HUDM 6026: Computational Statistics
Week 1: Introduction

Dobrin Marchev
Department of Human Development, TC

dm3174@tc.columbia.edu

The plan for today

- Introduction
- Random simulation, what is it good for?
- The R system of random number generation
- Tools for presentation and analysis of simulation output
- Simulation techniques
- Getting insight in classical procedures by simulation

Why simulation?

- Statistics is about random numbers
- We assume a model for generating the data
- The model contains parameters
- We estimate them from the data, using recipes
- Examples: regression, ANOVA, time series analysis, ...
- Crucial question: how good is a recipe?
- Classical (frequentist) statistics: confidence intervals
- Bayesian: combine prior and data to get posterior distribution
- Testing (mostly for zero values) is a special case

How to evaluate performance by analysis

- In some cases exact results can be obtained
- Especially for normally distributed data
- Examples: χ^2, t and F distributions (ANOVA)
- You will encounter them in other courses here
- In the old days we used tables
- Nowadays they are built in as functions in statistical software
- It is always wise to use analytical results

How to evaluate performance by simulation

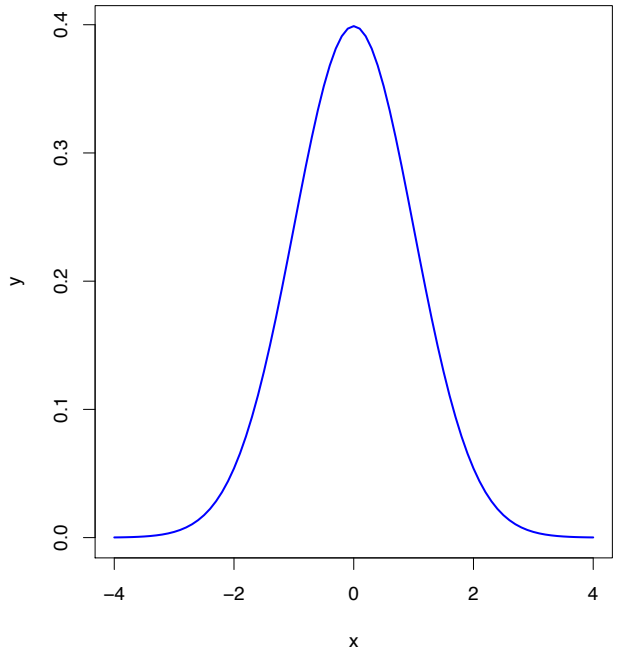
- Simulate data according to the model
- Apply the recipe
- Repeat this many times
- Study the distributions of parameter estimates
- Confidence intervals follow from them
- Probabilities of outcomes of tests can be determined
- Disadvantage: generalization can be hard
- What if I have more/less observations, larger variances, ...?
- You might have to simulate again for every case

The R system for generating random numbers

- For a given distribution you have four choices
- I use the normal distribution as an example
- Give me 100 drawings from standard normal: `rnorm(100)`
- The density over a vector x: `dnorm(x)`
- The cumulative distribution over a vector x: `pnorm(x)`
- The quantiles over a vector of probabilities p: `qnorm(p)`
- Let's take a better look

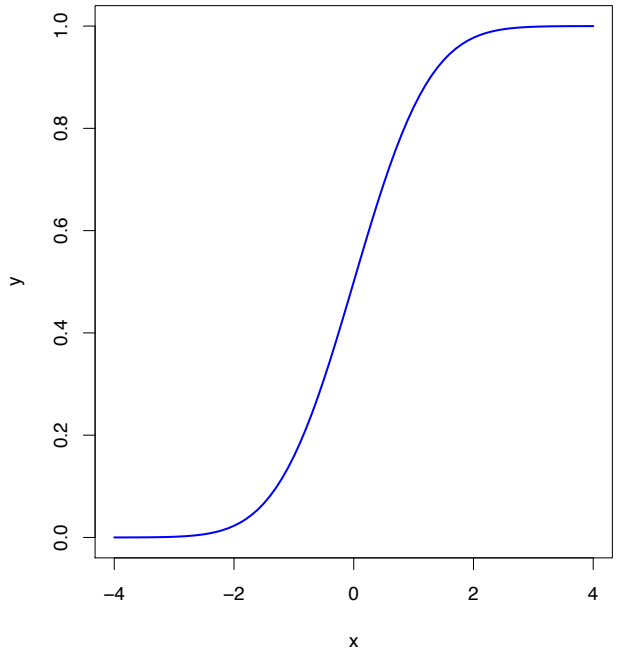
Example: normal density

```
x = seq(-4, 4 by = 0.1);
y = dnorm(x);
plot(x, y, type = 'l', col = 'blue', lwd = 2)
```



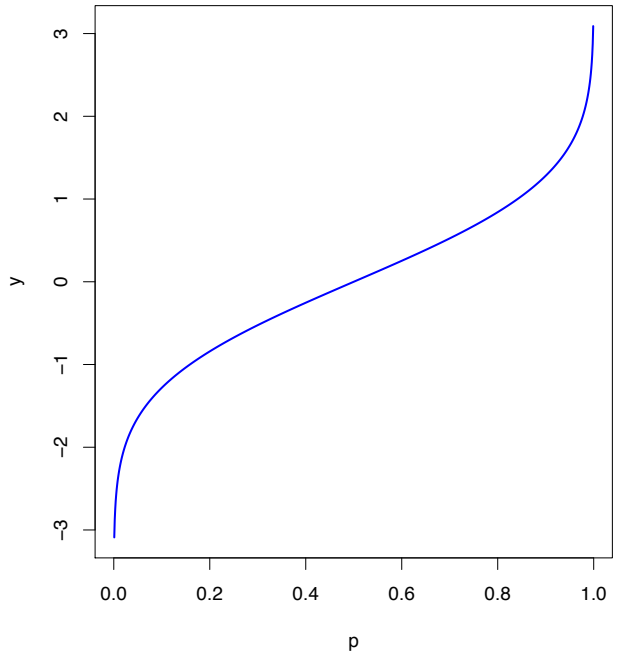
Example: cumulative normal distribution

```
x = seq(-4, 4 by = 0.1);
y = pnorm(x);
plot(x, y, type = 'l', col = 'blue', lwd = 2)
```



Example: normal quantiles

```
p = seq(0.001, 0.999, by =0.001)
y = qnorm(p)
plot(p, y, type = 'l', lwd = 2, col = 'blue')
```

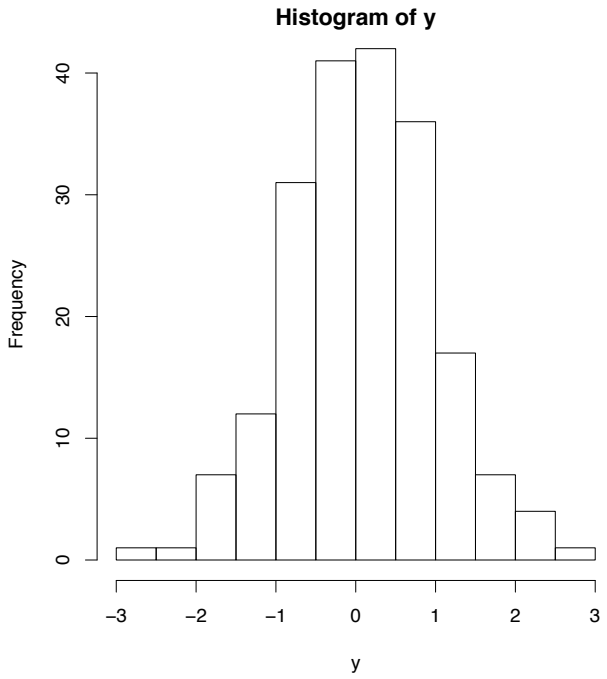


How to look at random numbers

- The result of `rnorm()` is a series of numbers
- There is little use in plotting them individually
- Instead we use the histogram
- We divide x-axis in a number of bins
- In each bin we count the number of observations
- We plot it as a bar chart

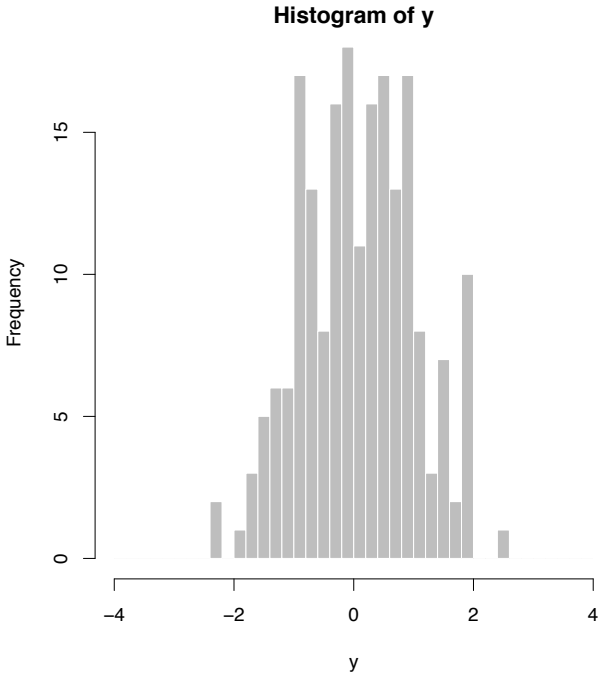
Example of a histogram

```
y = rnorm(200)
hist(y)
```



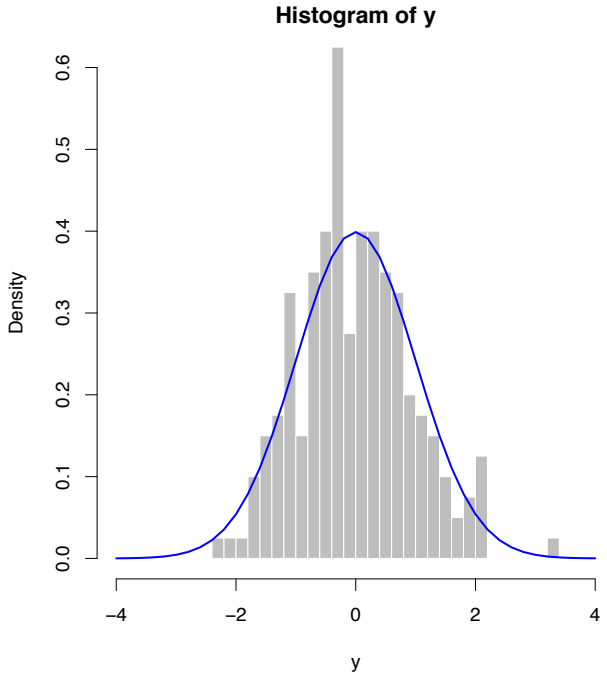
Playing with the histogram

```
y = rnorm(200)
b = seq(-4, 4, by = 0.2)  # More narrow bins
hist(y, breaks = b, col = 'gray', border = 'white')
```



Adding the theoretical normal density

```
y = rnorm(200);  b = seq(-4, 4, by = 0.2)
hist(y, breaks = b, col = 'gray', border = 'white', freq = F)
lines(b, dnorm(b), col = 'blue', lwd = 2)
```

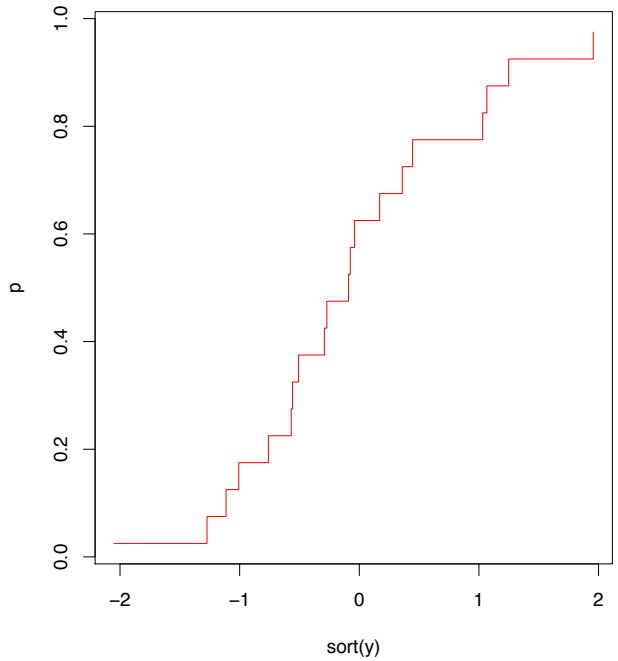


A different result every time?

- Look at the histograms carefully
- They seem to be based on different numbers
- That's indeed the case
- Every call to `rnorm()` gives a different result
- You might not like that (and you should not)
- It makes it hard to compare results
- Set the random "seed" to a starting value
- Like `set.seed(123)` or `set.seed(3245)`
- Any integer number you like (see R help for details)

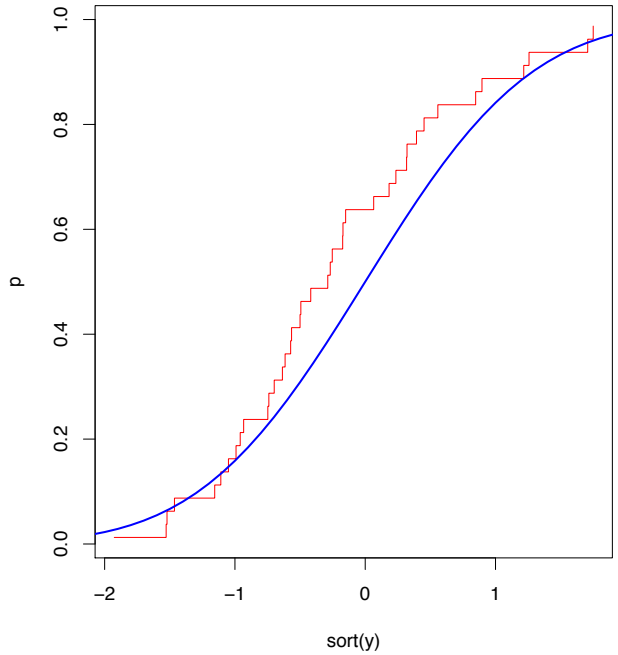
The empirical cumulative distribution

```
n = 20; y = rnorm(n)
p = ((1:n) - 0.5) / n   # Empirical probabilities
plot(sort(y), p, type = 's', col = 'red')
```



Adding the theoretical cumulative distribution

```
n = 40; y = rnorm(n); p = ((1:n) - 0.5) / n
plot(sort(y), p, type = 's', col = 'red')
x = seq(-3, 3, by = 0.1); lines(x, pnorm(x), col = 'blue', lwd = 2)
```



Randomness at work

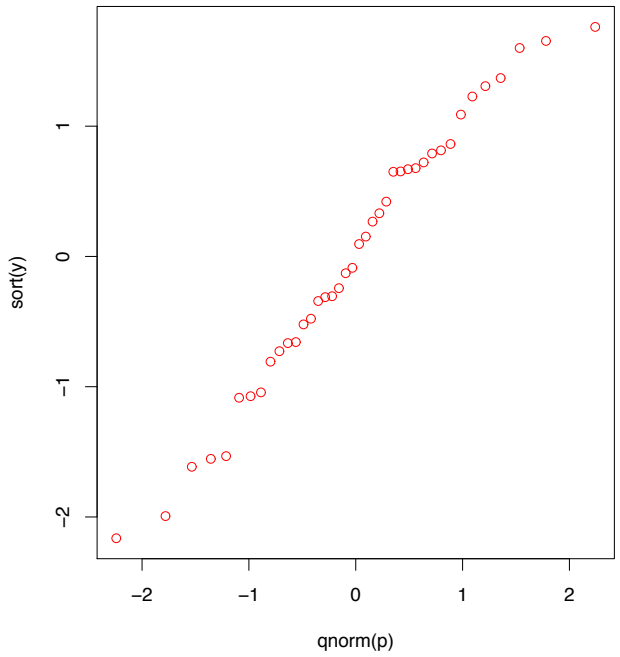
- The theoretical distribution fits not impressively
- This is generally the case for small samples
- Run these programs many times
- (Without `set.seed()`, of course)
- You will see randomness at work
- You wil appreciate variation in small samples better

The Q-Q plot

- We have our sorted observations
- And the empirical cumulative probabilities
- For the latter we can compute normal quantiles (with `qnorm()`)
- Plot one against the other
- We'll program it ourselves
- That has some educative value, I hope
- The function `qqnorm()` is easier to use
- Plot line, based on mean and variance, with `qqplot()`
- Of course, this only compares to the normal distribution

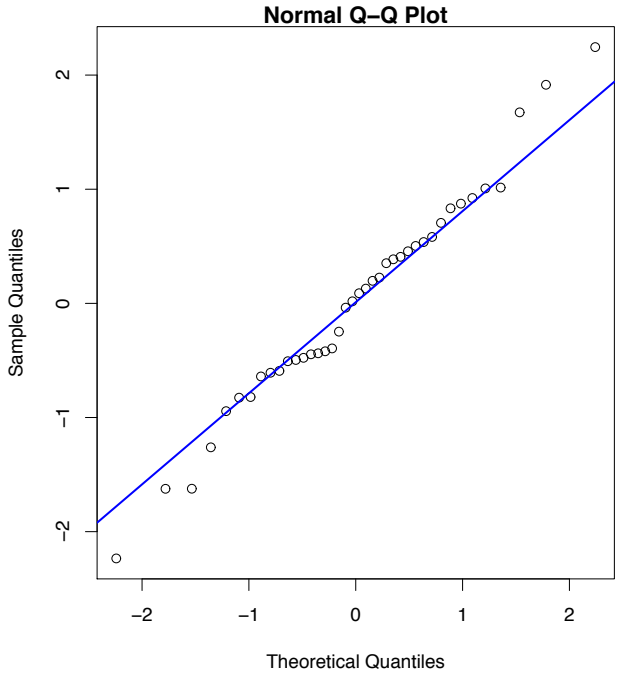
Q-Q plot from scratch

```
n = 40; y = rnorm(n);
p = ((1:n) - 0.5) / n
plot(qnorm(p), sort(y), col = 'red')
```



Q-Q plot the easy way

```
n = 40; y = rnorm(n);
qqnorm(y)
qqline(y, col = 'blue', lwd = 2)
```

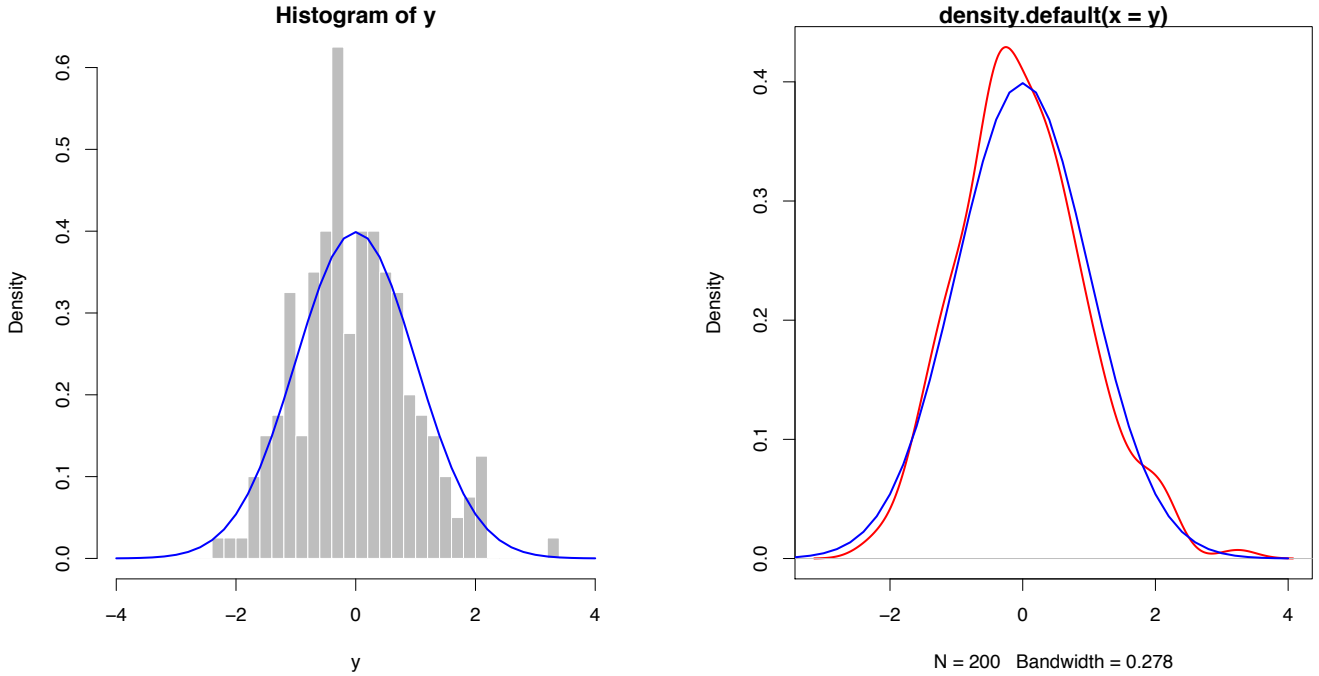


An improvement on the histogram

- The histogram is a great idea
- But it is sensitive to the width of the bins
- And to their positions (midpoints)
- An improvement is the kernel density estimator
- It is explained in the book
- We will discuss that later
- And develop a further improvement

Comparing histogram and kernels

```
y = rnorm(200); b = seq(-4, 4, by = 0.2)
plot(density(y), col = 'red', lwd = 2)
lines(b, dnorm(b), col = 'blue', lwd = 2)
```



How to generate random numbers

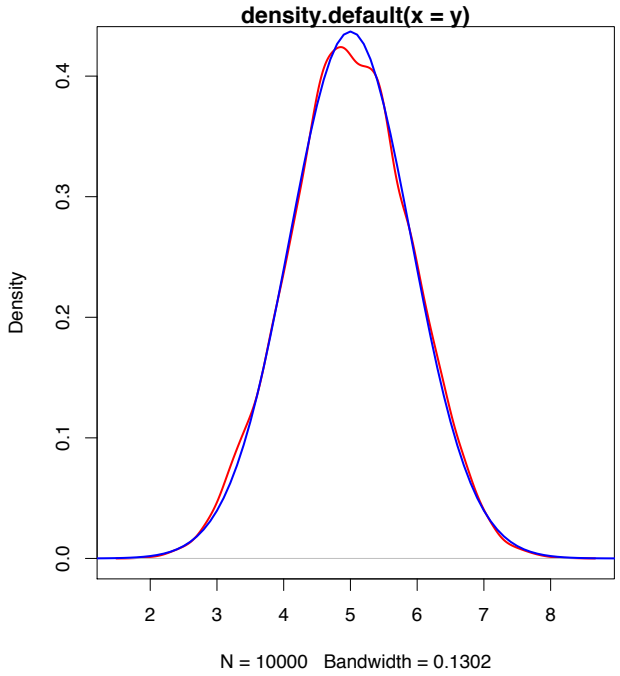
- The uniform distribution is always the starting point
- Sum 6 (or more) "uniforms" to approximate normal distribution
- This is of limited usefulness
- Apply a function: `-log(runif(100))` gives exponential distribution
- Inverse transform: compute quantiles of desired distribution (if possible)
- Acceptance-rejection sampling
- Combinations of distributions
- Mixtures

Sums of uniform random variables

- The uniform distribution runs from 0 to 1
- Mean 0.5, variance $1/12$
- Sum of n (independent) "uniforms" has mean $n/2$
- The variance is $n/12$
- Subtract $n/2$
- Divide by $\sqrt{n/12}$
- To get standard normal distribution
- This is an approximation, but a good one

Illustration of sums

```
Y = matrix(runif(10000 * 10), 10000, 10); y = apply(Y, 1, sum)
plot(density(y), col = 'red', lwd = 2)
x = seq(0, 10, by = 0.1); lines(x, dnorm(x, 10 * 0.5, sqrt(10 / 12)),
```



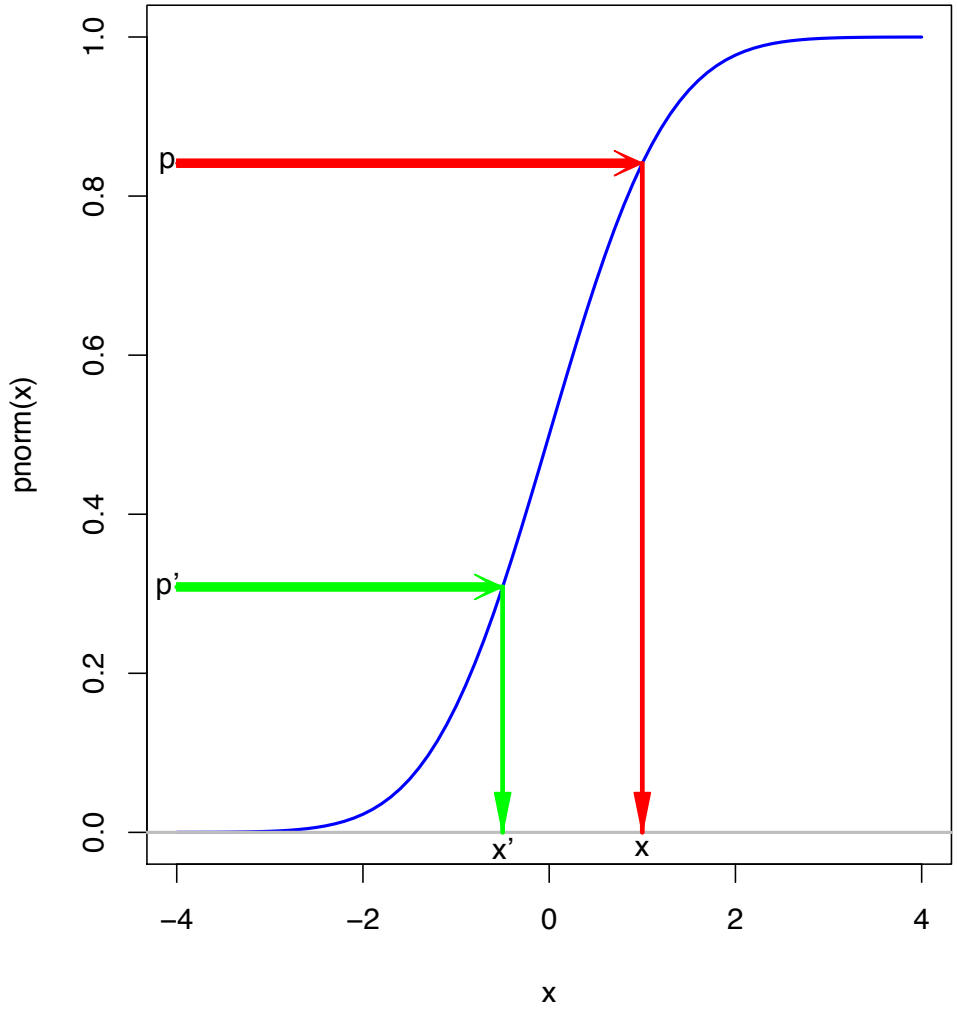
Simple transformations

- Apply a function to uniformly distributed random numbers
- Example: `y = -log(runif(n))` gives exponential distribution
- Sums of these leads to the gamma distribution
- Many of such "tricks" exist
- See examples in the book

Inverse transform or quantiles

- Remember the cumulative distribution: $p = F(x)$
- Where $F(x) = \int_{-\infty}^{x} f(t)dt$
- And $f(x)$ is the probability density
- We have that $0 < p < 1$
- Turn things around: $x = F^{-1}(p)$
- Generate uniformly distributed p
- Compute $x = F^{-1}(p)$
- Then x will have density f
- Of course, you need a procedure to invert F
- Example: `qnorm(runif(n))` is equivalent to `rnorm(n)`

Illustration of the quantile approach



A special case: the exponential distribution

- Density: $f(x) = \quad e^{-x}$ (positive x)
- Cumulative distribution: $F(x) = \quad 1 - e^{-x}$
- You can check this by integrating it yourself
- From $p = 1 - e^{-x}$ follows $x = -\log(1 - p)$
- But if p from uniform distribution, so is $1 - p$
- Hence $x = -\log(p)$ has exponential distribution

Inverting the cumulative distribution

- You need a formula for $F^{-1}(p)$
- In many cases an exact formula is not available
- But over the years very good approximations were found
- Try `?qnorm` to see pointers to literature
- Luckily, R has many built-in procedures