6 Monte Carlo simulation

To begin, the word 'simulation model' is slightly obscure or even misleading. Simulation itself is not a model, but simulation is used as a technique for approximate computations of a particular model. In our case, the model is some probability density, that can be multidimensional. Therefore, the purpose here is *posterior simulation*. Simulation is naturally computer intensive. That is, someone has to program an algorithm for doing it. And it can be more laborious than the specification of the model to be simulated. Perhaps for that reason, attention is (too) easily diverted from the discussion of the *model* to the discussion of the *algorithm* - as the 'simulation model' or 'computer model'. But these are slightly different things. There can be many different algorithms for simulating the same model. The model is intrinsically related to the scientific question, whereas simulation is a tool to compute something interesting out of the model.

In bayesian inference, simulation is an essential tool because we can then use an infinite variety of models instead of the simple ones that can be solved analytically by using conjugate priors.

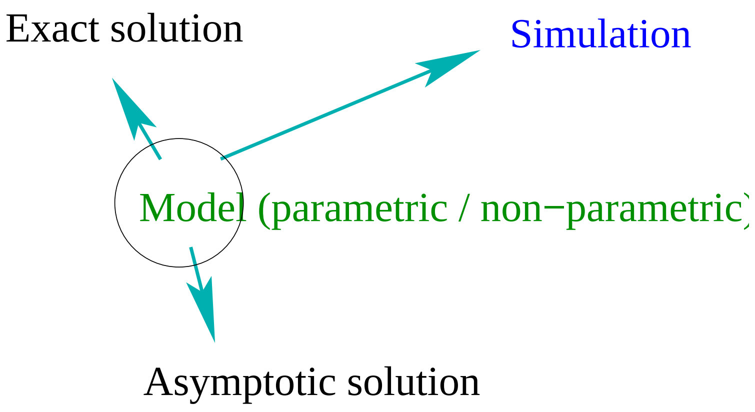


Figure 1: Model and its computation

6.1 Example: $\pi = 3.14159\ldots$

Any estimation problem could be approached by simulation. For example, the calculation of $\pi = 3.14159\ldots$. Assume a square of size $L \times L$. Assume as big a circle that can fit inside the square. The circle has radius $L/2$. The proportion of the area of the circle to the area of the square is then:

$$q = \frac{\pi (L/2)^2}{L^2} = \frac{\pi}{4}$$

Imagine then that we can play with darts and our target is the square so that the darts can fall *evenly randomly* all over the square. The probability to hit within the circle is then $q = \pi/4$. After n darts, we count how many times we hit the circle. The percentage of hits, q_n, is an approximation of the exact number $\pi/4$. Therefore,

$$\lim_{n\to\infty} q_n = q,$$

and we can approximate $\pi \approx 4q_n$. This shows the essential principle of Monte Carlo simulation. The more darts we simulate, the more accurate is our approximation. There would be other ways to compute approximations of π, and they can be much more efficient. Monte Carlo simulation is a tool that is usually used when no other tool can help, or when we are lazy to think of alternatives and the computers are conveniently available...

In this example, the model was a 2-dimensional uniform density of variables (X, Y) over a rectangle $[-L/2, L/2] \times [-L/2, L/2]$, and the probability we approximated was

$$P(X^2 + Y^2 < r^2).$$

Simulation in R could be done as:

```
X <- runif(1000000)
Y <- runif(1000000)
q <- sum(X^2+Y^2<1)/1000000
4*q
```

An early example of approximating π by simulation is the Buffon's needle experiment (*Georges Louis Leclerc Comte de Buffon 1707-1788*). In the experiment, we first make parallel lines at equal intervals on a flat surface. Then, a needle is 'randomly' dropped on the surface and we count how often the needle crosses a line.

"Monte Carlo -method" and its systematic use started from A-bomb research, dating back to 1944. The theory was developed by e.g. Fermi, Metropolis & Ulam.

Note the use of indicator variables:

Define an indicator variable $1_A(x)$ so that it is one when the value of x implies that A is true, and zero otherwise. Earlier we had the result $P(A) = E(1_A(X))$. The expected value is taken with respect to the distribution of variable X, so if we can simulate random draws from that distribution, then we can approximate (for any A we are interested in):

$$P(A) = E(1_A(X)) \approx \frac{1}{N} \sum_{i=1}^{N} 1_A(x_i).$$

In general, e.g. means of any quantities can be approximated by the averages of the simulated values, and the whole distribution can be approximated by the histogram of the simulated values. (This is the duality between the probability distribution and the corresponding empirical distribution).

6.2 Simple Monte Carlo simulation of binomial distribution

As stated above, the binomial probability of outcome X is:

$$P(X \mid N, r) = \binom{N}{X} r^X (1-r)^{N-X}$$

There are many readily available tools in different statistical packages that can be used to draw random samples from standard distributions. For example in R:

```
X<-rbinom(1000,N,r)
hist(X,freq=FALSE,min(X):max(X),
xlab='X',ylab='Probability',
main='Empirical distribution')
```

will generate 1000 random values of X from binomial(N, r) and plot the empirical distribution. The more samples, the more accurately the empirical distribution will represent the true distribution.

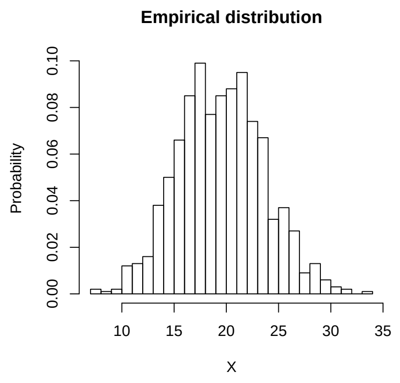


Figure 2: Simulation result of 1000 draws from binomial(100,0.2).

If the binomial distribution is not available, it is sufficient to have the most basic random number generator, the uniform distribution. For example in R:

```
for(i in 1:1000){X[i]<-sum(runif(100)<0.2)}.
```

This fundamental uniform distribution is usually found in most software (but some of them may not be of good quality). In computers, 'random values' are not truly random, but pseudo-random, i.e. produced by some deterministic algorithm. Once we have a generator of U$(0, 1)$ variables, then, in principle, we can generate all other distributions more or less efficiently and more or less accurately. And once we can simulate some random variable X from its distribution, it is very easy to obtain empirical distributions of any transformations of X. Using Monte Carlo simulation, we don't need to solve the analytical form of the probability density of the transformed variable $g(X)$. For example:

```
x <- runif(10000)
hist(x^2,100)
```

6.3 Inverse cdf -method

A simple Monte Carlo simulation of a continuous density can be based on solving the inverse of the cumulative density function (cdf). If the cumulative probability function $y = F(x)$ can be inverted analytically, $x = F^{-1}(y)$, then we can draw a value y from uniform density $U(0, 1)$ and calculate $x = F^{-1}(y)$. The resulting random values X will have cumulative probability F. Proof: $P(F^{-1}(Y) < x) = P(Y < F(x)) = F(x)$.

6.4 Transformation methods

In principle, it is possible to find some clever transformations of the basic uniform random variable so that the transformed variable has the required distribution. Then, it is only required that we can obtain random variables $X \sim \mathrm{U}(0,1)$, and compute the transformation $g(X)$. The problem is how to find what the transformation g should be for a given target distribution. For normal distribution, there are several known transformations, some of them work better than others. For example, if $X_1 \sim \mathrm{U}(0,1)$, and $X_2 \sim \mathrm{U}(0,1)$ independently, then

$$Y_1 = \sqrt{-2\log(X_1)}\cos(2\pi X_2)$$
$$Y_2 = \sqrt{-2\log(X_1)}\sin(2\pi X_2)$$

can be used as independent random draws from $\mathrm{N}(0,1)$ distribution. The idea of transformations is also used in data-analysis when the data distribution appears 'non standard'. After a suitable transformation, it can be made approximately normal, in which case a normal distribution might be chosen as the conditional distribution of data.

6.5 Example: Monte Carlo & virus exposure

Description: a large vat contains several million litres of milk that is known to be contaminated with some virus, but the level of contamination is unknown. Then, 50 samples, each 1 litre, are taken from the vat and tested. The testing is so reliable that it gives positive result if the number of virus particles is at least 1. It is not possible to distinguish the exact number of virus particles in a positive sample. Estimate the concentration of virus particles if there were 7 positive tests.

Assume that one needs to consume at least 8 virus particles in a short time to become infected. If m litres is consumed in a short time, what is the probability of infection? You can try different values for m in the range 4-10.

6.5.1 Estimating concentration in milk

The probability model for the number of virus particles X in a sample volume S litres can be assumed Poisson(λS), where λ is the mean concentration per litre. If the sample size is one litre, then the conditional probability of having no particles is $P(X = 0 \mid S = 1, \lambda) = \exp(-\lambda)$. Therefore, the probability of a positive test result is

$$p = P(X \geq 1 \mid S = 1, \lambda) = 1 - \exp(-\lambda),$$

which also specifies λ as a function of p: $\lambda = -\log(1-p)$. When 50 samples are taken, the probability of 7 positives is binomial

$$P(Y = 7 \mid N = 50, p) = \mathrm{binomial}(Y = 7 \mid 50, p).$$

Assuming a uniform prior for p, we get the posterior of p as

$$\pi(p \mid Y = 7, N = 50) = \mathrm{Beta}(7+1, 50-7+1) = \mathrm{Beta}(8, 44).$$

To obtain the posterior distribution of λ by Monte Carlo simulation, just draw many random samples of p from this beta-distribution, and then calculate λ from each sampled value of p, and draw the histogram. (Try using R, WinBUGS, or any suitable software).

6.5.2 Probability of infection

The probability of infection is the probability of having at least 8 particles, in a consumption volume of m.

$$P(\text{infection} \mid \lambda, m) = P(X \geq 8 \mid \lambda, m),$$

where $X \sim \text{Poisson}(\lambda m)$. The probability of infection can be expressed more easily as

$$(\spadesuit)\ \ 1 - P(X \leq 7 \mid \lambda, m) = 1 - \sum_{i=1}^{7} \frac{(m\lambda)^i \exp(-m\lambda)}{i!}.$$

Since λ is uncertain, this expression is also uncertain, and we obtain the final *probability* as

$$P(\text{infection} \mid Y = 7, N = 50, m) = \int_0^\infty P(\text{infection} \mid \lambda, m)\pi(\lambda \mid Y = 7, N = 50)\ \mathbf{d}\lambda.$$

In other words: taking the weighted average of ($\spadesuit$), weighting by the posterior density of λ. Using Monte Carlo, this integral can be calculated approximately by simulating p from the beta density, then calculating λ for each sampled p, and calculating ($\spadesuit$) for each sampled λ, and finally taking the average of the simulated sample of ($\spadesuit$).

Alternatively, if ($\spadesuit$) is interpreted as the proportion of 'servings' (of size m) with infective dose in a large population of all servings produced from this vat, then we might also report the distribution of the *proportion* ($\spadesuit$) with a bayesian CI.

6.6 Rejection sampling

This is a method that produces independent samples from the target distribution, but *indirectly* by generating from an instrumental distribution and discarding some of the results. Therefore, it is not so efficient as direct Monte Carlo method applied to a target density.

The goal is to draw random samples from target density $\pi(x)$. Assume that we have no easy way of sampling directly from it - but we can easily compute the values of that density function for any x. In rejection sampling, we need to find a function $g(x)$ so that g is a probability density over the same support as π, or at least the integral of g is finite so that it can be normalized to one, and

- we can draw samples from $g(x)$ (or a density proportional to g),
- the importance ratio $\pi(x)/g(x)$ must have a known bound. I.e. there must be some known constant M so that $\pi(x)/g(x) \leq M$ for all values x. In other words: $\pi(x) < Mg(x)$ so that the target density $\pi(x)$ is below $Mg(x)$.

The algorithm is then:

1. Sample X from $g(x)$ (or a density proportional to g).
2. With probability $\pi(x)/(Mg(x))$, accept x as a new draw from π, otherwise return to step 1.

This technique requires some preliminary work to select a suitable density g, so we would need to inspect the target density to some extent. At best, if g is closely the same as π, this method can be efficient.

6.7 Importance sampling

The goal here is to estimate the expected value of some $h(x)$, when x has density $\pi(x)$ which is difficult to sample directly.

$$E_\pi(h(x)) = \int_{-\infty}^{\infty} h(x)\pi(x)\mathbf{d}x = \int_{-\infty}^{\infty} h(x)\frac{\pi(x)}{g(x)}g(x)\mathbf{d}x = E_g\Big(h(x)\frac{\pi(x)}{g(x)}\Big) \approx \frac{1}{N}\sum_{i=1}^{N} h(x_i)\frac{\pi(x_i)}{g(x_i)}$$

where x_i are drawn from distribution $g(x)$. In this method, we are not rejecting any generated values x_i, like in rejection sampling. The quantities $\pi(x_i)/g(x_i)$ are called *importance ratios*, or *importance weights*. If these weights can take large values with small probability, then the result might be too much influenced by the chance of missing some large weights (because we can only generate a limited sample). The distribution of weights could be monitored to check if this could happen: estimates could be poor if the largest importance weights are 'too large' compared to others. (The behavior of small importance weights is less crucial because they have small effect on the estimate anyway).

Example: estimate $E(X^2)$, where $X \sim \mathrm{N}(0,1)$. In this case we know from theory that X^2 is χ^2_1-distributed with mean 1. Try importance sampling with $g(X)$ as t-distribution with 3 degrees of freedom.

```
x<- rt(10000,3)
w<- dnorm(x,0,1)/dt(x,3)
r<- x*x*w
mean(r)
```

6.8 Markov chain Monte Carlo (MCMC)

This is a version of Monte Carlo sampling in which the consecutive samples are not independent. Hence, if x_i and x_{i+1} are two consecutive samples from the same target distribution, then in Monte Carlo sampling $P(x_i, x_{i+1}) = P(x_i)P(x_{i+1})$, but in MCMC: $P(x_i, x_{i+1}) = P(x_{i+1} \mid x_i)P(x_i)$.

MCMC is based on constructing a Markov chain so that its stationary distribution is the required target distribution.

This means that the ordinary Monte Carlo method is more efficient than MCMC. But in many problems, direct Monte Carlo is not possible. Instead, MCMC is extremely general method and can be widely applied for computing e.g. posterior densities. We only need to be able to compute values of the (unnormalized!) target density at all points x.

But MCMC needs to be applied carefully since there is no guarantee that the results are automatically correct after a *finite* number of iterations. We need to check for possible convergence problems. The general MCMC algorithm is of the form:

1. Set initial value x_1. Set counter $i = 1$.

2. Generate next value, conditionally on the previous: $x_{i+1} \sim f(x \mid x_i)$, set counter $i = i + 1$.

3. Go back to 2, until required sample size is obtained.

There are many different versions of MCMC algorithms, e.g. slice sampling, Gibbs-sampling, Metropolis-algorithm, Metropolis-Hastings algorithm.

$\boxed{\text{WinBUGS} = \textbf{B}\text{ayesian inference }\textbf{U}\text{sing }\textbf{G}\text{ibbs }\textbf{S}\text{ampling}}$

MCMC methods always require some form of *convergence diagnostics*. It may happen that the sampler is simply stuck in one part of the parameter space. Then the resulting MCMC sample would not represent the target distribution! Therefore, it is good to investigate using other techniques what the posterior density might look like. Also, it is good to use several different MCMC runs with different starting values. (WinBUGS convergence diagnostics is based on this idea). It is always good to think of simple point estimates that could be calculated from the data. Obviously, sensible point estimates should be roughly in the region where the posterior distribution is. It is not good idea to apply MCMC blindly and take the results automatically, without double checking. However, all convergence diagnostics can only indicate lack of convergence. They can never prove that the target distribution is really correctly and accurately represented by the MCMC sample. The most problematic situations occur if the posterior is multimodal and multidimensional.

6.8.1 Slice sampling

Example: simulating from a truncated normal distribution:

$$\pi(x \mid \mu, \sigma^2) 1_{\{x>L\}} \quad \propto e^{-0.5(x-\mu)^2/\sigma^2} 1_{\{x>L\}},$$

This could also be done by simulating random draws from an untruncated density, and then accepting only values greater than L. But if L is large, then this algorithm would be running a long time before we have a reasonable sample. Slice sampling is actually one version of Gibbs sampling. The iteration step is as follows (Robert CP, Casella G: Monte Carlo Statistical Methods. Springer 1999):

$$\begin{array}{lll} \text{Step 1:} & x_t \mid z_{t-1} & \sim \mathrm{U}\Big(L, \mu + \sqrt{-2\sigma^2 \log(\sigma z_{t-1}\sqrt{2\pi})}\Big) \\ \text{Step 2:} & z_t \mid x_t & \sim \mathrm{U}\Big(0, \frac{1}{\sqrt{2\pi}\sigma} \exp(-0.5(x_t-\mu)^2/\sigma^2)\Big). \end{array}$$

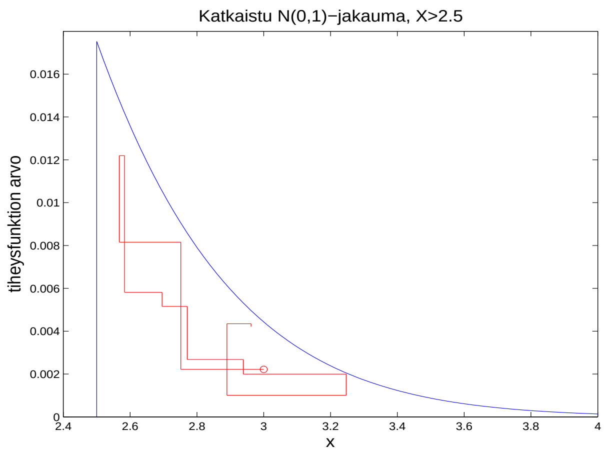
After some iterations, $(t = 1, 2, 3, \ldots)$, we obtain simulated draws $x_1, x_2, x_3, \ldots$. In these iterations, variable z is purely *instrumental variable*. In fact, slice sampling relies on *completion* of the target density $\pi(x)$ into $\pi(x, z)$, so that

$$\int \pi(x, z)\, \mathbf{d}z = \pi(x).$$

In this example:

$$\pi(x, z) \propto 1_{\{x>L\}} 1_{\{0<z<\frac{1}{\sigma\sqrt{2\pi}} \exp(-0.5(x-\mu)^2/\sigma^2)\}},$$

which is a 2-dimensional uniform density over the area in x, z-plane below the density function $\pi(x)$, in the region where $x > L$.



6.8.2 Gibbs sampling

Gibbs sampling is also known as *alternating conditional sampling*. Slice sampling is a special case of Gibbs sampling. To demonstrate Gibbs sampling, consider a simple 2D normal density:

$$\begin{bmatrix} X \\ Y \end{bmatrix} \sim N\Big(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix}\Big)$$

Recall that the 2D normal density function (mean zero, unit variance) is

$$\pi(x,y) = \frac{1}{2\pi\sqrt{1-\rho^2}} \exp\Big(- \frac{1}{2(1-\rho^2)}(x^2 - 2\rho xy + y^2)\Big).$$

It can be written in the form $\pi(x \mid y)\pi(y)$ or $\pi(y \mid x)\pi(x)$ since the marginal, and conditional, densities can be solved from the joint density:

$$\pi(x) = \int_{-\infty}^{\infty} \pi(x,y)\mathbf{d}y = N(0,1)$$

$$\pi(y) = \int_{-\infty}^{\infty} \pi(x,y)\mathbf{d}x = N(0,1)$$

and

$$\pi(y \mid x) = \frac{\pi(x,y)}{\pi(x)}$$

$$= \frac{\frac{1}{2\pi\sqrt{1-\rho^2}} \exp\Big(- \frac{1}{2(1-\rho^2)}(x^2 - 2\rho xy + y^2)\Big)}{\frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}x^2)}$$

$$= \frac{1}{\sqrt{2\pi}\sqrt{1-\rho^2}} \exp(-\frac{1}{2(1-\rho^2)}(\rho x - y)^2) \quad = \quad \mathrm{N}(\rho x, 1-\rho^2)$$

and similarly: $\pi(x \mid y) = \mathrm{N}(\rho y, 1-\rho^2)$.

Monte Carlo sampling from $\pi(x,y)$:

Sample x from $\pi(x) = \mathrm{N}(0,1)$, then sample y from $\pi(y \mid x) = N(\rho x, 1-\rho^2)$. Continue until required sample is collected. Each sampled pair of (x, y) is an independent draw from the joint distribution. (Same can be done by sampling first y from $\pi(y) = \mathrm{N}(0,1)$, and then x, given y).

Gibbs sampling from $\pi(x, y)$:

Pick some initial value for both $(x, y) = (x_0, y_0)$. Then, sample x_1 from $\pi(x \mid y_0)$. After that, sample y_1 from $\pi(y \mid x_1)$. Continue alternating in this way, sampling the next value of x (or y) given the previous sampled value of y (or x), until required sample is collected. Each sampled pair of (x, y) is NOT independent of the previous sampled values. In the long run, the empirical distribution of the large sample will approximate the target density. If the initial values (x_0, y_0) were in the fringe of the target density, it may take many steps before the sampled values represent the target density.

Gibbs sampling algorithm for d-dimensional density:

Assume we have a d-dimensional unknown vector $\theta_1, \ldots, \theta_d$, with the target density $\pi(\theta_1, \ldots, \theta_d \mid X)$. Gibbs sampling is based on solving the **full conditional** densities (fcd) so that these can be easily sampled at each iteration step t:

$$\pi(\theta_j \mid \theta_{-j}^{t-1}, X),$$

where θ_{-j}^{t-1} represents all elements of vector θ at iterations step $t-1$, apart from the single element θ_j.

$$\theta_{-1}^{t-1} = (\theta_1^t, \ldots, \theta_{j-1}^t, \theta_{j+1}^{t-1}, \ldots, \theta_d^{t-1}).$$

Each iteration step consists of a cycle, in which every element θ_j is sampled in turn, from the full conditional distribution, that is conditional to all other elements θ_k, $k \neq j$ at their current values. Of course, the distributions are all conditional to the data X, (if our aim is to sample from a posterior).

6.8.3 Metropolis-Hastings algorithm

This is a very general tool that can be used for simulating from complicated distributions, as long as we check it's working properly. For the algorithm, we need to choose a *proposal distribution* Q which is used to generate a proposed value x^* for the next round of iteration, that depends on the values at the previous step. The proposal density is thus of the form $Q(x^* \mid x_{i-1})$. The proposed value becomes accepted with probability

$$r = \min\Big(\frac{\pi(x^* \mid \mathrm{data}) Q(x_{i-1} \mid x^*)}{\pi(x_{i-1} \mid \mathrm{data}) Q(x^* \mid x_{i-1})}, 1\Big).$$

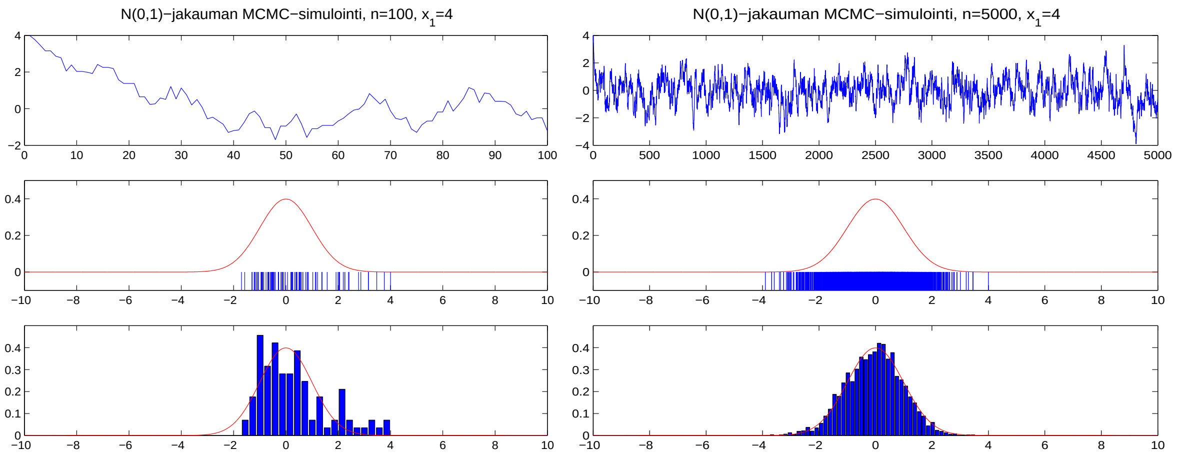
If it becomes accepted, it will be the new generated value for x at this current iteration step. Otherwise, the previous value remains also for the current step. As seen from the acceptance probability formula, it is sufficient that we are able to compute the posterior density **without normalizing constant**. All terms that are constants with respect to x will cancel out from the ratio. The algorithm can be slow if the proposal density is badly chosen, either too narrow or too wide. In practice, the acceptance probability is often computed in logarithmic form which makes fractions and products into a summations that are easier to compute (to avoid numerical under-/overflow due to divisions of possibly small probability densities). As a special case, we obtain Gibbs-sampler where the acceptance probability is always one.

6.8.4 Metropolis algorithm

Metropolis algorithm is a special case of M-H algorithm, in which the proposal density Q is symmetric. Therefore, it cancels out from the acceptance probability.

Example: MCMC simulation of N(0,1)

Assume the target distribution is $\mathrm{N}(0,1)$ and choose the proposal density as uniform distribution centered around the previous value $\mathrm{U}(x_{i-1} - L/2, x_{i-1} + L/2)$. The value of the proposal density is now simply constant $(1/L)$ and we only need to compute the ratio of two normal densities $\mathrm{N}(x^* \mid 0,1)/\mathrm{N}(x_{i-1} \mid 0,1)$.



The algorithm in R could be written as ($x_1 = 3, L = 2$, number of iterations $n = 10000$):

```
x<- 3; L <- 2; n<-10000
for(i in 2:n){
xprop <- runif(1,x[i-1]-L/2,x[i-1]+L/2)
aratio <- dnorm(xprop,0,1)/dnorm(x[i-1],0,1)
 rnd <- runif(1)
 x[i]<-(rnd<aratio)*xprop + (rnd>=aratio)*x[i-1]
}
hist(x,100,freq=FALSE)
xx<-seq(-5,5,by=0.01)
points(xx,dnorm(xx,0,1),'l',col='red')
```

Try this with different widths `L` and iterations `n`.