Tutorial week 2 - Simulations, the law of large numbers and the central limit theorem

Torbjørn Ergon

2021-01-20

Contents

Key terms and concepts covered in this tutorial

- Simulations in R
- for-loops
- Law of large numbers
- Central limit theorem
- Sampling distribution
- Numerical convergence

Summary

We will here show how you can draw random values from a probability distribution and how you can repeat calculations by the use of loops in R. We will use this to investigate the important central limit theorem as well as the law of large numbers.

Preparations

We assume that you are familiar with histograms and probability distributions. You may want to refresh these concepts from earlier course work or look at the resources we refer to on Canvas.

We recommend that you download the R Markdown file (*.Rmd) for each tutorial to a local folders on your computer. Later tutorials may also include datasets that you need to download. Name the folders "Week01", "Week02", and so on. Using "...01" instead of just "...1" ensures that the folder gets sorted before "Week10". Change the name of the file if you modify the document!

Drawing random values in R

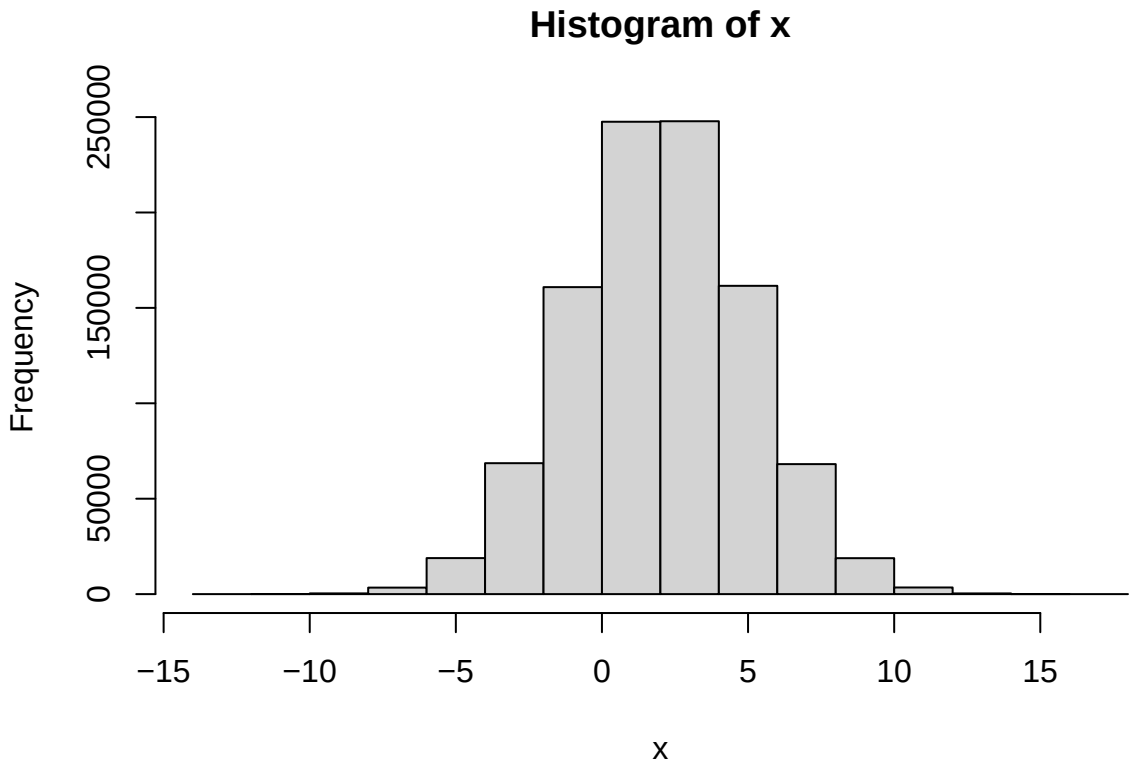
In R we can draw random values from a normal distribution with the `rnorm`-function. For example, the following draws 10 values from a normal distribution with mean 2 and standard deviation 3:

```
rnorm(n = 10, mean = 2, sd = 3)
```

```
##  [1]  3.2709540 -0.4005255  1.2162939 -3.7280884  0.3199824  1.0349364
##  [7]  2.3131600 -0.5520295 -1.1149140  6.0640114
```

We can easily draw a sample of a million values from this distribution and plot a histogram of these values. Below, we first assign these random values to a vector `x` and plot a histogram of the values:

```
x = rnorm(n = 1000000, mean = 2, sd = 3)
hist(x)
```



We can also compute the mean and standard deviation of these random values:

```
mean(x)
```

```
## [1] 2.000358
```

```
sd(x)
```

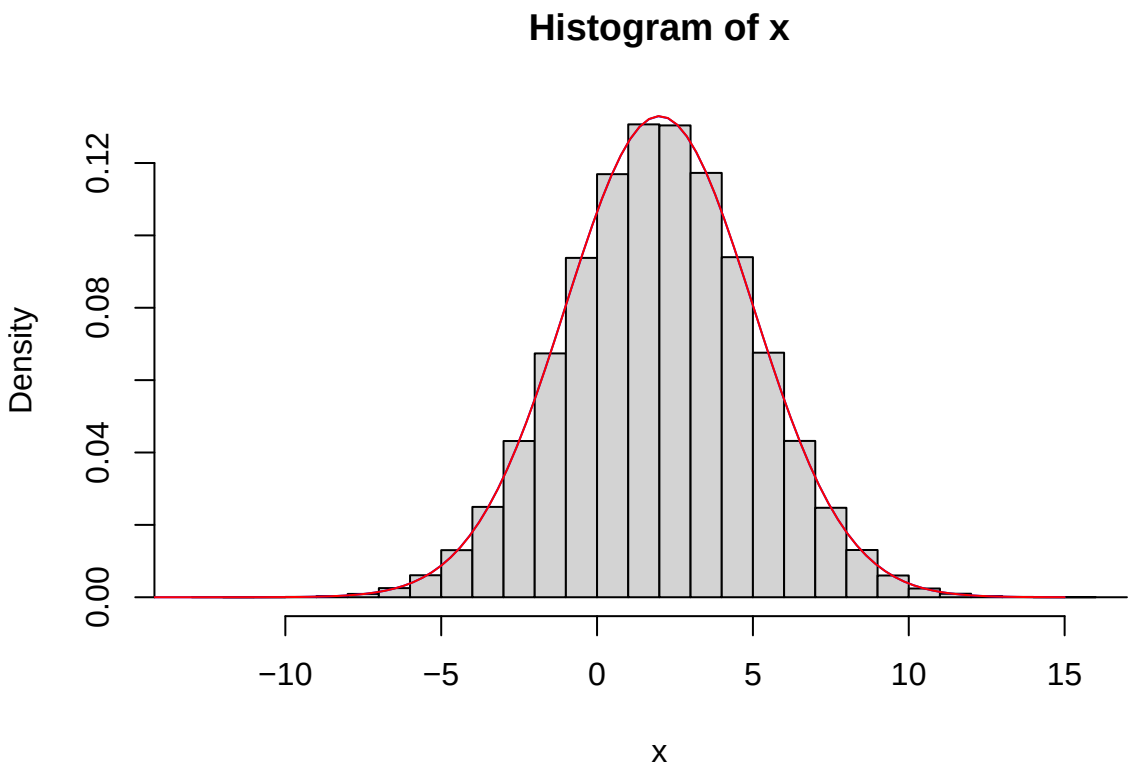
```
## [1] 2.998263
```

Since the sample size is so large (one million), the mean and standard deviation of the sample is very close to the mean and standard deviation of the distribution we drew them from (2 and 3 respectively) - this is due to the "Law of large numbers" which states the properties of the sample will be close to the properties of the population it is drawn from when sample size is large.

Note, that if you rerun the above code, you will get slightly different numbers because you will draw different values. If you draw fewer numbers (say 10), then the mean and standard deviation of the sample may deviate more from the mean and standard deviation of the distribution you draw from. Go ahead - try this out in the Rmd file for this document: First find the R chunk that calculates the mean and standard deviation above and click on the downward pointing triangle on the right side to run all R code prior to this chunk. Then click on the green triangle pointing to the right to run the code in the current chunk. Try this by modifying the code! Try it several times!

Also the *distribution* of values in the sample will approach the distribution of the population it is drawn from (a theoretical normal distribution in this case) when sample size is large (as stated by the "Law of large numbers"). We can show this by first using the `freq = FALSE` argument in the `hist` function to make the area of the histogram equal to 1 (as in a distribution), and then overlaying the distribution of the population:

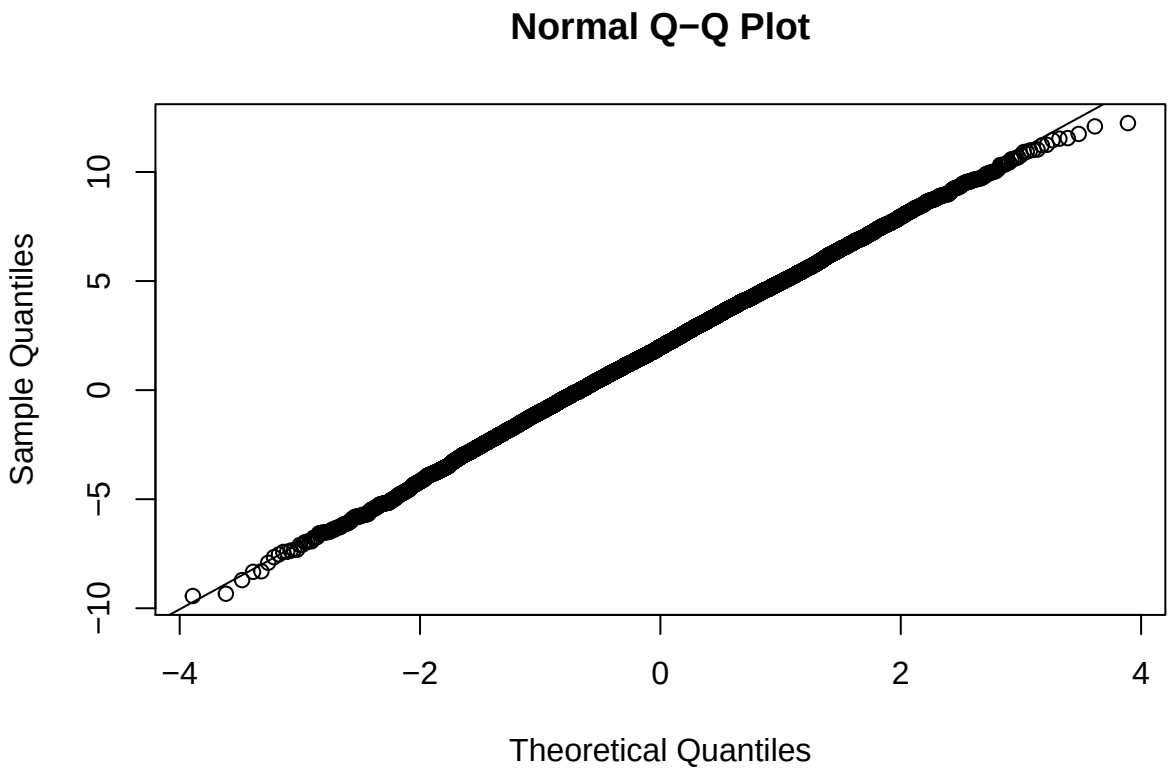
```
x = rnorm(n = 1000000, mean = 2, sd = 3)
hist(x, freq = FALSE) # with freq = FALSE, the area of the histogram will be 1
xx = seq(-15, 15, length.out = 100) # 100 values between -15 and 15
lines(xx, dnorm(xx, mean = mean(x), sd = sd(x)), col="blue") # using mean and sd from sample
lines(xx, dnorm(xx, mean = 2, sd = 3), col="red") # using mean and sd of population
```



In the above plot there is a red line for the normal distribution with the mean and standard deviation of the *population*. We have also plotted a blue line for the normal distribution with the mean and standard deviation of the *sample*. However, with sample size of one million, the red line completely covers the blue line. Try running the above chunk several times with smaller sample size!

Another way of assessing whether the values in a sample seem to follow a particular theoretical distribution is a so called "quantile-quantile plot" or a "qq-plot" for short. When the theoretical distribution is a normal distribution, we can use the function `qqnorm` to produce such a plot (it takes a bit of time to do this with a sample of a million, so a sample of 10,000 has been used below instead)

```
x = rnorm(n = 10000, mean = 2, sd = 3)
qqnorm(x)
qqline(x) # this just adds the line in the plot
```



If the distribution of the sample resembles the theoretical distribution, then all points in the plot should be close to the line (there will usually be some deviations due to chance for the extreme low and high values as in the plot above). Note that we have not specified the mean and standard deviation of the theoretical normal distribution, so the plot assesses whether the sample seems to be normally distributed or not in general (without specifying the mean and standard deviation).

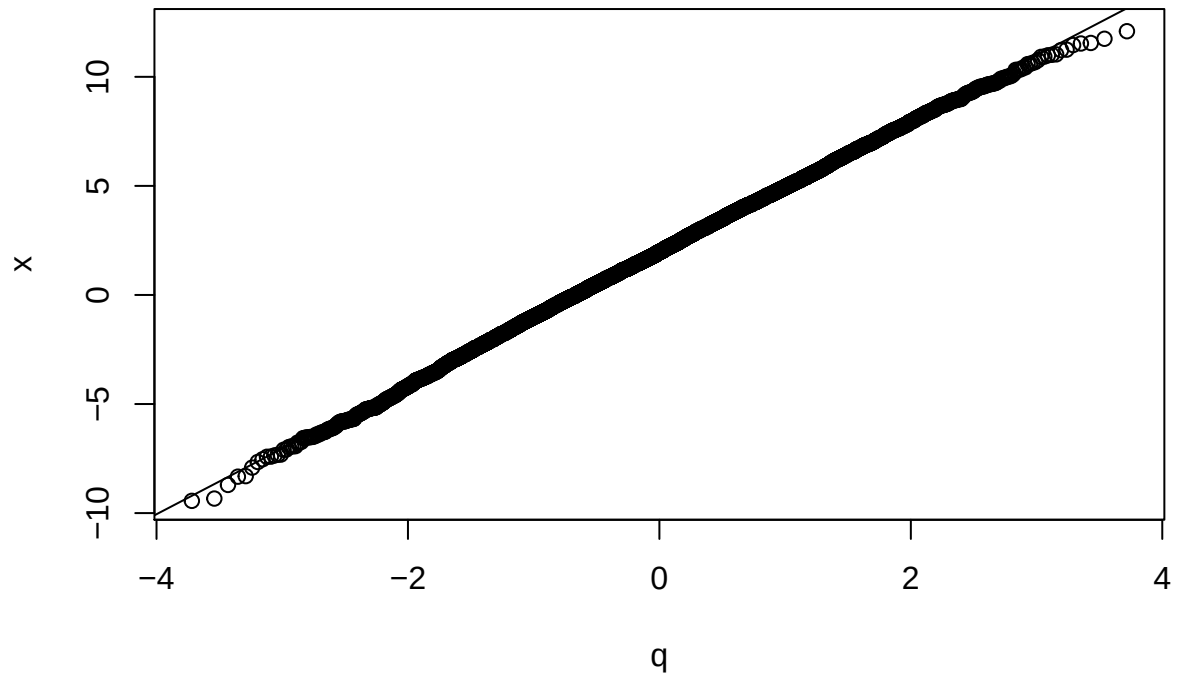
FOR A DEEPER UNDERSTANDING:

The qq-plot above is made in the following way (studying this helps you understand what a qq-plot is and how it can be interpreted):

1. For each value in the sample (vector `x` here) one first finds what proportion of values that are smaller or equal to the given value. Lets call these proportions for `p` (`p` range from `1/n` to `1` where `n` is the number of values).
2. Then one finds the value `q` in the standard normal distribution for which a proportion `p` of the distribution is smaller than the value (function `qnorm`).
3. Finally `x` is plotted against `q`.

It is easiest to do this if we first sort the vector `x` from smallest to largest, which is what we will do below:

```
x = sort(x)    # sort the values from smallest to largest
n = length(x) # number of values
p = (1:n)/n    # step 1 above
q = qnorm(p)  # step 2 above
plot(x~q)      # step 3 above
abline(mean(x), sd(x)) # Plot a line with intercept = mean(x) and slope = sd(x)
```



For further help to interpret qq-plots, see this external resource.

To understand what the code chunk above (or any code you see) does, it is a good idea to try it out with a small example. Set `x2 = rnorm(n = 10000, mean = 2, sd = 3)`, replace `x` with `x2` (you don't want to overwrite `x`) and then run line by line in the Console window and look at the result.

Loops and simulations

Loops are useful for repeating a set of calculations. As an example, the code chunk below repeats the following 10,000 times:

1. Draw 10 random numbers from a normal distribution and store this in a vector `x`.
2. Compute the mean of `x` and store this in a vector `means`.

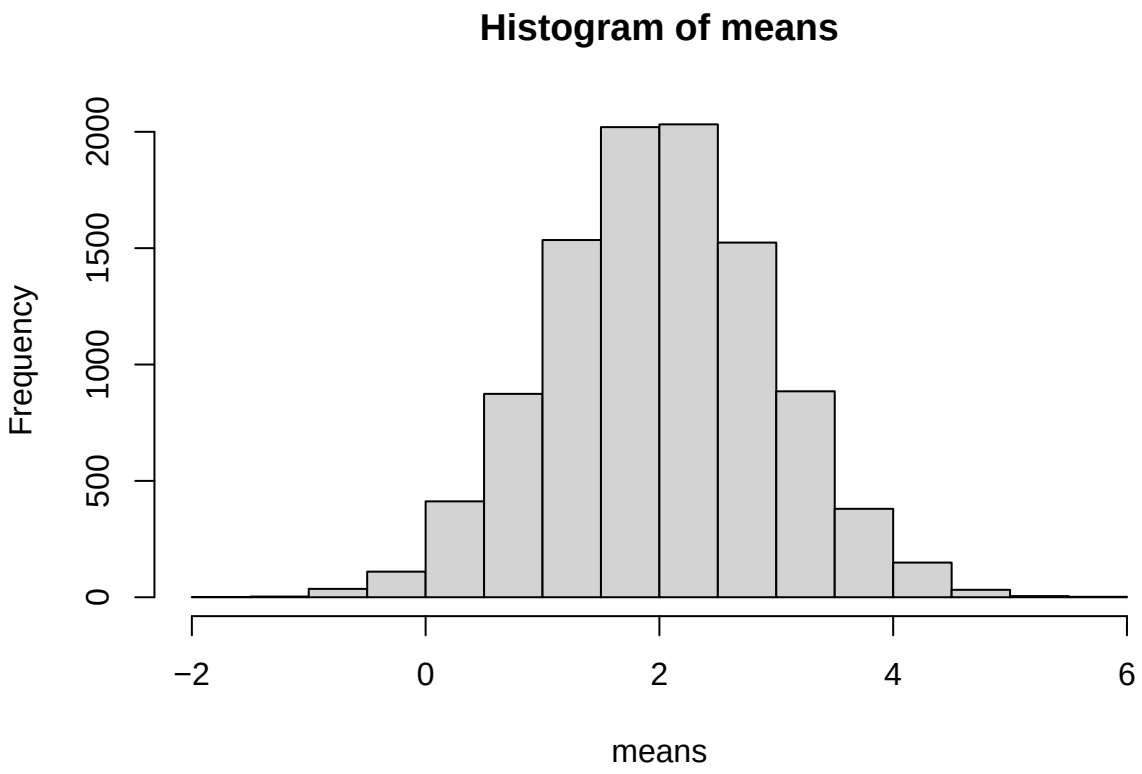
```
n_iter = 10000             # Number of iterations
mu_x = 2                   # Mean of the normal distribution (same as used above)
sd_x = 3                   # Standard deviation of the normal distribution (same as used above)
n = 10                     # Sample size
means = rep(NA, n_iter)    # An empty vector to store the means in
for(i in 1:n_iter){         # Start of the loop
  x = rnorm(n, mu_x, sd_x)  # Using the same mean and sd as above
  means[i] = mean(x)       # Compute mean(x) and store it as the i'th element in vector 'means'
}                           # End of loop
```

Here `i` is the iteration counter. The first time R runs through the loop, `i` equals 1, the second time `i` equals 2, and so on until `i` equals `n_iter`. `x`is a vector of length 10 that is overwritten with new random values in every iteration of the loop, and `means` is a vector of length 10^4 that stores the results (mean of the 10 random values for each iteration).

If you are new to loops, you may want to pause here and make a simpler loop to make sure you understand how it works (you can for example compute `i^2` (`i` to the power of 2) and `2^i` (2 to the power of `i`) in a loop where you let `i` go from 1 to 10; you should get the same result as running `(1:10)^2` and `2^(1:10)`). You may also watch this YouTube video from the Google Developers.

Let us look at the distribution of the means computed above:

```
hist(means)
```



What we have done in the loop above is to simulate the *sampling distribution* of mean(x) when there are 10 values of x drawn from a normal distribution with mean 2 and standard deviation 3. Note that increasing the number of iterations (this is the sample size of mean(x) while the sample size of x is 10) will get us closer and closer to the sampling distribution of mean(x) due to the "Law of large numbers" which we investigated in the previous section. Since 10,000 is already a large number, increasing the number of iterations (sample size of mean(x)) will not change the above histogram much (you can try it!). When increasing number of iterations further will not change the results of the simulations in a noticeable way, we say that the simulation has *converged*.

Perhaps not surprisingly, this distribution looks quite "bell-shaped" like a normal distribution. The mean seems to be around 2 which is what we used as the mean of the normal distribution for which all the 10,000 samples of 10 were drawn from. However, if you compare the width of the histogram to the histogram we plotted earlier (note that we used the same standard deviation of 3 also here), we see that the histogram above is much narrower (look at the values on the x-axis). You may remember from your earlier statistics course that the standard deviation of the mean of n values drawn independently from the same distribution is $\text{sd(x)}/\sqrt{n}$ where sd(x) is the standard deviation of the distribution that the values are drawn from ($\text{sd(x)}/\sqrt{n}$ is often called the "standard error of the mean"). Let us check if this is correct:

```
(sem_theoretical = sd_x/sqrt(n))
```

```
## [1] 0.9486833
```

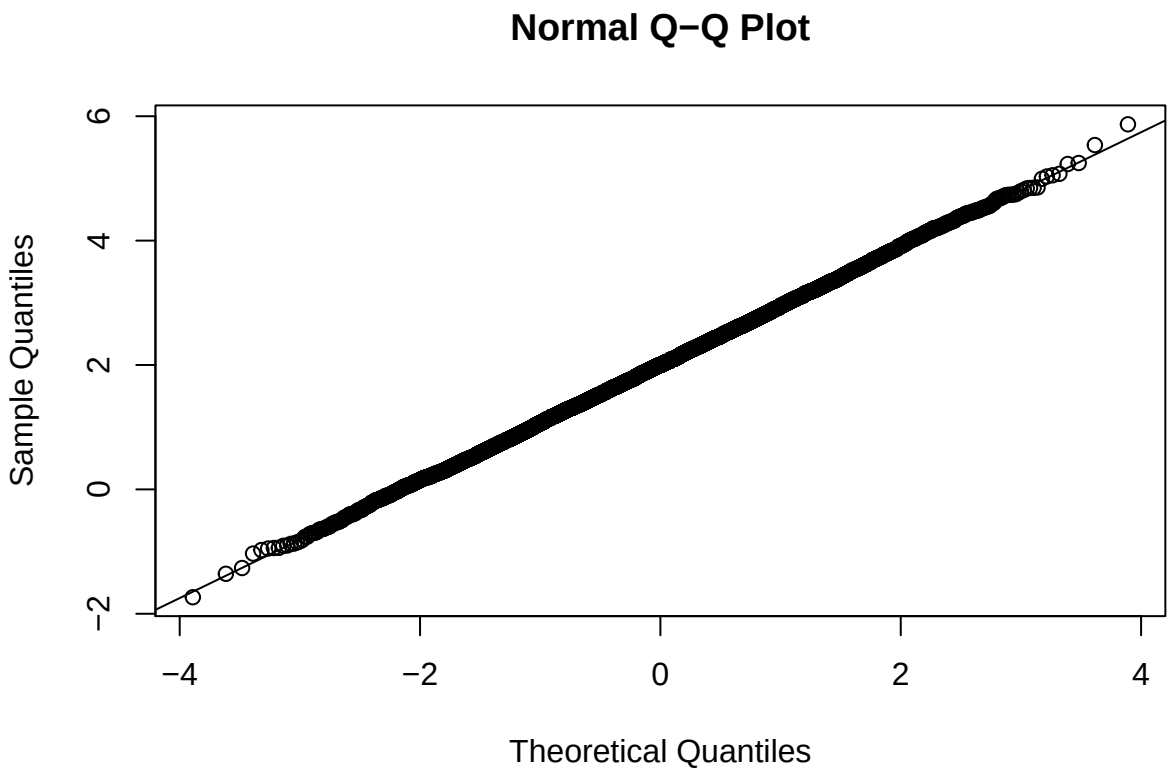
```
(sem_simulated = sd(means)) # Standard error of the mean from the above simulation
```

```
## [1] 0.9421239
```

The parentheses around the assignments above is a trick to get R to print the value at the same time as the value is assigned to a new variable. The difference between the simulated standard error of the mean, 0.9421239, and theoretical value, 0.9486833, is about -0.69% of the theoretical value when number of iterations (samples of mean(x)) is 10^4. This difference will be even smaller if you increase the number of iterations (you can try - the values in this paragraph change if you do).

Finally, we can confirm that the distribution of the mean is indeed close to a normal distribution:

```
qqnorm(means)
qqline(means)
```



The central limit theorem

In the previous section we saw that the mean of a sample of values drawn from a normal distribution is itself normally distributed (we showed this by simulation). The important central limit theorem states that this will be the case no matter what the distribution the values are drawn from as long as the sample size is large enough. For example, you can try to replace the normal distribution in the above simulation with a uniform distribution by replacing `rnorm(n, mux, sdx)` with `runif(n, min, max)`. In a uniform distribution, all values between `min` and `max` are equally likely. With a sample size of only `n = 10` you will see a slight deviation from a normal distribution, but as you increase `n` this deviation will be smaller.

Exercises

1. Draw one 10 random values from a normal distribution with mean 12 and standard deviation 2.

a) What do you expect the variance of the random values to be?
b) Compute the empirical variance of the random values. Is the variance close to what you expected? If not, why not? Repeat this with 10 new random values.
c) Compute the variance of one million values drawn from the same distribution. Is this closer to the expected value? Why?

2. Compute the values `2^i` when `i` goes from 1 to 10 in a for-loop (you should get the same results as when running `2^(1:10)` in R).

3. Do the same as above, but with `i` going from 0 to 10.

4. Use R to simulate the probability distribution of the sum of two independent dice throws and plot a histogram. **HINTS:**

   - You can simulate one dice throw with `sample(1:6, 1)` and two independent dice throws with `sample(1:6, 2, replace = T)` (Why do we need to use `replace = T`? Remember that you can look up the description of any R-function in the 'help' (e.g. type `?sample`))
   - The `hist()` function (using default arguments) does not produce a nice histogram with integer values. Instead, you can use `table` to count how any vector elements you have of each value (try e.g. `table(c(1,1,2,3))`), and then make a bar-plot of these counts (table values) using `barplot()`.

Assignment

The assignment this week is to produce a document in R Markdown that demonstrates the central limit theorem.

Pick a hypothetical example that you describe in words. You can for example look at the distribution of the mean height of trees in a forest or the mean number of parasites per individual in a sample of individuals (specify the distribution of tree heights or number of parasites in an individual and try the simulation for different sample sizes). Some distributions that you can consider using are uniform (`runif()`), binomial (`rbinom()`), exponential (`rexp()`) or Poisson (`rpois()`) (read about the distributions in Wikipedia (English version) and see how you specify the parameters in the R help). You may also use the `sample()` function to draw from a set of values (with different probabilities). Do not draw values from a normal distribution (because the central limit theorem says that the mean of a sample of values drawn from *any* distribution (not just the normal distribution) will be normally distributed).

Demonstrate the central limit theorem by using a series of simulations with increasing sample size (try for example a sample size of 2, 5, 10, 20, ...). Make sure you use high enough number of iterations for the distribution to converge (10 000 iterations will usually be sufficient). Write some explanations and interpretations in words - don't just include the simulations.

You may use the Rmd-file for this document as a template, but make sure you remove all content that is not your own in the end. Submit a pdf-file, together with the Rmd-file on Canvas. You can create the pdf-file either by saving (or printing) the html-file to a pdf-file from your browser, or you can knit directly to a pdf-file if you have installed LaTeX (see instructions in the Week 1 module on Canvas). We need the pdf-file to provide comments in your text.