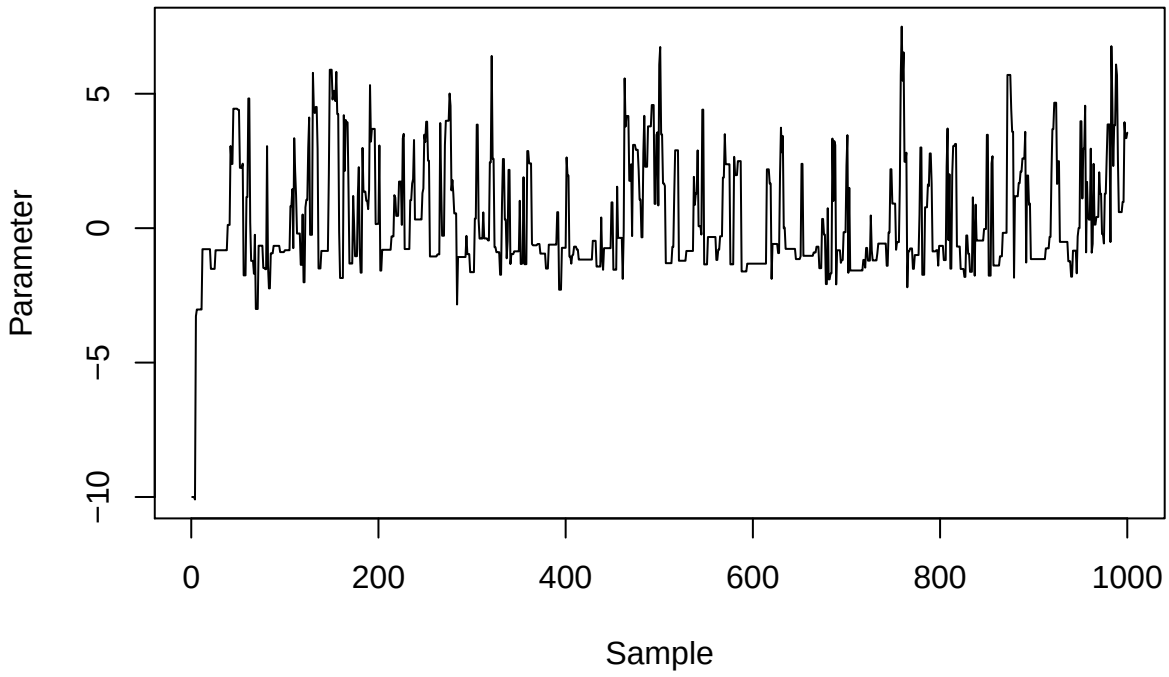
```
#start with x
run = function(x, f, q, nsteps) {
    #return a matrix with nsteps rows, and the same number of columns as length(x)
    res = matrix(NA, nsteps, length(x))
    for (i in seq_len(nsteps))
        # we need to use an "assignment vector" to pass x to the function
        res[i,] = x <- onestep(x, f, q)
    drop(res) #squeeze the dimensions to 1 level
}
```

We'll pick a place to start

```
start = -10
res = run(start, f, q, 1000)
```

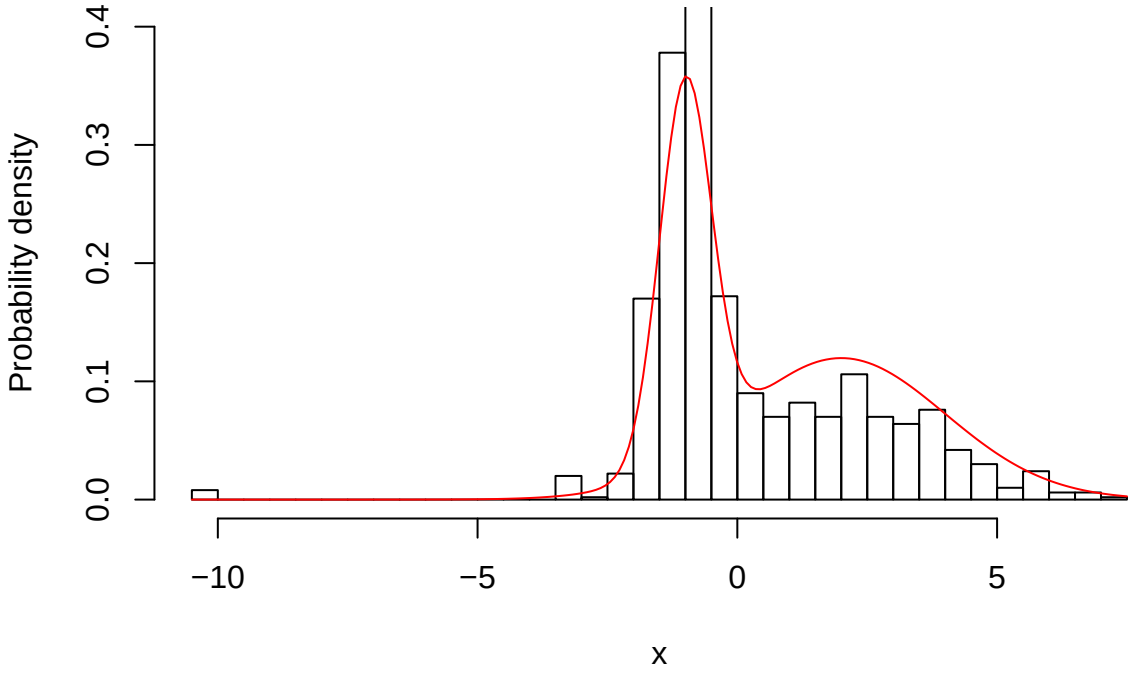
Let's plot the first 1000 steps of the Markov chain

```
plot(res, type="l", ylab="Parameter", xlab="Sample")
```



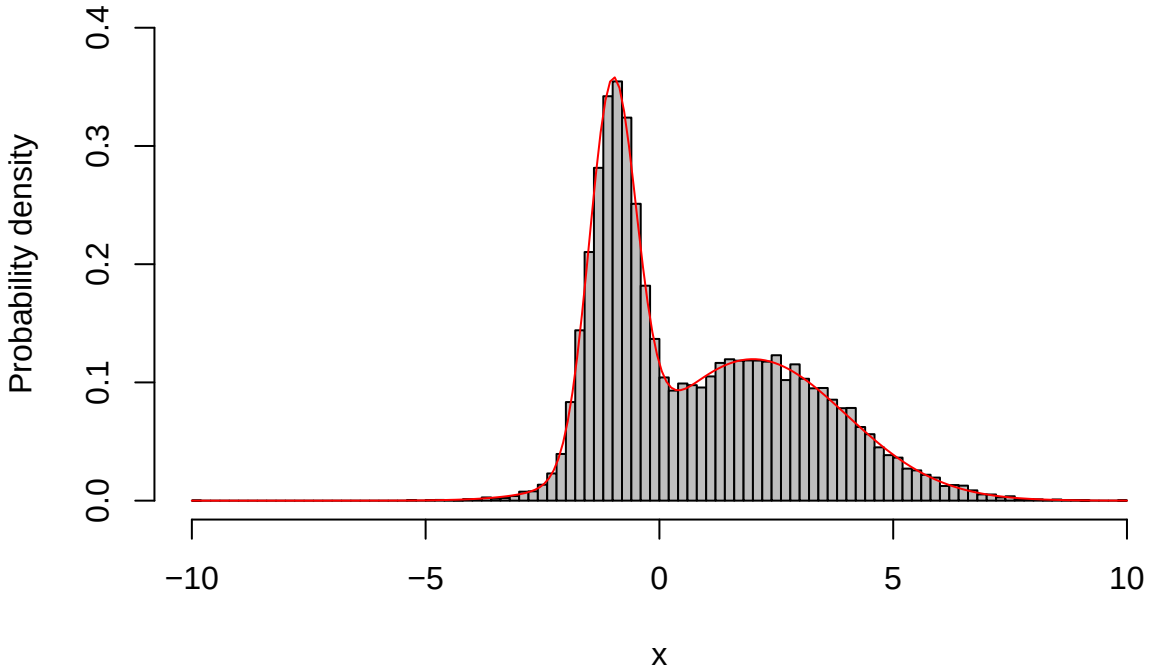

Even with only a thousand (non-independent) samples, we're starting to resemble the posterior distribution fairly well.

```
hist(res, 50, freq=FALSE, main="", ylim=c(0, .4), xlab="x", ylab="Probability density")

# add the curve
curve(f(x), add=TRUE, col="red", n=200)
```




Run this for a really long time and you'll see they match exactly

```
set.seed(1)
longrun = run(start, f, q, 50000)

hist(longrun, 100, freq=FALSE, main="", ylim=c(0, .4),
     xlab="x", ylab="Probability density", col="grey")

curve(f(x) , add=TRUE, col="red", n=200)
```




Now let's see what happens when we run the chain with different distributions to same from

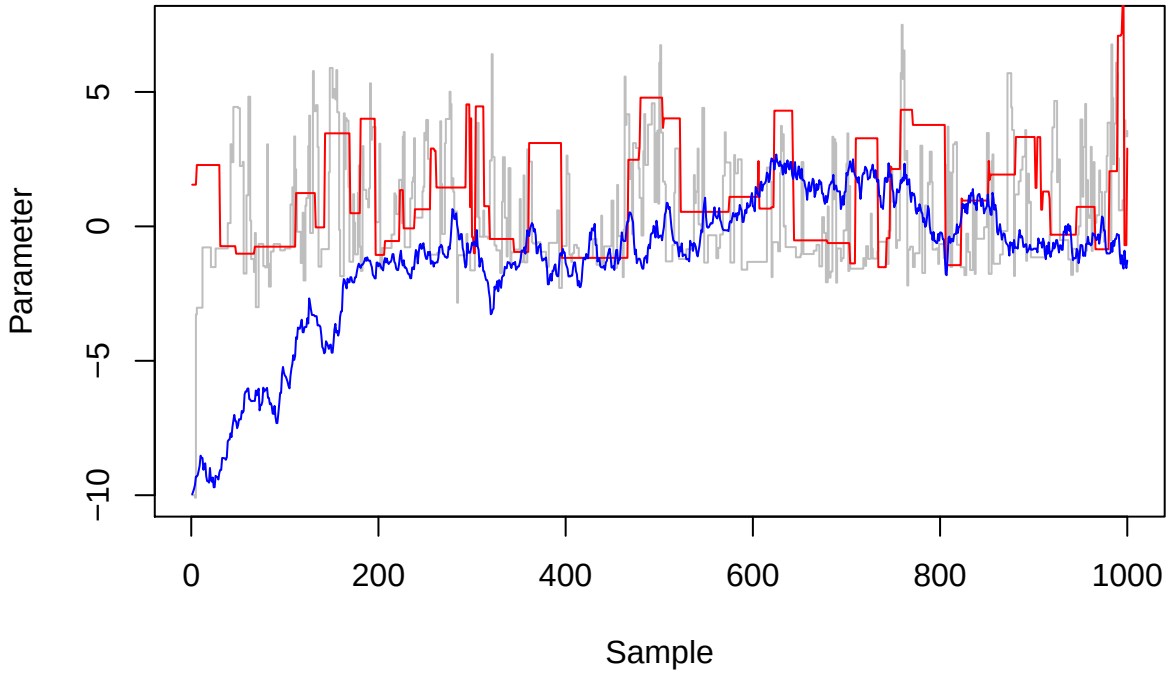
1. very wide standard deviation (33 units) and
2. a very small standard deviation (3 units).

1) will mean that there might be very large differences between x(t) and proposed x', whereas 2) will mean that all proposed steps are small

```
wide = run(-10, f, function(x) rnorm(1, x,  33), 1000)
narrow = run(-10, f, function(x) rnorm(1, x,  .3), 1000)
```

Let's plot the first 1000 steps of the Markov chain

```
plot(res, type="s", xpd=NA, ylab="Parameter", xlab="Sample",
     col="grey")
lines(wide, col="red")
lines(narrow, col="blue")
```



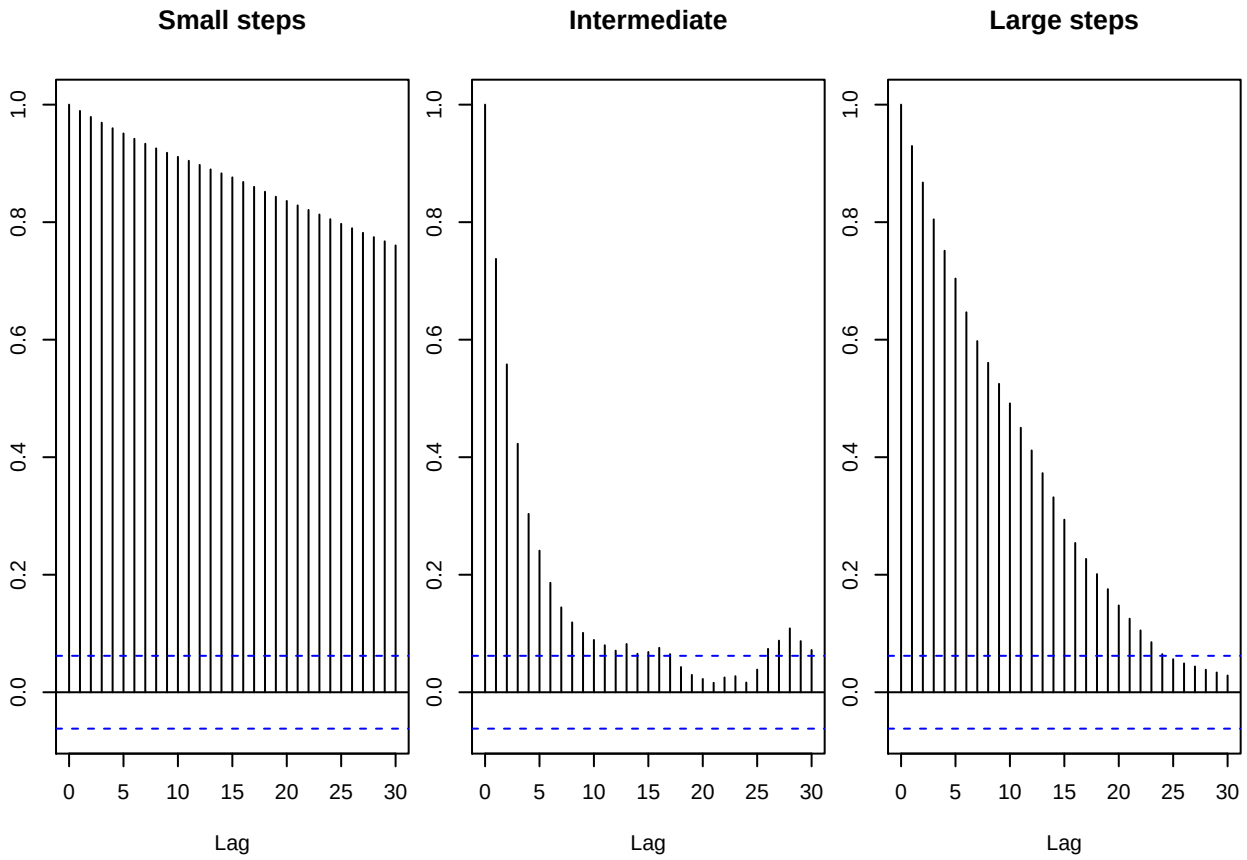

The original (grey line) trace is bouncing around quite freely.

the red trace rejecting most proposals, so it tends to stay put for a long time.

The blue trace proposes small moves that tend to be accepted, but it moves following a random walk for most of the trajectory

You can see the degree to which proposed x' are accepted

```
par(mfrow=c(1, 3), mar=c(4, 2, 3.5, .5))
acf(narrow, main="Small steps")
acf(res, main="Intermediate")
acf(wide, main="Large steps")
```

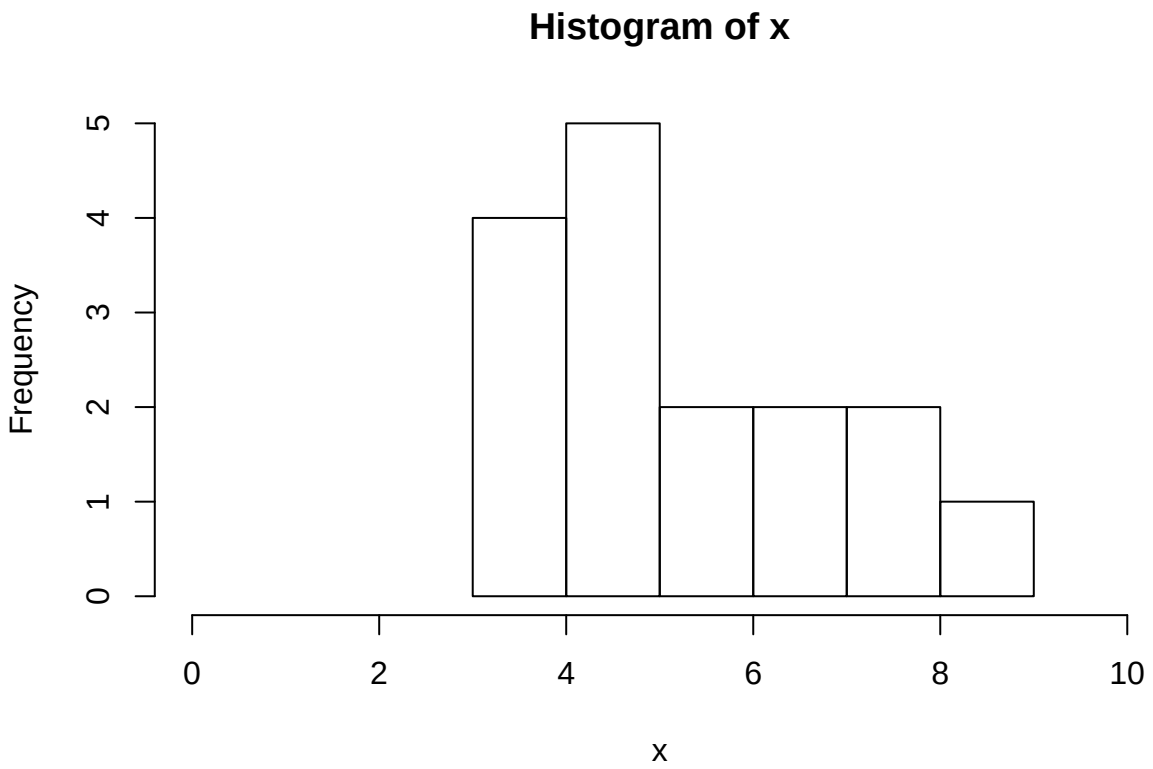



**Example 2: estimate the posterior distribution for 2 parameters from an Laplace distribution with a uniform prior probability distribution.**

Laplace has 2 parameters, location $\mu$ and scale $b$

A dataset consists of counts of the number of wolf pups in a sample of 16 wolf dens. Reference: **The Wolf in the Southwest: The Making of an Endangered Species**, Brown, D.E., University of Arizona Press

```
# The wolf pups dataset
x = c(5, 8, 7, 5, 3, 4, 3, 9, 5, 8, 5, 6, 5, 6, 4, 7)

hist(x, xlim=c(0,10))
```




The log posterior density of the Laplace distribution model, when assuming uniform priors:

```
#install.packages("VGAM")
library(VGAM)
```

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

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

```
##
## Attaching package: 'VGAM'
```

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

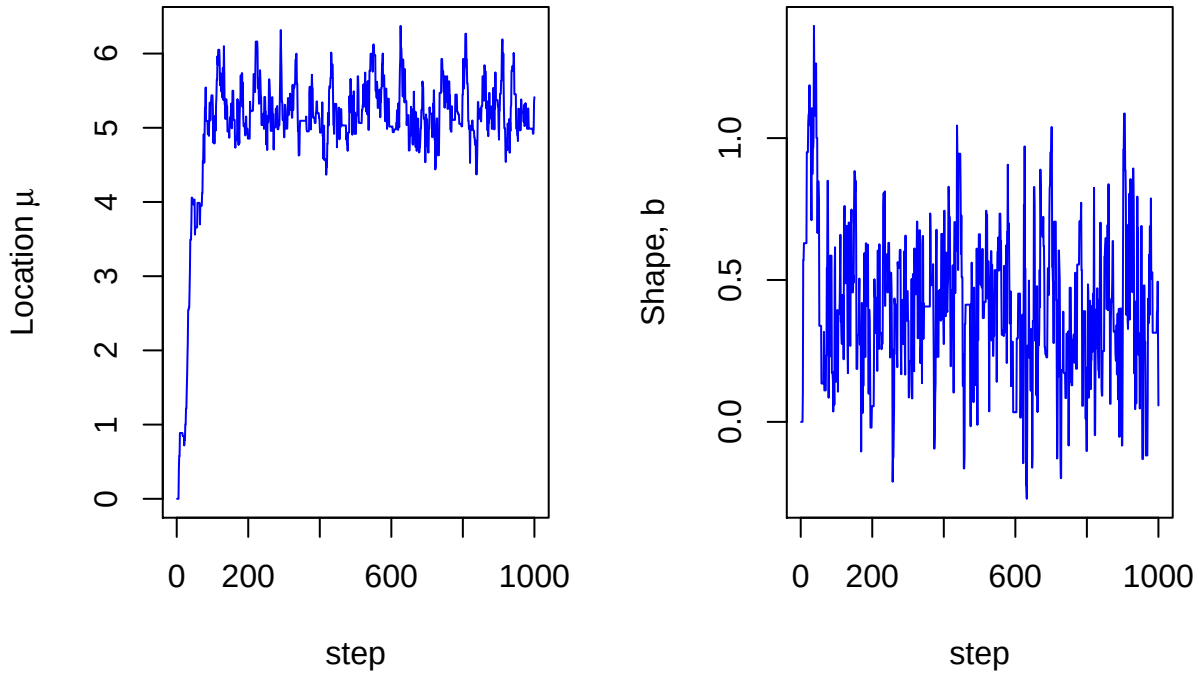
```
#The Laplace distribution is not part of base R but is available in the VGAM package.
laplacef = function(pars) {sum(VGAM::dlaplace(x, pars[1], exp(pars[2]), log = TRUE))}
```

we need to write a new steps function. This time we'll combine the 'steps' and the 'allsteps' together

```
# we're passing it 1 function, f, an initializing vector, and number of time steps to run
allsteps_laplace = function(laplacef, inits, nsteps) {

    samples = matrix(NA, nrow = nsteps, ncol = length(inits))
    samples[1,] = inits

    for(i in 2:nsteps) {
      #current log density
      curr_log_dens = laplacef(samples[i - 1, ])

      # pick a new point
      xp = samples[i - 1, ] + runif(length(inits), -0.5, 0.5)
```

```
      #acceptance probability (remember this is a logarithm)
      alpha = laplacef(xp)

      ## Accept new point with probability exp(alpha - current log density):
      if(runif(1) < exp(alpha - curr_log_dens)) {
        samples[i, ] = xp
      } else {
        samples[i, ] = samples[i - 1, ]
      }
    }
    #return the samples
    samples
}
```

```
samples = allsteps_laplace(laplacef, inits = c(0,0), nsteps=1000)

par(mfrow=c(1,2))
# Plotting location MU
plot(samples[,1], type = "l", ylab = expression(Location ~ mu), col = "blue", xlab="step")

# Plotting shape B
plot(samples[,2], type = "l", ylab = "Shape, b", col = "blue", xlab="step")
```




Calculating median posterior and 95 % C.I. discarding the first 250 draws as "burn-in"

```
mus = quantile(samples[250:1000,1], c(0.025, 0.5, 0.975))
mus
```

```
##     2.5%      50%    97.5%
## 4.626818 5.171095 5.998036
```

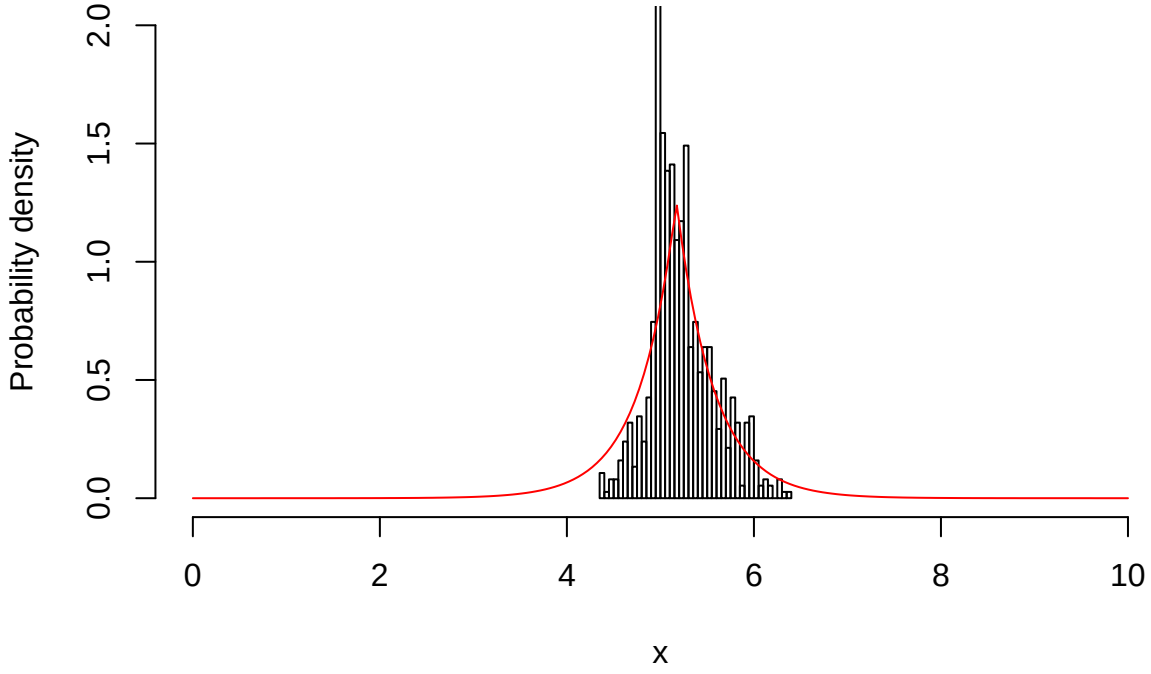
same thing for b

```r
bs = quantile(samples[250:1000,2], c(0.025, 0.5, 0.975))
bs
```

```
##       2.5%        50%      97.5%
## -0.1182964  0.3991501  0.9365079
```

Ok, we have a Laplace model, how well does it work?

```r
hist(samples[250:1000], 50, freq=FALSE, main="", ylim=c(0,2), xlim=c(0, 10),
     xlab="x", ylab="Probability density")

# add the curve
curve(VGAM::dlaplace(x, mus[[2]], bs[[2]]), add=TRUE, col="red", n=200)
```




## links

http://faculty.washington.edu/kenrice/BayesIntroClassEpi515.pdf

http://www.scholarpedia.org/article/Bayesian#Markov__Chain__Monte__Carlo__.28MCMC.29

https://github.com/bayesball/LearnBayes/blob/master/inst/doc/BinomialInference.pdf

https://github.com/bayesball/LearnBayes/blob/master/inst/doc/DiscreteBayes.pdf

https://github.com/bayesball/LearnBayes/blob/master/inst/doc/MCMCintro.pdf