Computational statistics, lecture 3

Frank Miller, Department of Computer and Information Science, Linköping University

February 3, 2026

LINKÖPING UNIVERSITY

Simulation in Statistics

(Literature: Givens and Hoeting, 6; Gentle, 7.1-7.3)

- Computer-generated random variables
- Purpose:
  - Simulate a situation where a statistical model can be assumed
  - Simulate situation repeatedly to investigate properties of estimators, confidence intervals, significance tests
    - Example: power of a test in situations where assumptions are not fulfilled
  - Perform Monte Carlo integration

- Problem: Given a density *f* of a target distribution, generate random draws $X_1, \ldots, X_n$ which follow the target distribution

Random variables from familiar distributions

- Computer-generated random variables are not random but deterministic (Gentle, 2009, Ch.7.1)
- Algorithms are used such that the deterministic nature is not visible, and variables seem random

- Deterministic algorithm generates values between 0 and 1 which follow well independent draws from Unif[0,1]
- Then, random variables following other familiar distributions can be generated from Unif[0,1] and are implemented in statistical software, see Givens and Hoeting (2013), Tab. 6.1

Generating Unif[0,1] random variables

- Linear congruential generator
  - x_0 is seed
  - $x_{k+1} = (ax_k + c) \bmod m$
  - $r_k = x_k/m$ is the generated k-th random number
- Here, a, c, and m need to be chosen carefully (m large and often a prime)
- Sequences like that have a period (they repeat after some numbers)

- Another type of generator works on the bit-level, shifting 0-1-sequences
- The "Mersenne twister" is a good generator used in **R**; it is based on shifting 0-1-sequences and twisting terms by some matrix multiplication
- The period of the Mersenne twister is $p = 2^{19937} - 1$ (both p and 19937 are primes)

The seed

- If seed is fixed, the following generated sequence of random variables is fixed
- Therefore, the seed is determined often based on the system time by default
- For purposes of **reproducibility of results**, the user can choose a specific seed, which can be communicated to other users
- In **R**:

```
> set.seed(2026)
> cbind(sample(1:5), sample(6:10), sample(11:15))
```

```
     [,1] [,2] [,3]
[1,]    5    8   12
[2,]    1    9   15
[3,]    4    6   13
[4,]    2   10   11
[5,]    3    7   14
```

- Note that there was a change in the method used by the **sample**-function from **R**-version 3.6.0 (but even later versions might still use the older method if they had been updated from older versions). You can then change between versions by
set.seed(2026, sample.kind="Rejection") (new)
set.seed(2026, sample.kind="Rounding") (old, use only for reproducing older results)

Random variables of familiar distributions in **R**

- In **R**, random variables can be generated for a number of distributions, e.g:
- **rbeta, rcauchy, rchisq, rexp, rf, rgamma, rlnorm, rnorm, rt, runif, rweibull**
- **rbinom, rgeom, rhyper, rmultinom, rnbinom, rpois**

```
x <- rnorm(6, mean = 1.2, sd = 2)
x
[1]  3.8839870  2.8328797  3.5344539 -2.5464309  3.2059822  0.1872261

rbinom(25, size = 3, prob = 0.25)
[1] 1 2 0 0 0 0 0 2 3 0 0 2 1 1 0 0 1 0 1 1 2 2 1 0 0
```

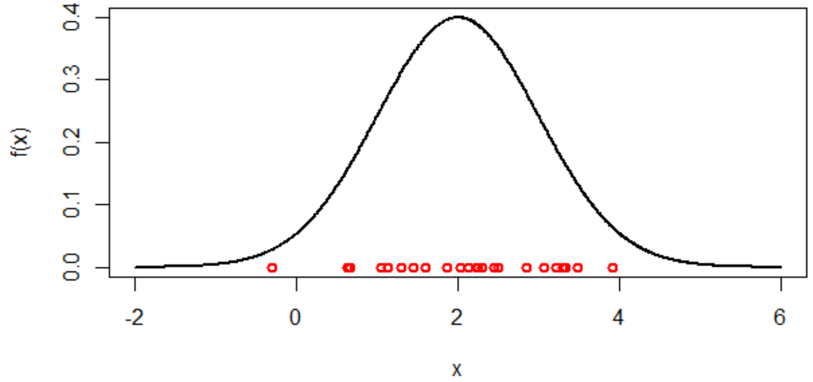
Random variables from non-familiar distributions

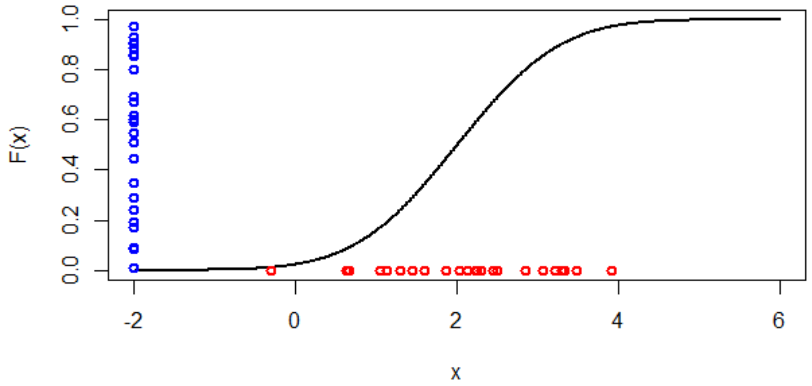
- Problem: Given a density f of a target distribution, generate random draws $X_1, \dots, X_n$ which follow the target distribution
- Now: Density f of arbitrary form

- Methods we will consider:
  - Inverse cumulative distribution function (CDF) method
  - Rejection sampling
  - Generate normal distributed variables
  - Composition sampling (use of conditional distributions)
  - Markov chain Monte Carlo (MCMC) → Lecture 4

Inverse CDF method

- Continuous random variable X with density f and distribution function F
- Then: $F(X)$ is uniformly distributed on [0,1]





- Therefore: if we can generate uniformly distributed random variables U, we can compute $X = F^{-1}(U)$ and obtain the desired sample

Inverse CDF method

- Example 1: We want to generate random variables X with triangle distribution having density

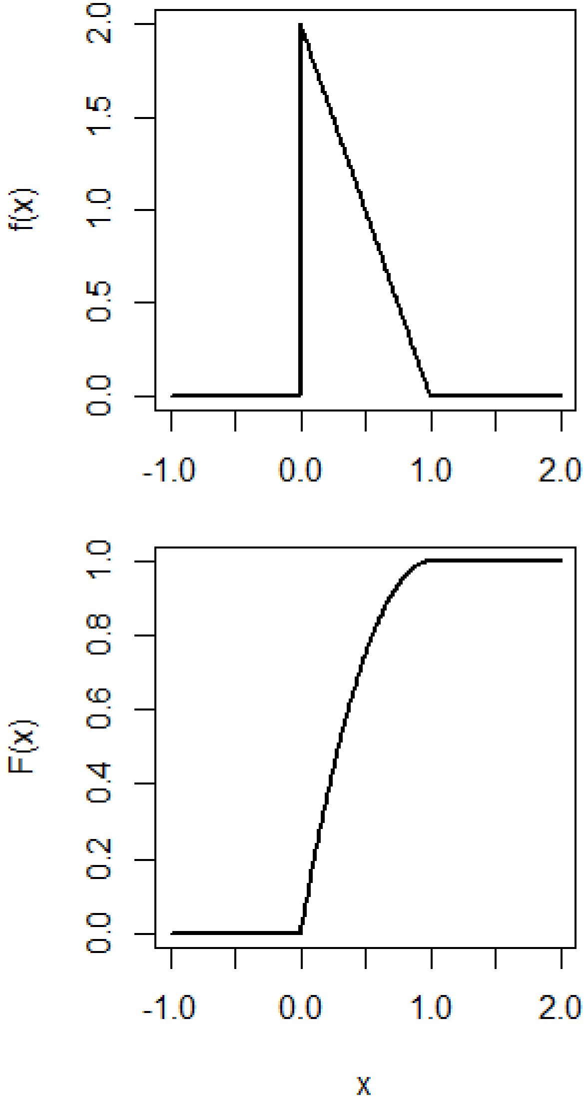
$$f(x) = \begin{cases} 2-2x, & \text{if } 0 \le x \le 1, \\ 0, & \text{otherwise} \end{cases}$$

- We compute the distribution function:

$$F(x) = \int_{-\infty}^{x} f(t)\,dt = \begin{cases} 0, & \text{if } x < 0, \\ 2x - x^2, & \text{if } 0 \le x \le 1, \\ 1, & \text{if } x > 1. \end{cases}$$

- The inverse distribution function is $F^{-1}(y) = 1 - \sqrt{1-y}$ since $y = 2x - x^2 \iff x^2 - 2x + y = 0 \iff$

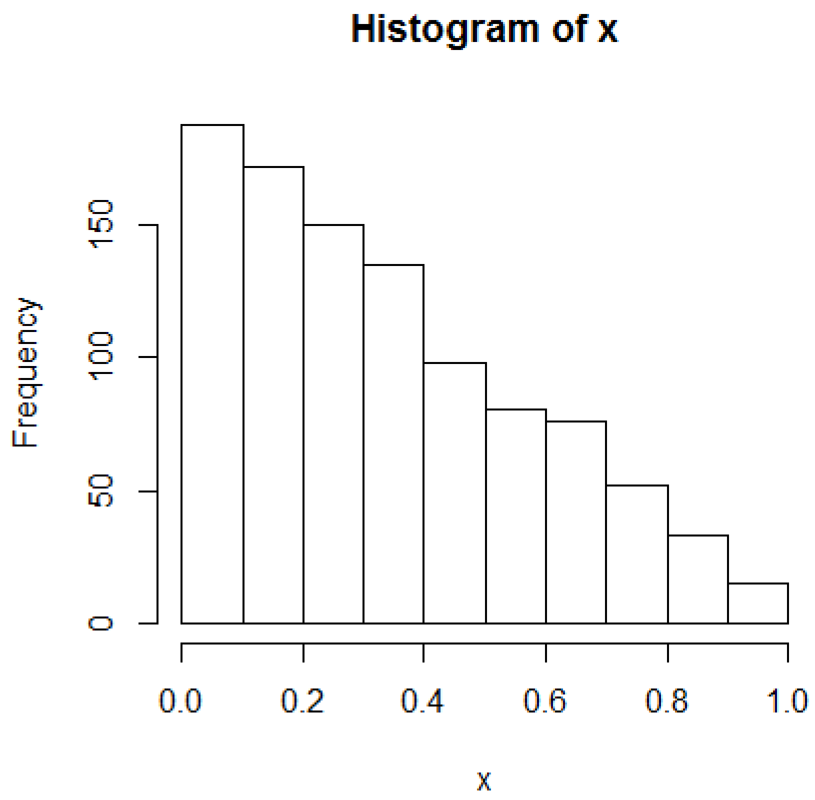
$$x_{1,2} = 1 \pm \sqrt{1-y} \Rightarrow 1 - \sqrt{1-y}$$



Inverse CDF method

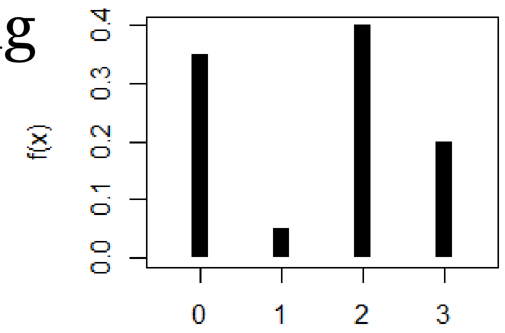
- 1000 random numbers for the triangle distribution can be generated by:

```
u <- runif(1000)
x <- 1-sqrt(1-u)
hist(x)
```

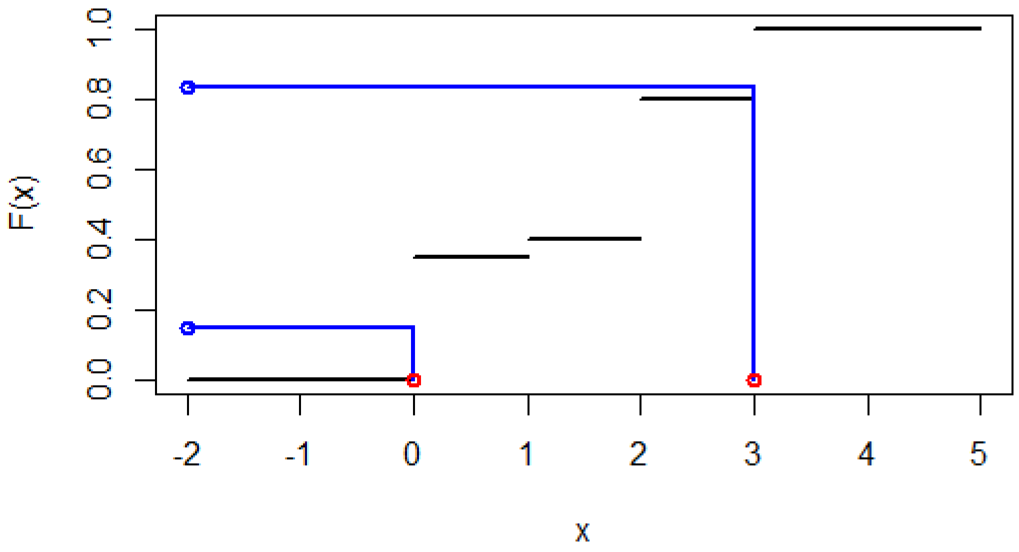


Inverse CDF method – discrete random variables

- Example 2: We want to generate a random variable X being
  0 with probability 0.35,
  1 with probability 0.05,
  2 with probability 0.4,
  3 with probability 0.2

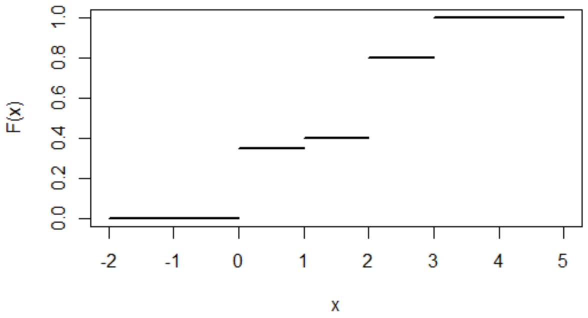


- $F(x) = P(X \leq x)$; how to apply the inverse CDF method?



Inverse CDF method – discrete random variables

- Example 2: We want to generate a random variable X being
  0 with probability 0.35,
  1 with probability 0.05,
  2 with probability 0.4,
  3 with probability 0.2



- How to apply inverse CDF method?
- Generate $U \sim \text{Unif}[0,1]$
- If $U \leq 0.35$, then $X = 0$,
  if $0.35 < U \leq 0.4$, then $X = 1$,
  if $0.4 < U \leq 0.8$, then $X = 2$,
  if $0.8 < U$, then $X = 3$.

This is 1 if the condition in (…) is true, otherwise it is 0

```
u <- runif(100000)
x <- (u>0.35)+(u>0.4)+(u>0.8)
```

Inverse CDF method

- Inverse CDF method worked well in preceding examples
- In general, drawbacks are:
  - Computation of F^{-1} might be difficult
  - Often less efficient as alternatives
  - Difficult to generalize to multiple dimensions*

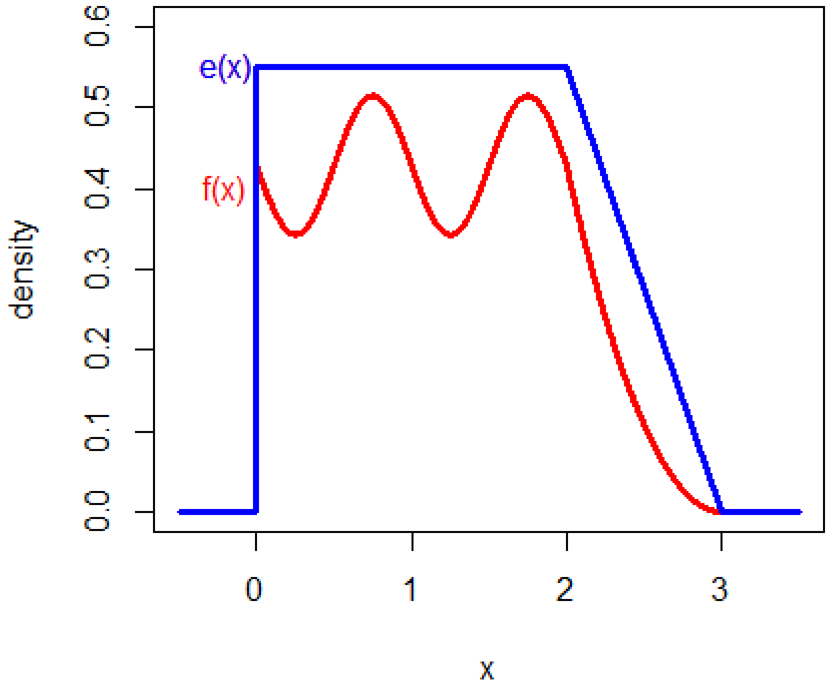
* but there is the method of *optimal transport,* which (under quadratic loss) is equal to the inverse CDF method for one-dimensional cases; it offers a multidimensional generalization

Rejection sampling

- Problem: Given a density f of a target distribution, generate random draws $X_1, \dots, X_n$ which follow the target distribution
- It can be difficult to sample with respect to f
- Situation: There is another density g which can be sampled from and which is after scaling larger than f for all x,

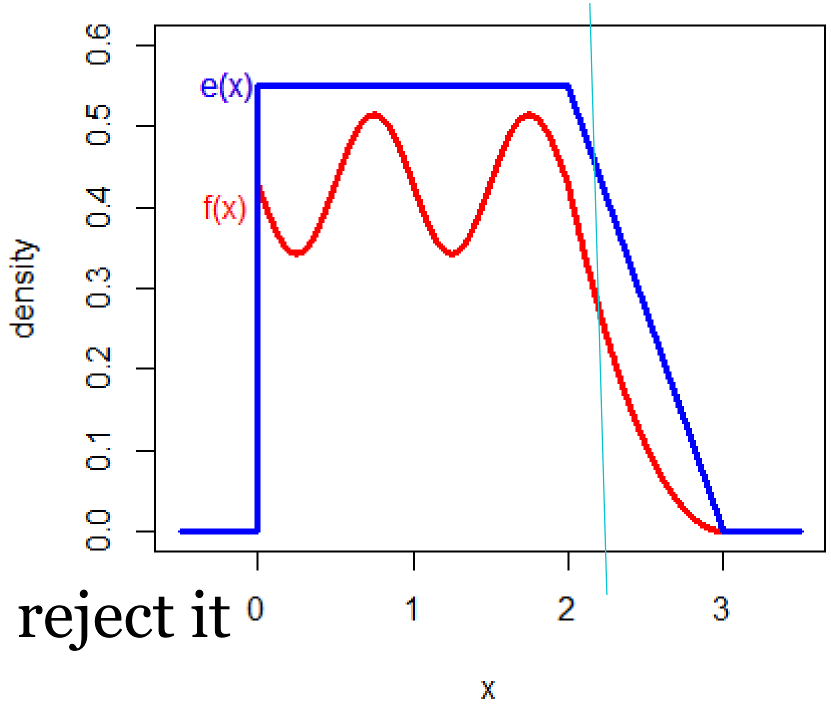
  $$e(x) = g(x)/\alpha \geq f(x)$$

  for all x and some $\alpha < 1$
- $e(x)$ is called "envelope"



Rejection sampling

- $e(x) = g(x)/\alpha \geq f(x)$ for all x and some $\alpha < 1$

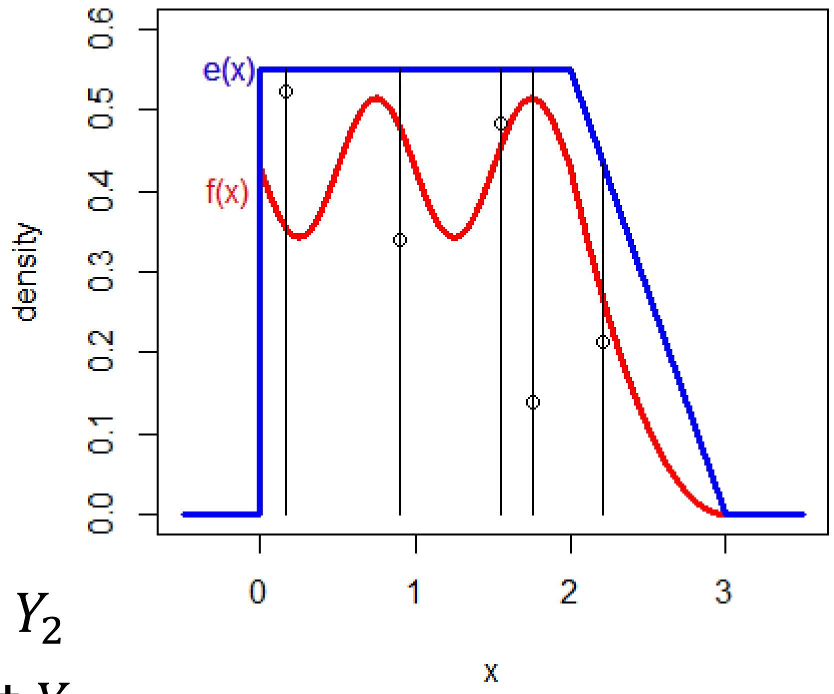


- Rejection sampling algorithm:

1. Sample $Y \sim g$
2. Sample $U \sim \text{Unif}(0,1)$
3. If $U \leq f(Y)/e(Y)$, accept Y; set $X = Y$; otherwise reject it
4. If more samples desired go to 1.

Example (for picture above): $Y = 2.21$; $f(Y) = 0.267, e(Y) = 0.435$, $f(Y)/e(Y) = 0.616$; sample U; If $U \leq 0.616$, use Y, otherwise reject it

Rejection sampling



1. Sample $Y \sim g = e\alpha$
2. Sample $U \sim \text{Unif}(0,1)$
3. If $U \leq f(Y)/e(Y)$, accept Y; set $X = Y$; otherwise rej.
4. If more samples desired, go to 1

Example (for picture above):

$(Y_1, U_1) = (2.21, 0.492)$ ➔ $U_1 < 0.616$ ➔ accept Y_1

$(Y_2, U_2) = (0.17, 0.952)$ ➔ $U_2 > f(0.17)/e(0.17)$ ➔ reject Y_2

$(Y_3, U_3) = (1.76, 0.250)$ ➔ $U_3 < f(1.76)/e(1.76)$ ➔ accept Y_3

$(Y_4, U_4) = (1.55, 0.880)$ ➔ $U_4 > f(1.55)/e(1.55)$ ➔ reject Y_4

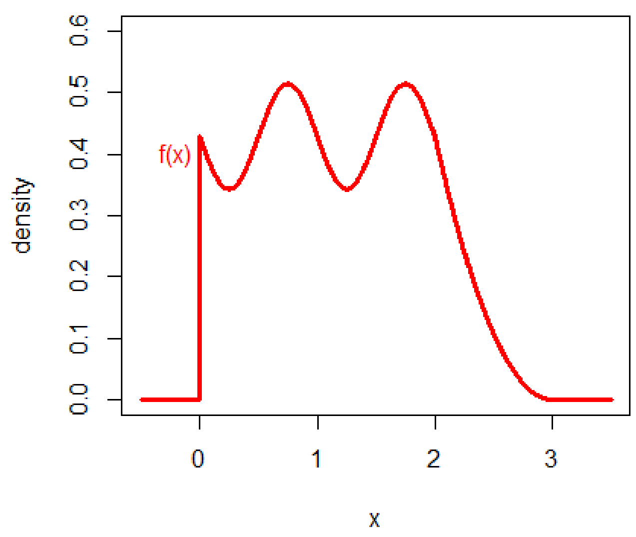
$(Y_5, U_5) = (0.90, 0.619)$ ➔ $U_5 < f(0.90)/e(0.90)$ ➔ accept Y_5 ➔ use $(2.21, 1.76, 0.90)$

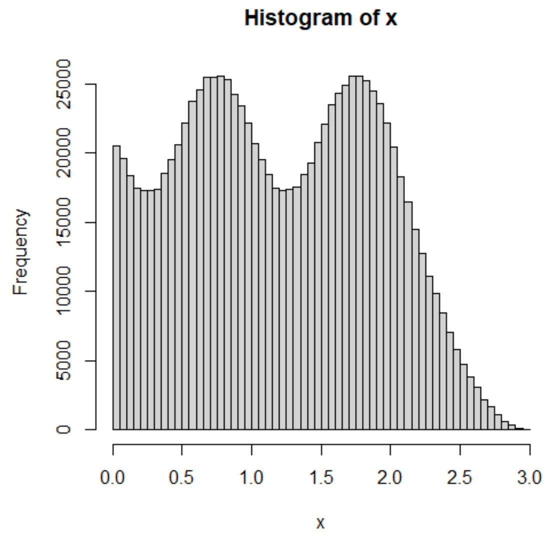
- The proportion of rejected samples is called waste; it is $1 - \alpha$

Using the generated distribution

- We have now a way to generate an arbitrary distribution
- We can use it then for different purposes
- Example 3: We have now generated a vector **x** in **R** with 10^6 values according to this distribution with density $f(x)$. What is the mean, the standard deviation, and $P(X > 2)$ for this distribution?

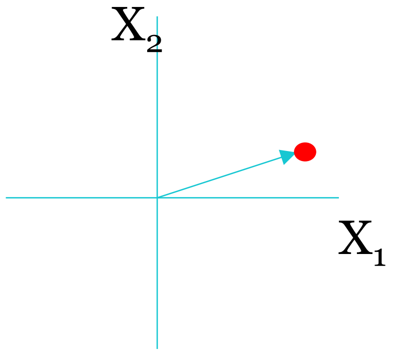
```
hist(x, breaks=60)
mean(x)
[1] 1.205569
sd(x)
[1] 0.687377
mean((x>2))
[1] 0.142867
```





Generate univariate standard normal variables

- **R**-function **rnorm** uses by default the inverse CDF method
  - Inverse distribution function Φ^{-1} of normal distribution function used; no closed formula for Φ^{-1} exists; **R** uses approximation algorithm which is very precise
- Another method: Box-Muller (can be selected in **R**, too)
  - If (X_1, X_2) standard bivariate normally distributed (i.e., X_1 and X_2 independent standard normal), then $Y = X_1^2 + X_2^2$ has χ_2^2-distribution; χ_2^2 equal to Exp(1/2)
  - Angle A uniformly distributed on $[0, 2\pi]$
  - Idea: Generate squared length Y with inverse CDF method from one uniform dist. U and angle A from an independent uniform dist.
  - $X_1 = \sqrt{-2\log(U)} \cos A \, , X_2 = \sqrt{-2\log(U)} \sin A$
- Box-Muller is around 40% slower as the default in **R**



Generate a multivariate normal distribution

- If X is multivariate normal $N(0, I)$, then $Y = A^T X + \mu$ has a $N(\mu, \Sigma)$-distribution with $\Sigma = A^T A$
- To generate $Y \sim N(\mu, \Sigma)$:
  - compute A with $\Sigma = A^T A$ (e.g. Cholesky decomposition),
  - generate $X \sim N(0, I)$ (by generating p independent $N(0,1)$-variables),
  - compute $Y = A^T X + \mu$
- Package **mvtnorm** with random generator **rmvnorm** can be used

```
library(mvtnorm)
Sig <- matrix(c(1,0.5,0.25, 0.5,1,0.5, 0.25,0.5,1), nrow = 3)
rmvnorm(5, mean = c(2, -1, 0), sigma = Sig)
          [,1]       [,2]        [,3]
[1,] 2.447613 -1.7390355 -0.43272668
[2,] 1.933296 -1.9893701  0.47161126
[3,] 2.872344 -0.6396215  0.61383902
[4,] 2.505797 -0.8080744 -0.51696135
[5,] 2.614221 -1.9280517  0.04471815
```

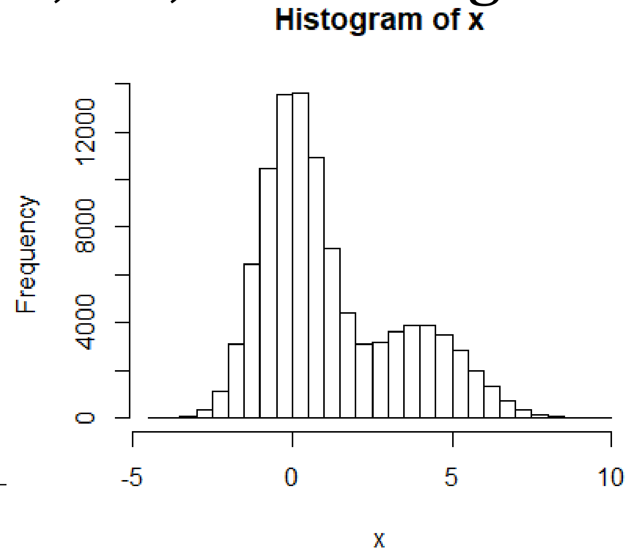
$$\mu = \begin{pmatrix} 2 \\ -1 \\ 0 \end{pmatrix}, \Sigma = \begin{pmatrix} 1 & 0.5 & 0.25 \\ 0.5 & 1 & 0.5 \\ 0.25 & 0.5 & 1 \end{pmatrix}$$

Each row is one generated random vector from the $N(\mu, \Sigma)$-distribution

Composition sampling (use of conditional distributions)

- If $f_i(x), i = 1, \dots, n$, are densities, a random variable X with density $\sum_i p_i \cdot f_i(x)$, where $\sum_i p_i = 1$, is called finite mixture; p_i=mixing parameters
- A finite mixture distribution can be generated by:
  - simulating the group-membership using the discrete distribution for mixing parameters
  - simulating the distribution of this group's distribution, i.e., simulating the conditional distribution given the group
- Ex. 4: X normal mixture of $N(0, 1)$ and $N(4, 1.5^2)$ with mixing parameter 0.7 and 0.3, respectively

```
g <- rbinom(100000, size = 1, prob = 0.3)
x <- rnorm(100000, mean = 4*g, sd = 1+0.5*g)
hist(x, breaks = 25)
```



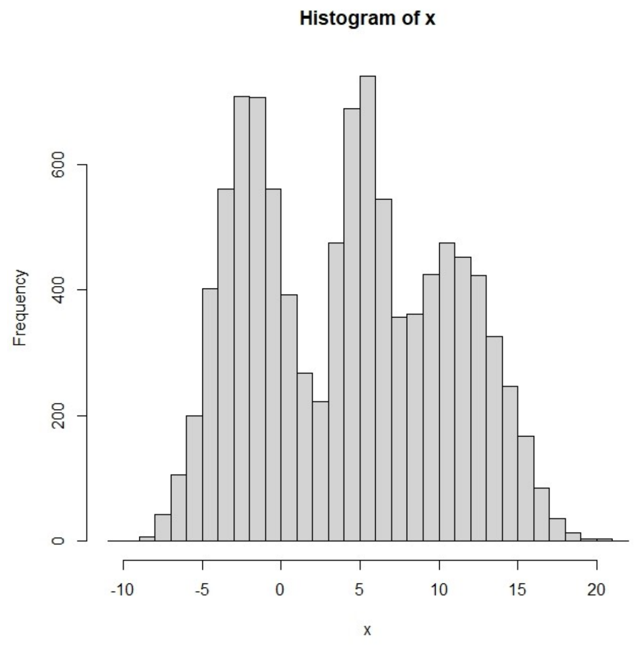
Composition sampling (use of conditional distributions)

- More flexible code for simulating a finite mixture distribution (e.g., a finite normal mixture) with composition sampling:
  - Define mean, standard deviations and mixing parameters as vector:

```
mu    <- c(-2, 5, 11)
sigma <- c(2.2, 1.4, 2.9)
prob  <- c(0.4, 0.25, 0.35)
n     <- 10000
```

  - Generate mixture by:

```
g <- sample(length(mu), n, replace=TRUE, p=prob)
x <- rnorm(n, mean = mu[g], sd = sigma[g])
hist(x, breaks = 25)
```



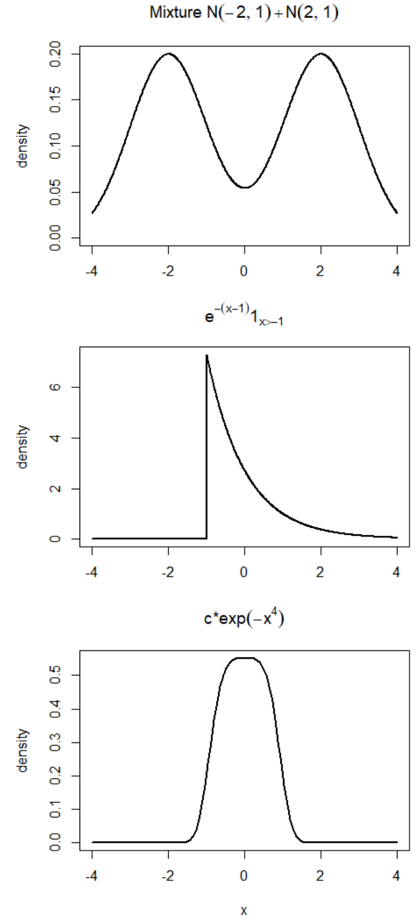
Example 5:

You have the standard functions for generation of random variables available in **R**. With which method would you generate random variables for the following distributions?

a) An equal mixture of $N(-2,1)$ and $N(2,1)$,

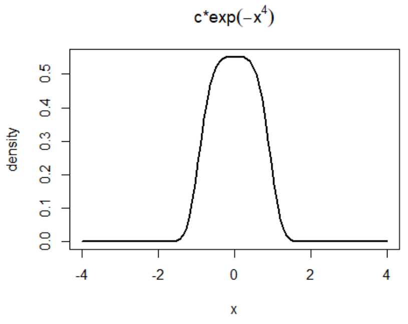
b) Distribution with density: $f(x) = e^{-(x-1)}\mathbf{1}\{x > -1\}$

c) Distribution with density: $f(x) = c\,\exp(-x^4)$,



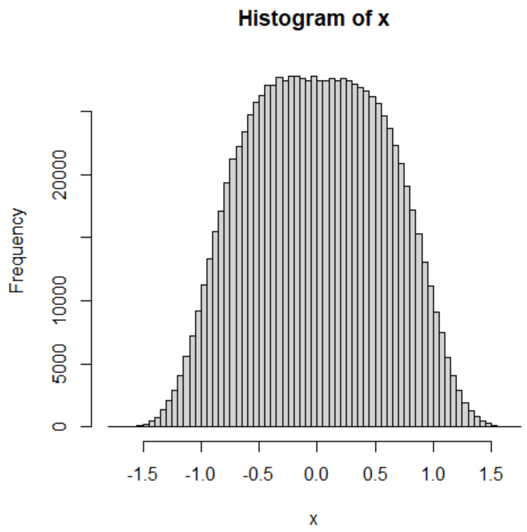
Example 6:

Assume you have now generated a sample for the distribution with density: $f(x) = c\,\exp(-x^4)$ with an appropriate method and it is stored in vector **x**.



What is the variance and kurtosis of this distribution?
Kurtosis: $E\left(\left(\frac{X-\mu}{\sigma}\right)^4\right)$, where μ is the mean and σ the standarddeviation

```
hist(x, breaks=60)
# variance
var(x)
[1] 0.337153
# kurtosis
mean(((x-mean(x))/sd(x))^4)
[1] 2.187151
```



The kurtosis is 2.19 which is less than for the normal distribution (kurt.=3). This distribution has therefore thinner tails than the normal.