107.330 Statistische Simulation und computerintensive Methoden

Random Number Generation

Alexandra Posekany

WS 2020

Random sample

The concept of random sample is one of the cornerstones of statistics

Definition

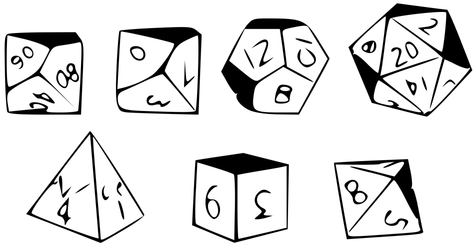
A collection of random variables $x_1, \ldots, x_n$ is called a independent and identically distributed (iid) random sample if they are all independent and have all the same probability distribution.

The concept is quite simple, but to create a collection of numbers that behave like realized values of a random sample turns out to be a challenging problem.

Using physical processes

To create **real** random numbers usually physically processes are used, like for example

- flipping a coin, rolling a dice, roulette, . . .
- using decaying rates of a radioactive source
- noise from the atmosphere



Disadvantages of physical processes for random number generations

Using physical processes for random number generation is usually difficult, because

- they are usually expensive to create
- reproducibility is difficult
- one can actually not be sure if it is really random and iid

For example tossing a coin often might damage the coin. Or unobserved and uncontrollable processes might have an impact on the physical process of interest. For example when using gamma rays hits on a detector might be occasionally be influenced by changes in the magnetic field.

Using physical processes conclusions

Due to the reasons above physical processes are usually not used for random number generation in a statistical context.

However in other areas like cryptography and slot machines and so on they are often compulsory.

Sampling from finite populations in R

A special case for random sampling in R is the function `sample`. It can sample from a finite sample with or without replacement. Hence it can also used to sample from a multinomial distribution or to permute a vector. Some short examples:

- Coin toss:

```
> sample(0:1, size = 10, replace = TRUE)
 [1] 1 1 0 1 1 1 0 0 0 0
```

- Lottery draw:

```
> sample(1:45, size = 7, replace = FALSE)
[1] 36 22 45 31 12 25  3
```

Sampling from finite populations in R II

- Permutation:

```
> options(width = 55)
> sample(LETTERS)
 [1] "G" "A" "J" "D" "Z" "O" "K" "Y" "C" "R" "Q" "V"
[13] "W" "F" "B" "N" "L" "H" "E" "M" "P" "S" "I" "U"
[25] "T" "X"
```

- Multinomial distribution:

```
> rbind(samples=samples<-table(sample(1:4, size=250,
+ replace=TRUE,prob=c(0.2,0.4,0.3,0.1))),
+ frequencies=samples/250,
+ probabilities=c(0.2,0.4,0.3,0.1))
                   1      2      3      4
samples       39.000 94.000 84.000 33.000
frequencies    0.156  0.376  0.336  0.132
probabilities  0.200  0.400  0.300  0.100
```

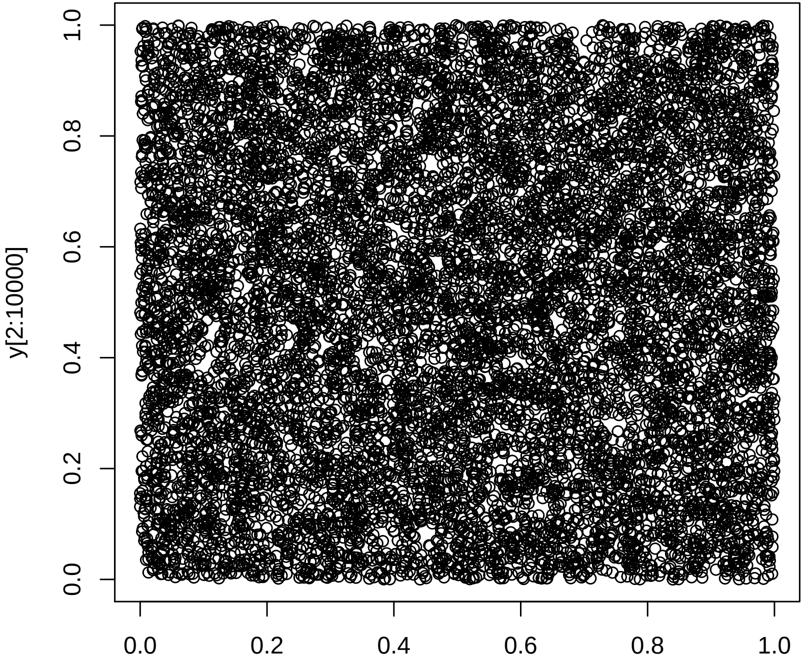
Real random numbers in R

One can however still access real random numbers in R using the R package `random`.

The `random` package is an interface to the webpage https://www.random.org/ which provides random numbers based on atmospheric noise.

```
> library(random)
> x <- randomNumbers(n=10000, col=1, min=0,
+                    max=100000, check=TRUE)
> range(x)
> y <- x/100000
> range(y)
```

Real random numbers in R II



Pseudo-random number generator

The solutions used on computers in the context of statistics especially to create random numbers are pseudo-random number generators (PRNG).

The development of PRNGs has a long history and many different ones were suggested. In this course only basic algorithms and principles will be discussed.

But let us first discuss the word pseudo in PRNG. Basically all PRNG have the following idea:

1. start with an initializing value, usually called the seed.
2. Start from the seed an recursive process that produces a sequence of numbers of the required length.

This means that for a given seed the sequence is actually deterministic and therefore reproducible. Nevertheless the sequence will look random and it should be difficult to predict the next number in a sequence without knowing the seed and the recursive rules used.

Pseudo-random number generator II

A more formal definition of of PRNG:

Definition

A PRNG is an algorithm that, starting from an initial seed (or seeds) produces a sequence of numbers that behaves as if it were a random sample from a particular probability distribution when analyzed using statistical goodness-of-fit tests.

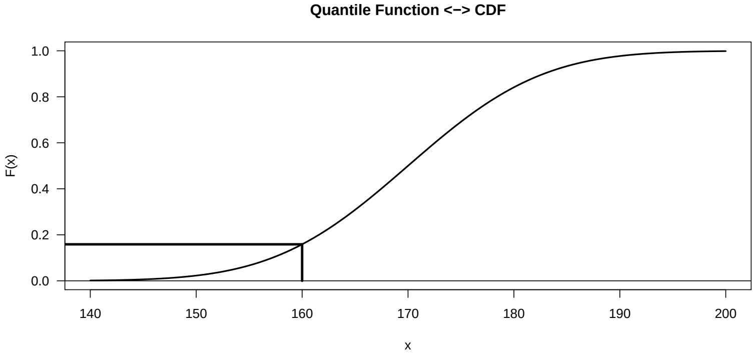
Although there are many probability distributions that are of practical interest, PRNGs focus mainly on uniform distributed random variables coming from the interval [0,1], denoted $\mathit{unif}\,[0,1]$. However, any other probability distribution can be derived from uniform random numbers.

Role of $unif[0,1]$ for random number generation

Let X be a random variable with cumulative distribution function (cdf) F_X is defined as

$F_X(x) = \mathbb{P}(X \leq x)$.

The quantile function or inverse cdf is then defined as $F_X^{-1}(u) = \inf\{x : F_X(x) \geq u\}$

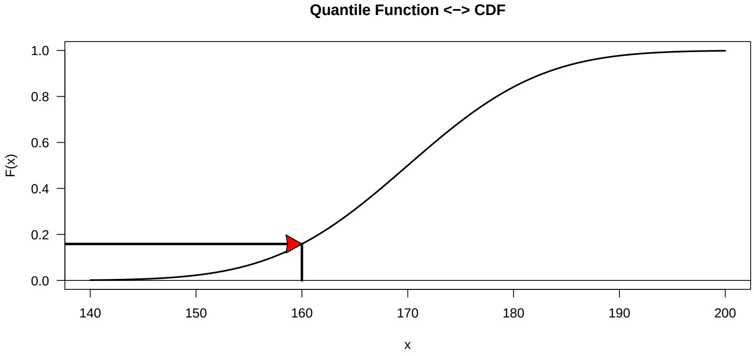


probability integral transform

The major role of uniform random variables is then formalized in the following Lemma, also know as the probability integral transform:

Lemma

Let $U \sim unif[0,1]$, then the random variable $F^{-1}(U)$ has the distribution F.



Inversion method

Hence if we want a random sample having cdf F_X, the inversion method has the following steps:

1. Compute the quantile function F_X^{-1}.
2. Generate a $u \sim \mathit{unif}[0, 1]$.
3. Make the transformation $x = F_X^{-1}(u)$

Therefore, if a good PRNG for $\mathit{unif}[0, 1]$ is available it is easy to generate random variables having other cdfs, as long as we can invert the cdf!

Congruential random number generator

There are many PRNGs to generate uniform random numbers. We will discuss in this context only the so called linear congruential random number generator.

The main idea of the linear congruential random number generator is to define a sequence based on the linear formula

$$x_{n+1} = (a \cdot x_n + c) \pmod{m}$$

Linear Congruential Random Number Generation Algorithm

For the algorithm we choose

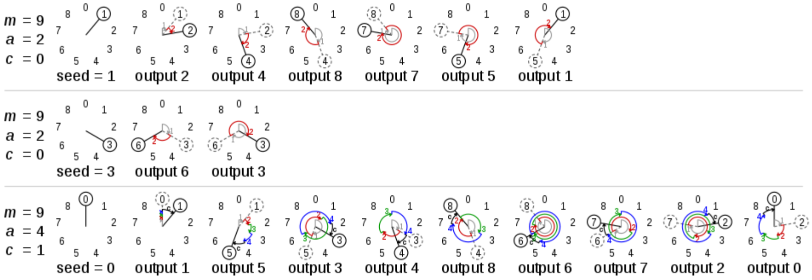
- a large integer the **"modulus"** m,
- another integer the **"multiplier"** a which is usually close to the square root of m and
- another integer the **"increment"** c with $0 \leq c \leq m$

The method has then the following steps:

1. choose an initial integer x_0 as **"starting value"** or **"seed"**.
2. compute $x_1 = (a \cdot x_0 + c) \pmod{m}$.
3. compute $u_1 = x_1/m$.
4. compute $x_2 = (a \cdot x_1 + c) \pmod{m}$.
5. compute $u_2 = x_2/m$.
6. continue with $x_{n+1} = (a \cdot x_n + c) \pmod{m}$ and $u_{n+1} = x_{n+1}/m$

The sequence $u_1, u_2, \ldots$ are uniform pseudo random numbers.

Linear Congruential Random Number Generator visualised



Linear Congruential Random Number Generator II

This methods has the following properties:

- The method depends extremely on the choice of m, a and c.
- If x_0, m, a and c are unknown to the user it is usually impossible for him/her to predict the next value.
- The method is cyclic, which means, that once we obtain $x_k = x_0$ the sequences will repeat each other.
- The largest possible cycle length is m, therefore m should be large.

Linear Congruential Random Number Generator III

An R function for this method could look for example like:

```
> mc.gen <- function(n,m,a,c=0,x0)
+     {
+     us <- numeric(n)
+     for (i in 1:n)
+         {
+         x0 <- (a*x0+c) %% m
+         us[i] <- x0 / m
+         }
+     return(us)
+     }
```

Linear Congruential Random Number Generator IV

Let us look at two examples:

```
> round(mc.gen(8,17,3,0,2),4)
[1] 0.3529 0.0588 0.1765 0.5294 0.5882 0.7647 0.2941
[8] 0.8824
> round(mc.gen(8,29241,171,41,3),4)
[1] 0.0189 0.2412 0.2412 0.2412 0.2412 0.2412 0.2412
[8] 0.2412
```

Hence we can see from the second example that the performance depends heavily on the choice of m, a and c. In general it is recommended that m is a prime number to avoid complications as in the second case where a=171 is a divisor of m=29241.

In R the function `runif` can be used to generate (pseudo) random numbers from a uniform distribution. The functions however uses a more sophisticated version of the PRNG.

Transformation methods

Assume the target distribution f is linked in a simple way to other distributions from which it is simple to draw random numbers. Then it is often easy to write algorithm which exploit this relationship to simulate variables coming from f.

Such algorithms are then called transformation methods for which of course the relationships between different distributions need to be known.

The following slides list first some distributions and their densities and then some relations.

Distributions I

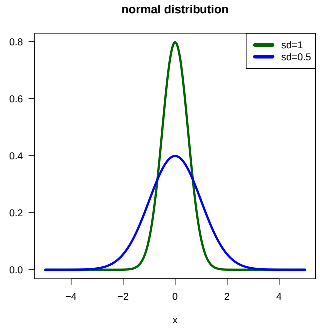
Normal distribution: $f(x;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$ is the density of the normal distribution denoted as $N(\mu,\sigma^2)$, where $\mu \in \mathbb{R}$ and $\sigma \in \mathbb{R}^+$.

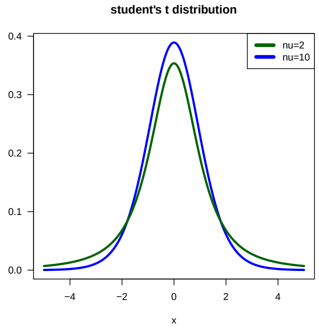
student's t-distribution: $f(x;\nu) = \frac{\Gamma((\nu+1)/2)}{\sqrt{2\nu}\Gamma(\nu/2)}\left(1+\frac{x^2}{\nu}\right)^{-\frac{\nu+1}{2}}$ is the density of the t-distribution with ν degrees of freedom, denoted as t_ν, where $\nu \in \mathbb{N}^+$. For $\nu = 1$ the distribution is called Cauchy distribution.

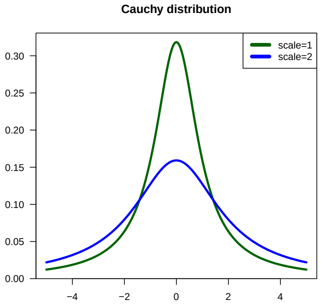
Cauchy distribution: $f(x;\gamma) = \dfrac{1}{\pi\gamma\left(1+\frac{x^2}{\gamma^2}\right)}$ is the density of the Cauchy-distribution with scale parameter γ.

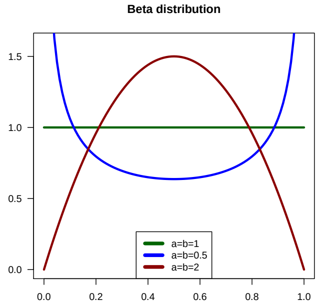
Beta distribution: $f(x;\alpha,\beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}$ is the density of the Beta distribution denoted as $Be(\alpha,\beta)$, where $x \in [0,1]$, $\alpha \in \mathbb{R}$ and $\beta \in \mathbb{R}^+$.

Distributions









Distributions II

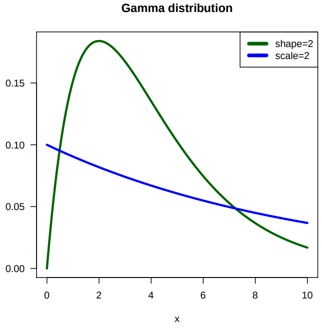
Gamma distribution: $f(x;\alpha,\beta)=\frac{\beta^\alpha}{\Gamma(\alpha)}x^{\alpha-1}e^{-\beta x}$ is the density of the Gamma distribution denoted as $Ga(\alpha,\beta)$, where $x\geq 0$, the shape parameter $\alpha\in\mathbb{R}$ and the scale parameter $\beta\in\mathbb{R}^+$.

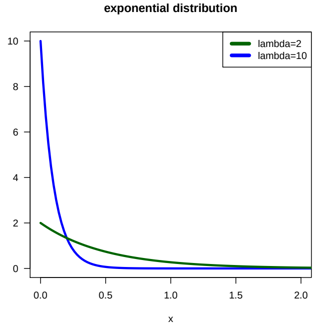
Exponential distribution: $f(x;\lambda)=\lambda e^{-\lambda x}$ is the density of the exponential distribution denoted as $Exp(\lambda)$, where $x\geq 0$ and $\lambda\in\mathbb{R}^+$.

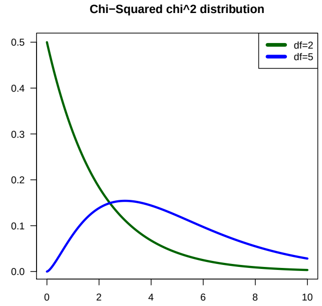
Chi-square distribution: $f(x;k)=\frac{x^{k/2-1}e^{-x/2}}{2^{k/2}\Gamma(k/2)}$ is the density of the chi-square distribution with k degrees of freedom, denoted as χ^2_k, where $x\geq 0$ and $k\in\mathbb{N}^+$.

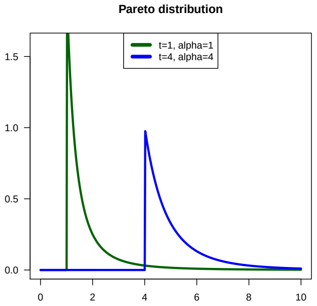
Pareto distribution: $f(x;\alpha,x_0)=\alpha x_0^\alpha x^{-(\alpha+1)}$ is the density of the Pareto (Type I) distribution with tail index α, denoted as $Pa(\alpha,x_0)$, where $x\geq x_0$ and $\alpha\in\mathbb{R}^+$.

Distributions









Relations between Distributions visualised

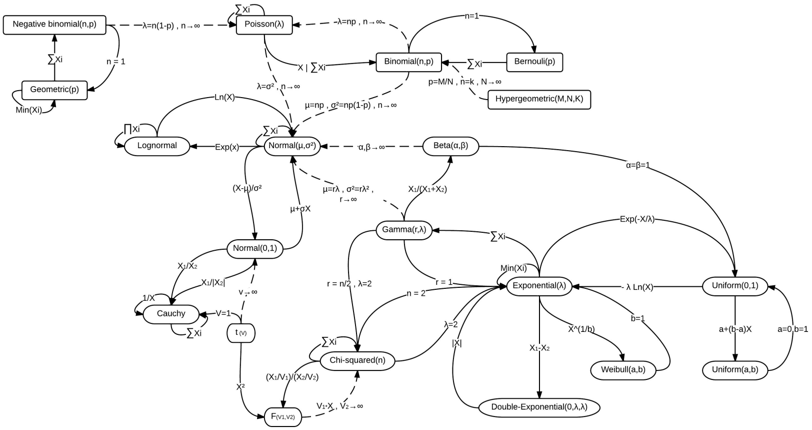


Figure 1: Source: www.math.wm.edu/∼leemis/2008amstat.pdf

Important Relations between Distributions

Some useful relations between distributions are:

- Let $x \sim N(\mu, \sigma^2)$, then $y = e^x$ is log-normal distributed.
- Let $x_1, \ldots, x_k$ be k independent standard normal random variables. Then $y = \sum_{i=1}^{k} x_i^2$ has a χ_k^2 distribution.
- $Ga(1, \lambda)$ is a $Exp(\lambda)$ distribution.
- $Ga(k/2, 1/2)$ is a χ_k^2 distribution.
- Let $x_1 \sim Ga(\alpha, 1)$ and $x_2 \sim Ga(\beta, 1)$, then $y = \frac{x_1}{x_1 + x_2}$ has a $Be(\alpha, \beta)$ distribution.
- Let $x \sim N(0, 1)$ and $\nu z^2 \sim \chi_\nu^2$ then $y = \frac{x}{z}$ has a t_ν distribution.
- Let $x_1 \sim \chi_{\nu_1}^2$ and $x_2 \sim \chi_{\nu_2}^2$ be independent. Then $y = \frac{x_1/\nu_1}{x_2/\nu_2}$ has a F-distribution with (ν_1, ν_2) degrees of freedom.
- Let $x \sim Exp(\alpha)$ then $y = y_0 e^x$ has a $Pa(\alpha, y_0)$ distribution.

Random variables and distributions in R

For many common distributions R provides easy to use functions which usually can compute the density, cdf and quantiles and can generate random samples.

The function names are all quite logical and have for a distribution with R name `foo` the following structure:

purpose:	density	cdf	quantiles	random numbers
function name:	<u>d</u>foo	<u>p</u>foo	<u>q</u>foo	<u>r</u>foo

For example the normal distribution has the R name `norm` and hence the corresponding function names are `dnorm",pnorm,qnormandrnorm`.

Other methods

However what to do if random variables are needed from a distribution where the inverse function cannot be computed in closed form or if we do not now any transformations which give the desired sampling distribution?

One could of course approximate the inverse function and use numerical methods but there are still plenty of other methods which might be better solutions for such problems.

The only method discussed in this course is the so-called acceptance-rejection method.

Basis of the acceptance-rejection method

The basis of the acceptance-rejection method is the following theorem:

Theorem

Let f and g be two density functions where there exists a constant c such that $f(x) \leq cg(x)$ for all x. Let Y have density g and let $U|Y = y$ have a uniform distribution on $[0, cg(y)]$. Then $X = Y|U < f(Y)$ has density f.

Acceptance-rejection method

Assume the target density is f and one can generate random numbers from a so-called instrumental or proposal density g with the relation

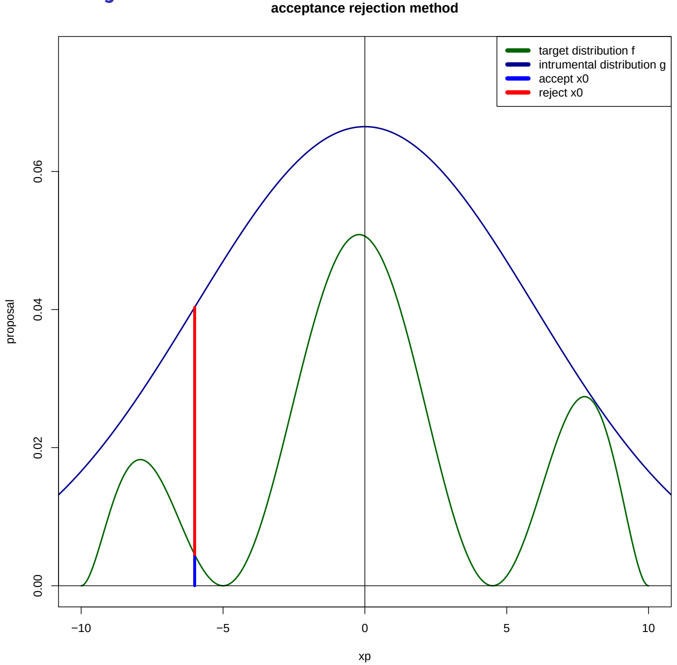
$f(x) \leq cg(x)$.

The acceptance-rejection method has then the following steps:

- Generate a random number y coming from g and a u coming from $unif[0,1]$.
- Accept $x = y$ if $u \leq f(x)/(cg(x))$.
- If $u > f(x)/(cg(x))$ reject $x = y$ and start over.

The accepted value x comes then from density f. To obtain n random numbers, the method has to be repeated until n values have been accepted.

Acceptance Rejection Method Visualised



Properties of the acceptance-rejection method

The acceptance rejection methods has the following properties:

- The constant c is always larger than 1.
- The probability of accepting a proposal is exactly $1/c$. Thus if different instrumental densities are available the one should be chosen which needs the smallest value c.
- To obtain n sample from f requires then roughly cn random samples from g and from $unif$.
- This method can actually also be used, if the density f is only known (needed) up to a normalizing constant which is often the case in Bayesian statistics. We will revisit acceptance-rejection sampling in the Bayesian context as Metropolis-Hastings sampling.

Generating normal distributed random variables

In the following slides we will use the normal distribution as an example how different strategies look like to obtain random samples from this distribution.

We will discuss the following three approaches:

- Using an approximation of the inverse cdf.
- Using an acceptance-rejection algorithm.
- Using a transformation method - Box-Muller algorithm.

“Inverse method” to generate normal random variables

The cdf of the standard normal distribution

$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-z^2/2} dz$

cannot be expressed explicitly, hence one has to use approximations to $\Phi(x)$, $\Phi(x)^{-1}$, when applying the inverse method.

A know approximation is related to the Taylor polynomial of the exponential function

$\Phi(x) \approx 1 - f(x)[b_1 \cdot t + b_2 \cdot t^2 + b_3 \cdot t^3 + b_4 \cdot t^4 + b_5 \cdot t^5] \quad (x > 0)$,

where $f(x)$ is the normal density, $t = (1 + px)^{-1}$ and
$p = 0.2316419$, $b_1 = 0.31938$, $b_2 = -0.35656$, $b_3 = 1.78148$,
$b_4 = -1.82125$ and $b_5 = 1.33027$.

“Inverse method” to generate normal random variables II

Similarly we have the approximation by the rational function

$\Phi(a)^{-1} \approx t - \frac{a_0 + a_1 \cdot t}{1 + b_1 \cdot t + b_2 \cdot t^2}$,

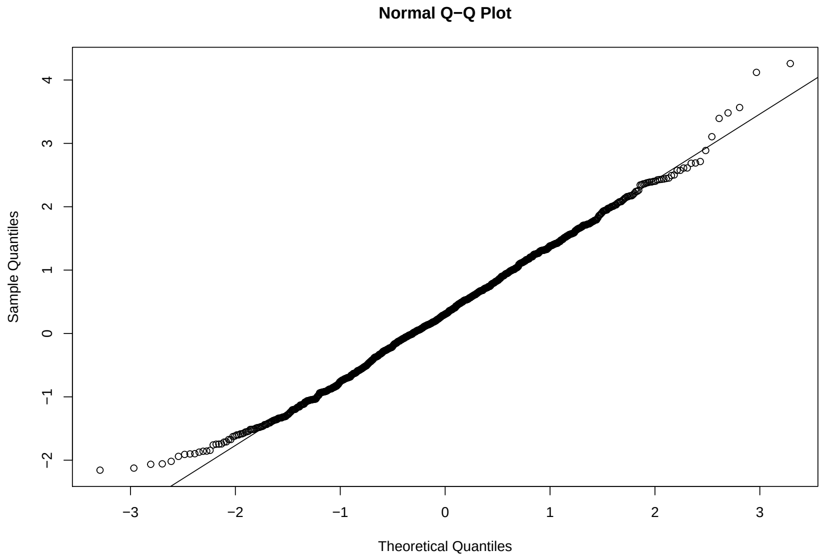
where $t^2 = \log(a^{-2})$, $a_0 = 2.30753$, $a_1 = 0.27061$, $b_1 = 0.99229$ and $b_2 = 0.04481$.

The approximations are exact up to an error of 10^{-8}.

“Inverse method” to generate normal random variables III

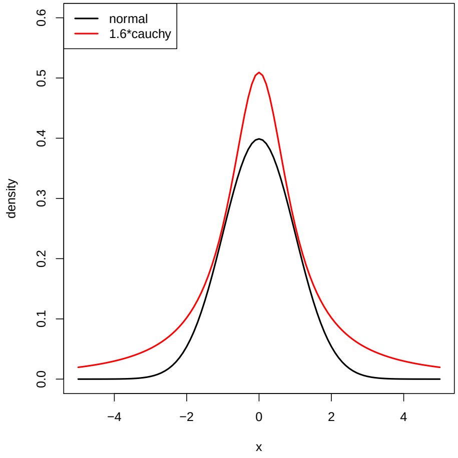
```
> InvCdfNormal <- function(x)
+     {
+     a0 <- 2.30753
+     a1 <- 0.27061
+     b1 <- 0.99229
+     b2 <- 0.04481
+
+     t2 <- log(x^{-2})
+     t1 <- sqrt(t2)
+
+     t1 - (a0 - a1*t1)/(1+b1*t1+b2*t2)
+     }
> RnormalInverse <- function(n)
+     {
+     us <-runif(n)
+     InvCdfNormal(us)
+ }
```

“Inverse method” to generate normal random variables IV



Acceptance-rejection to generate normal random variables

Assume that we have an easy way to obtain random draws from a Cauchy distribution. Then we can use the Cauchy distribution as an instrumental density to draw from the target distribution (normal) with $c = 1.6$.

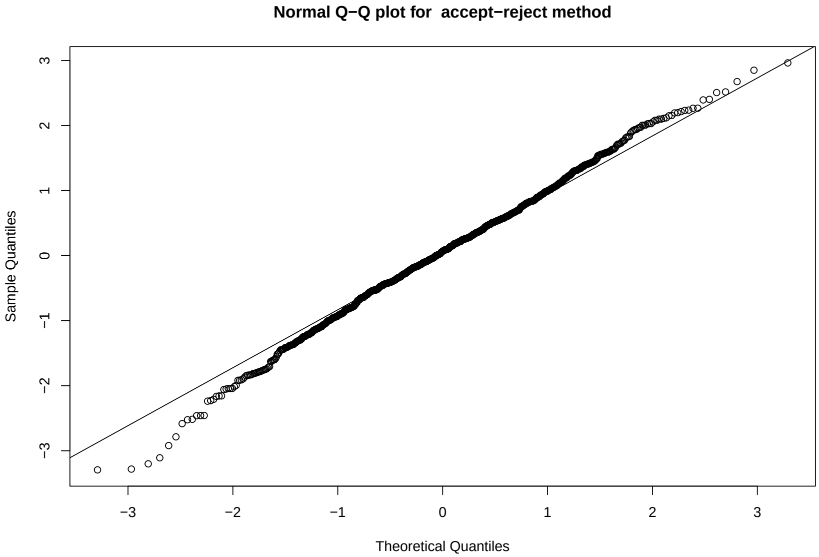


Acceptance-rejection to generate normal rvs II

```r
> n <- 1000; iter <- 0; accepted <- 0
> x <- numeric(n)
> while(accepted<n)
+     {
+     u <- runif(1)
+     iter <- iter + 1
+     y <- rt(1,df=1)
+     CC <- 1.6
+     if (dnorm(y)/(CC*dt(y,1)) >= u)
+         {
+         accepted <- accepted+1
+         x[accepted] <- y
+         }
+     }
```

Acceptance-rejection to generate normal rvs III

```
> qqnorm(x, main="Normal Q-Q plot for  accept-reject method")
> qqline(x)
```



Transformation method to generate normal random variables

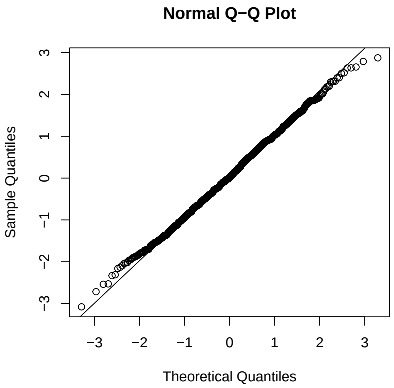
A popular transformation based algorithm to sample standard normal distributed random variables is the Box-Muller algorithm. Its motivation is that a bivariate standard normal distribution (X, Y) has the **polar coordinates** r and θ, where $r^2 = X^2 + Y^2 \sim \chi^2_2$ which is also an $Exp(0.5)$ distribution. Next, the angle is uniform on $unif[0, 2\pi]$ since the distribution is invariant under rotation. It is also easy to generate exponential distributed random variables using the inverse method (see exercises).

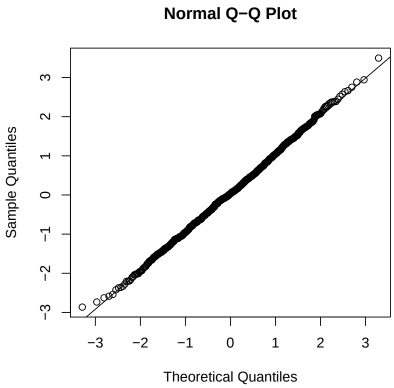
The algorithm has then the steps:

1. Generate u_1 and u_2 iid from $unif[0, 1]$.
2. $x_1 = \sqrt{-2\log(u_1)}\cos(2\pi u_2)$ and $x_2 = \sqrt{-2\log(u_1)}\sin(2\pi u_2)$
3. Take x_1 and x_2 as two iid sample from $N(0, 1)$.

Box Muller Algorithm

```
> u1<-runif(1000); u2<-runif(1000)
> x1<-sqrt(-2*log(u1))*cos(2*pi*u2)
> x2<-sqrt(-2*log(u1))*sin(2*pi*u2)
```


Normal Q−Q Plot
Sample Quantiles
Theoretical Quantiles


Normal Q−Q Plot
Sample Quantiles
Theoretical Quantiles