Using R for Introduction to Econometrics

Christoph Hanck, Martin Arnold, Alexander Gerber and Martin Schmelzer

2018-07-16

Contents

Chapter 1

Introduction

The interest in the freely available statistical programming language and software environment R is soaring. By the time we wrote first drafts for this project, more than 11000 addons (many of them providing cutting-edge methods) were made available on the Comprehensive R Archive Network (CRAN), an extensive network of FTP servers around the world that store identical and up-to-date versions of R code and its documentation. R dominates other (commercial) software for statistical computing in most fields of research in applied statistics. The benefits of it being freely available, open source and having a large and constantly growing community of users that contribute to CRAN render R more and more appealing for empirical economists and econometricians as well.

A striking advantage of using R in econometrics courses is that it enables students to explicitly document their analysis step-by-step such that it is easy to update and to expand. This allows to re-use code for similar applications with different data. Furthermore, R programs are fully reproducible which makes it straightforward for others to comprehend and validate results.

Over the recent years, R has thus become an integral part of the curricula of econometrics classes we teach at University of Duisburg-Essen. In some sense, learning to code is comparable to learning a foreign language and continuous practice is essential for the learning success. Of course, presenting bare R code chunks on slides has mostly a deterring effect for the students to engage with programming on their own. This is why we offer tutorials where both econometric theory and its applications using R are introduced, for some time now. As for accompanying literature, there are some excellent books that deal with R and its applications to econometrics like Kleiber and Zeileis (2008). However, we have found that these works are somewhat difficult to access, especially for undergraduate students in economics having little understanding of econometric methods and predominantly no experience in programming at all. Consequently, we have started to compile a collection of reproducible reports for use in class. These reports provide guidance on how to implement selected applications from the textbook *Introduction to Econometrics* (Stock and Watson, 2015) which serves as a basis for the lecture and the accompanying tutorials. The process has been facilitated considerably with the release of `knitr` (2018). `knitr` is an R package for dynamic report generation which allows to seamlessly combine pure text, LaTeX, R code and its output in a variety of formats, including PDF and HTML. Being inspired by *Using R for Introductory Econometrics* (Heiss, 2016)[1] and with this powerful toolkit at hand we decided to write up our own empirical companion to Stock and Watson (2015) which resulted in **U**sing **R** **f**or **I**ntroduction **t**o **E**conometrics (*URFITE*).

Similarly to the book by Heiss (2016) this project is neither a comprehensive econometrics textbook nor is it intended to be a general introduction R. *URFITE* is best described as an interactive script in the style of a reproducible research report which aims to provide students of economic sciences with a platform-independent e-learning arrangement by seamlessly intertwining theoretical core knowledge and empirical skills in undergraduate econometrics. Of course the focus is set on empirical applications with R; we leave out tedious derivations and formal proofs wherever we can. *URFITE* is closely aligned on Stock and Watson (2015) which does very well in motivating theory by real-world applications. However, we take it a step further and enable students not only to learn how results of case studies can be replicated with R but we also intend to strengthen their ability in using the newly acquired skills in other empirical applications. To support this, each chapter contains interactive R programming exercises. These exercises are used as supplements to code chunks that display how previously discussed techniques can be implemented within R. They are generated using the DataCamp light widget and are backed by an R-session which is maintained on DataCamp's servers. You may play around with the example exercise presented below.

This interactive application is only available in the HTML version.

As you can see above, the widget consists of two tabs. `script.R` mimics an `.R`-file, a file format that is commonly used for storing R code. Lines starting with a # are commented out, that is they are not recognized as code. Furthermore, `script.R` works like an exercise sheet where you may write down the solution you come up with. If you hit the button *Run*, the code will be executed, submission correctness tests are run and you will be notified whether your approach is correct. If it is not correct, you will receive feedback suggesting improvements or hints. The other tab, `R Console`, is a fully functional R console that can be used for trying out solutions to exercises before submitting them. Of course you may submit (almost any) arbitrary R code and use the console to play around and explore. Simply type a command and hit the enter key on your keyboard.

As an example, consider the following line of code presented in chunk below. It tells R to compute the number of packages available on `CRAN`. The code chunk is followed by the output produced.

```r
# compute the number of packages available on CRAN
nrow(available.packages(repos = "http://cran.us.r-project.org"))
```

```
## [1] 12721
```

Each code chunk is equipped with a button on the outer right hand side which copies the code to your clipboard. This makes it convenient to work with larger code segments. In the widget above, you may click on `R Console` and type `nrow(available.packages())` (the command from the code chunk above) and execute it by hitting *Enter* on your keyboard[2].

[1] Heiss (2016) builds on the popular *Introductory Econometrics* by Wooldridge (2016) and demonstrates how to replicate the applications discussed therein using R.

[2] The R session is initialized by clicking anywhere into the widget. This might take a few seconds. Just wait for the indicator next to the button *Run* to turn green

As you might have noticed, there are some out-commented lines in the widget that ask you to assign a numeric value to a variable and then to print the variable's content to the console. You may enter your solution approach to `script.R` and hit the button *Run* in order to get the feedback described further above. In case you do not know how to solve this sample exercise (don't panic, that is probably why you are reading this), a click on *Hint* will prompt you with some advice. If you still can't find a solution, a click on *solution* will provide you with another tab, `Solution.R` which contains sample solution code. It will often be the case that exercises can be solved in many different ways and `Solution.R` presents what we consider as comprehensible and idiomatic.

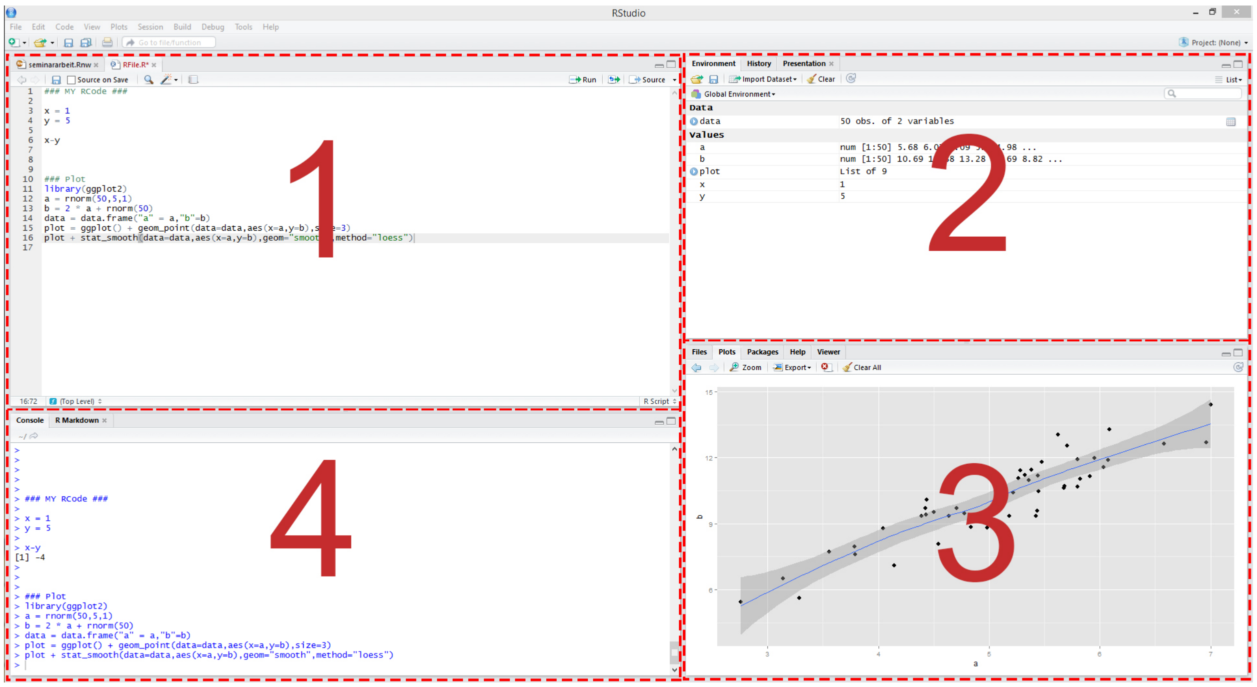
Conventions Used in this Book

- *Italic* text indicates new terms, names, buttons and alike.
- `Constant width text`, is generally used in paragraphs to refer to `R` code. This includes commands, variables, functions, data types, databases and file names.
- Constant width text on gray background is used to indicate `R` code that can be typed literally by you. It may appear in paragraphs for better distinguishability among executable and non-executable code statements but it will mostly be encountered in shape of large blocks of `R` code. These blocks are referred to as code chunks (see above).

Acknowledgements

We thank Alexander Blasberg and Kim Hermann for proofreading and their constructive criticism.

1.1 A Very Short Introduction to `R` and *RStudio*



```
### MY RCode ###

x = 1
y = 5

x-y



### Plot
library(ggplot2)
a = rnorm(50,5,1)
b = 2 * a + rnorm(50)
data = data.frame("a" = a,"b"=b)
plot = ggplot() + geom_point(data=data,aes(x=a,y=b),size=3)
plot + stat_smooth(data=data,aes(x=a,y=b),geom="smooth",method="loess")
```

```
Data
data        50 obs. of 2 variables
Values
a           num [1:50] 5.68 6.0 .09 .98 ...
b           num [1:50] 10.69 1 8 13.28 69 8.82 ...
plot        List of 9
x           1
y           5
```

```
>
>
>
>
>
> ### MY RCode ###
>
> x = 1
> y = 5
>
> x-y
[1] -4
>
>
>
> ### Plot
> library(ggplot2)
> a = rnorm(50,5,1)
> b = 2 * a + rnorm(50)
> data = data.frame("a" = a,"b"=b)
> plot = ggplot() + geom_point(data=data,aes(x=a,y=b),size=3)
> plot + stat_smooth(data=data,aes(x=a,y=b),geom="smooth",method="loess")
> 
```

Figure 1.1: *RStudio*: the four panes

R Basics

This section is meant for those who have never worked with R or *RStudio*. If you at least know how to create objects and call functions, you can skip it. If you would like to refresh your memories or get a feeling for how to work with *RStudio*, keep reading.

First of all start *RStudio* and create a new R Script by selecting *File*, *New File*, *R Script*. In the editor pane type

```r
1 + 1
```

and click on the button labeled *Run* in the top right corner of the editor. By doing so, your line of code is send to the console and the result of this operation should be displayed right underneath it. As you can see, R works just like a calculator. You can do all the arithmetic calculations by using the corresponding operator (+, - , *, / or ^). If you are not sure what the last operator does, try it out and check the results.

Vectors

R is of course more sophisticated than that. We can work with variables or more generally objects. Objects are defined by using the assignment operator `<-`. To create a variable named `x` which contains the value `10` type `x <- 10` and click the button *Run* yet again. The new variable should have appeared in the environment pane on the top right. The console however did not show any results, because our line of code did not contain any call that creates output. When you now type `x` in the console and hit return, you ask R to show you the value of `x` and the corresponding value should be printed in the console.

`x` is a scalar, a vector of length 1. You can easily create longer vectors by using the function `c()` (*c* for "concatenate" or "combine"). To create a vector `y` containing the numbers 1 to 5 and print it, do the following.

```r
y <- c(1, 2, 3, 4, 5)
y
```

```
## [1] 1 2 3 4 5
```

You can also create a vector of letters or words. For now just remember that characters have to be surrounded by quotes, else wise they will be parsed as object names.

```r
hello <- c("Hello", "World")
```

Here we have created a vector of length 2 containing the words `Hello` and `World`.

Do not forget to save your script! To do so, select *File*, *Save*.

Functions

You have seen the function `c()` that can be used to combine objects. In general, function calls look all the same, a function name is always followed by round parentheses. Sometimes, the parentheses include arguments

Here are two simple examples.

```r
z <- seq(from = 1, to = 5, by = 1)

mean(x = z)
```

```
## [1] 3
```

In the first line we use a function called `seq` to create the exact same vector as we did in the previous section but naming it `z`. The function takes on the arguments `from`, `to` and `by` which should be self-explaining.

The function `mean()` computes the arithmetic mean of its argument `x`. Since we pass the vector `z` as the argument `x` to `mean()`, the result is `3`!

If you are not sure what argument a function expects you may consult the function's documentation. Let's say we are not sure how the arguments required for `seq()` work. Then we can type `?seq` in the console and by hitting return the documentation page for that function pops up in the lower right pane of *RStudio*. In there, the section *Arguments* holds the information we seek.

On the bottom of almost every help page you find examples on how to use the corresponding functions. This is very helpful for beginners and we recommend to look out for those.

Chapter 2

Probability Theory

This chapter reviews some basic concepts of probability theory and demonstrates how they can be applied in `R`.

Most of the statistical functionalities in `R`'s standard version are collected in the `stats` package. It provides simple functions which compute descriptive measures and facilitate calculus involving a variety of probability distributions. However, it also holds more sophisticated routines that e.g. enable the user to estimate a large number of models based on the same data or help to conduct extensive simulation studies. `stats` is part of the base distribution of `R`, meaning that it is installed by default so there is no need to run `install.packages("stats")` or `library("stats")`. Simply execute `library(help = "stats")` in the console to view the documentation and a complete list of all functions gathered in `stats`. For most packages a documentation that can be viewed within *RStudio* is available. Documentations can be invoked using the `?` operator, e.g. upon execution of `?stats` the documentation of the `stats` package is shown in the help tab of the bottom-right pane.

In what follows, we lay our focus on (some of) the probability distributions that are handled by `R` and show how to use the relevant functions to solve simple problems. Thereby we will repeat some core concepts of probability theory. Among other things, you will learn how to draw random numbers, how to compute densities, probabilities, quantiles and alike. As we shall see, it is very convenient to rely on these routines, especially when writing custom functions.

2.1 Random Variables and Probability Distributions

For a start, let us briefly review some basic concepts of probability theory.

- The mutually exclusive results of a random process are called the *outcomes*. 'Mutually exclusive' means that only one of the possible outcomes is observed.
- We refer to the *probability* of an outcome as the proportion of the time that the outcome occurs in the long run, that is if the experiment is repeated very often.
- The set of all possible outcomes of a random variable is called the *sample space*.
- An *event* is a subset of the sample space and consists of one or more outcomes.

These ideas are unified in the concept of a *random variable* which is a numerical summary of random outcomes. Random variables can be *discrete* or *continuous*.

- Discrete random variables have discrete outcomes, e.g. 0 and 1.
- A continuous random variable takes on a continuum of possible values.

Probability Distributions of Discrete Random Variables

A typical example for a discrete random variable D is the result of a die roll: in terms of a random experiment this is nothing but randomly selecting a sample of size 1 from a set of numbers which are mutually exclusive outcomes. Here, the sample space is $\{1, 2, 3, 4, 5, 6\}$ and we can think of many different events, e.g. 'the observed outcome lies between 2 and 5'.

A basic function to draw random samples from a specified set of elements is the the function `sample()`, see `?sample`. We can use it to simulate the random outcome of a die roll. Let's role the die!

```
sample(1:6, 1)
```

```
## [1] 3
```

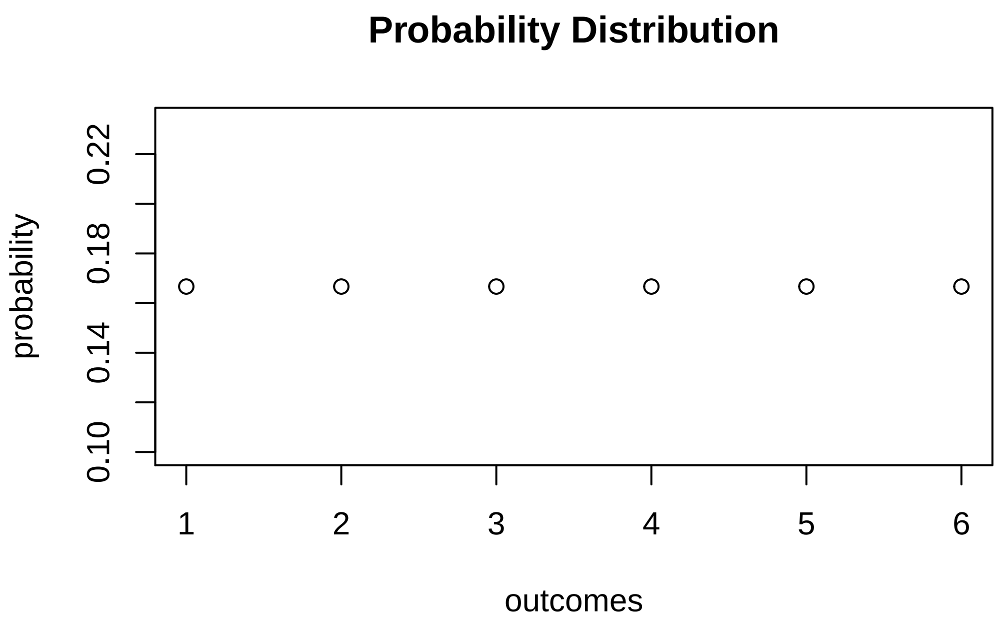
The probability distribution of a discrete random variable is the list of all possible values of the variable and their probabilities which sum to 1. The cumulative probability distribution function states the probability that the random variable is less than or equal to a particular value.

For the die roll, this is straightforward to summarize in a table:

Outcome	1	2	3	4	5	6
Probability distribution	1/6	1/6	1/6	1/6	1/6	1/6
Cumulative probability distribution	1/6	2/6	3/6	4/6	5/6	1

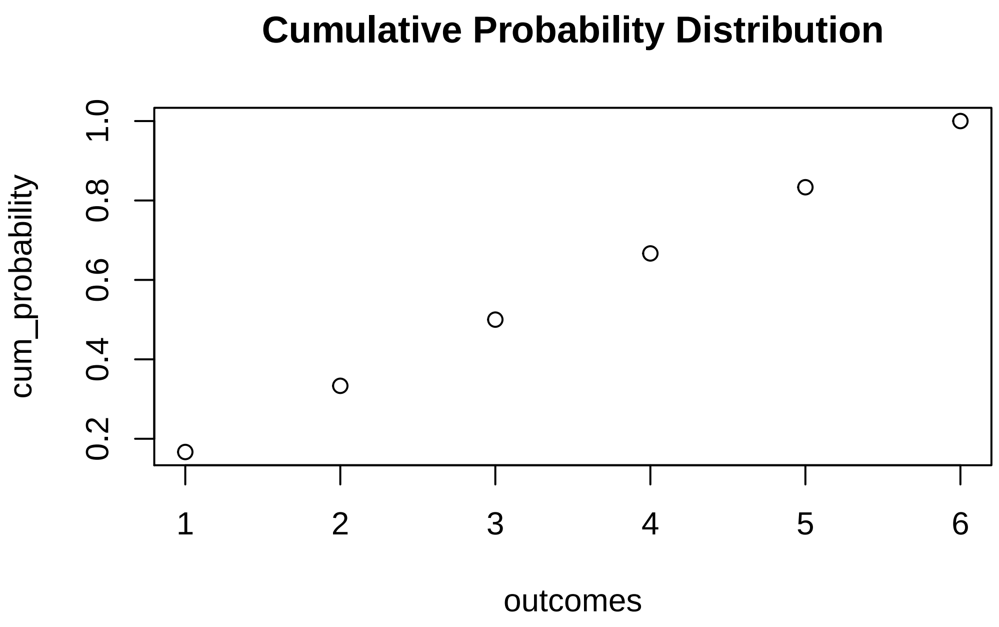
We can easily plot both functions using R. Since the probability equals 1/6 for each outcome, we set up the vector `probability` by using the function `rep()` which replicates a given value a specified number of times.

```
# generate the vector of probabilities
probability <- rep(1/6, 6)

# plot the probabilites
plot(probability,
     main = "Probability Distribution",
     xlab = "outcomes"
     )
```



For the cumulative probability distribution we need the cumulative probabilities i.e. we need the cumulative sums of the vector `probability`. These sums can be computed using `cumsum()`.

```r
#generate the vector of cumulative probabilities
cum_probability <- cumsum(probability)

# plot the probabilites
plot(cum_probability,
     xlab = "outcomes",
     main = "Cumulative Probability Distribution"
     )
```



Bernoulli Trials

The set of elements from which `sample()` draws outcomes does not have to consist of numbers only. We might as well simulate coin tossing with outcomes H (heads) and T (tails).

```r
sample(c("H", "T"), 1)
```

```
## [1] "T"
```

The result of a single coin toss is a *Bernoulli* distributed random variable i.e. a variable with two possible distinct outcomes.

Imagine you are about to toss a coin 10 times in a row and wonder how likely it is to end up with a sequence of outcomes like

$$H\,H\,T\,T\,T\,H\,T\,T\,H\,H.$$

This is a typical example of what we call a *Bernoulli experiment* as it consists of $n = 10$ Bernoulli trials that are independent of each other and we are interested in the likelihood of observing $k = 5$ successes H that occur with probability $p = 0.5$ (assuming a fair coin) in each trial. Note that the order of the outcomes does not matter here.

It is a well known result that the number of successes k in a Bernoulli experiment follows a binomial distribution. We denote this as

$$k \sim B(n, p).$$

The probability of observing k successes in the experiment $B(n, p)$ is given by

$$f(k) = P(k) = \binom{n}{k} \cdot p^k \cdot q^{n-k} = \frac{n!}{k!(n-k)!} \cdot p^k \cdot q^{n-k}$$

where $\binom{n}{k}$ means the binomial coefficient.

In `R`, we can solve the problem stated above by means of the function `dbinom()` which calculates the probability of the binomial distribution given the parameters `x`, `size`, and `prob`, see `?binom`.

```
dbinom(x = 5,
       size = 10,
       prob = 0.5
       )
```

```
## [1] 0.2460938
```

We conclude that the probability of observing Head $k = 5$ times when tossing the coin $n = 10$ times is about 24.6%.

Now assume we are interested in $P(4 \leq k \leq 7)$ i.e. the probability of observing 4, 5, 6 or 7 successes for $B(10, 0.5)$. This is easily computed by providing a vector as the argument `x` in our call of `dbinom()` and summing up using `sum()`.

```
sum(
  dbinom(x = 4:7,
         size = 10,
         prob = 0.5
         )
  )
```

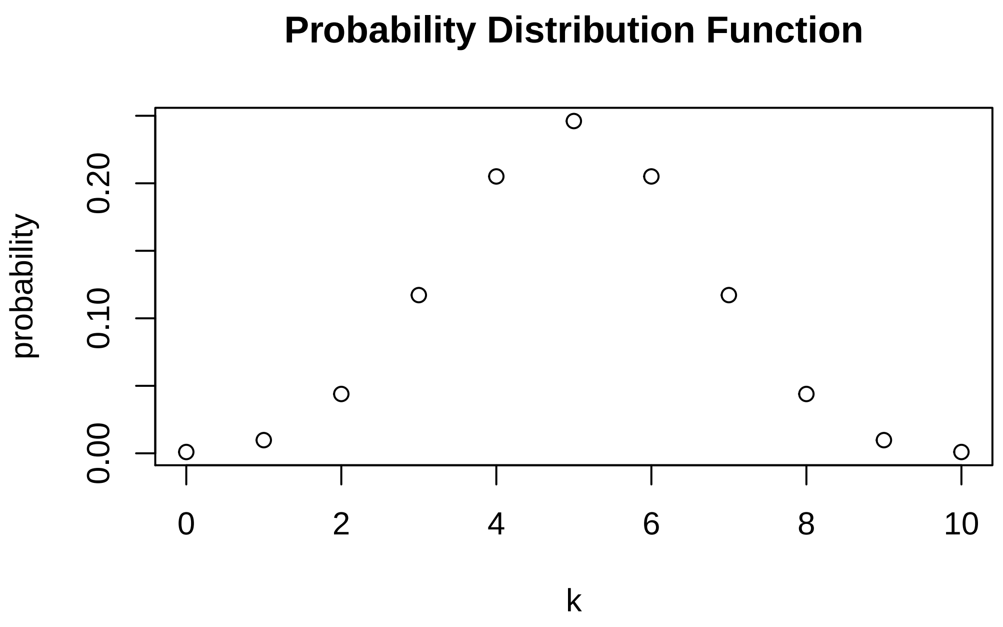
```
## [1] 0.7734375
```

The probability distribution of a discrete random variable is nothing but a list of all possible outcomes that can occur and their respective probabilities. In the coin tossing example we face 11 possible outcomes for k.

```
# set up vector of possible outcomes
k <- 0:10
```

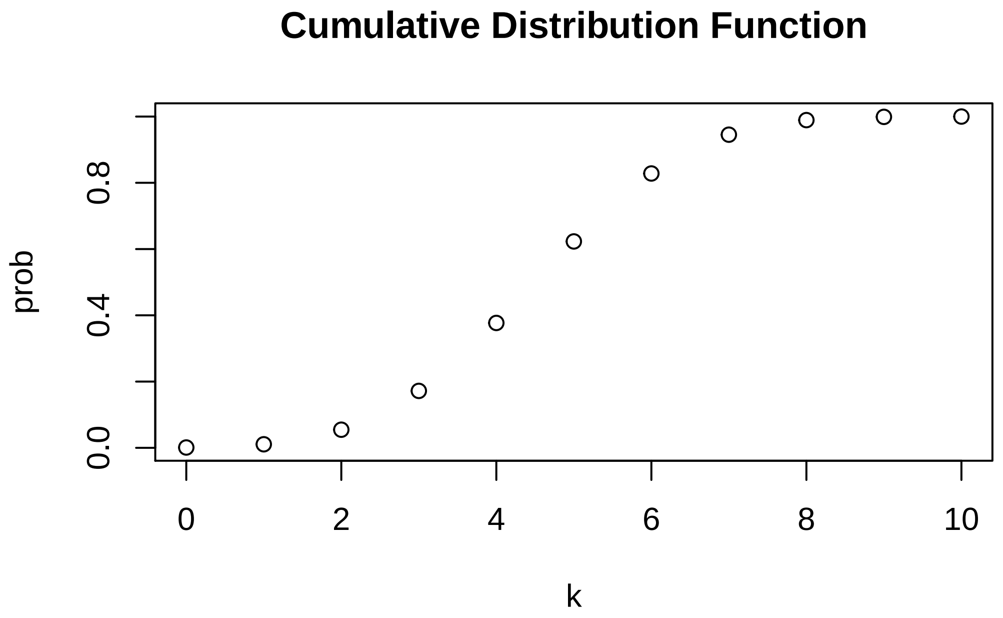
To visualize the probability distribution function of k we may therefore do the following:

```
# assign probabilities
probability <- dbinom(x = k,
                      size = 10,
                      prob = 0.5
                      )

# plot outcomes against probabilities
plot(x = k,
     y = probability,
     main = "Probability Distribution Function"
     )
```



In a similar fashion we may plot the cumulative distribution function of k by executing the following code chunk:

```r
# compute cumulative probabilities
prob <- cumsum(
              dbinom(x = 0:10,
                     size = 10,
                     prob = 0.5
                     )
              )

# plot the cumulative probabilities
plot(x = k,
     y = prob,
     main = "Cumulative Distribution Function"
     )
```



Expected Value, Mean and Variance

The expected value of a random variable is the long-run average value of its outcomes when the number of repeated trials is large. For a discrete random variable, the expected value is computed as a weighted average of its possible outcomes whereby the weights are the related probabilities. This is formalized in Key Concept 2.1.

Key Concept 2.1
Expected Value and the Mean

Suppose the random variable Y takes on k possible values, $y_1, \dots, y_k$, where y_1 denotes the first value, y_2 denotes the second value, and so forth, and that the probability that Y takes on y_1 is p_1, the probability that Y takes on y_2 is p_2 and so forth. The expected value of Y, $E(Y)$ is defined as

$$E(Y) = y_1 p_1 + y_2 p_2 + \cdots + y_k p_k = \sum_{i=1}^{k} y_i p_i$$

where the notation $\sum_{i=1}^{k} y_i p_i$ means "the sum of y_i p_i for i running from 1 to k". The expected value of Y is also called the mean of Y or the expectation of Y and is denoted by μ_y.

In the die example, the random variable, D say, takes on 6 possible values $d_1 = 1, d_2 = 2, \dots, d_6 = 6$. Assuming a fair die, each of the 6 outcomes occurs with a probability of 1/6. It is therefore easy to calculate the exact value of $E(D)$ by hand:

$$E(D) = 1/6 \sum_{i=1}^{6} d_i = 3.5$$

$E(D)$ is simply the average of the natural numbers from 1 to 6 since all wights p_i are 1/6. Convince yourself that this can be easily calculated using the function `mean()` which computes the arithmetic mean of a numeric vector.

```
mean(1:6)
```

```
## [1] 3.5
```

An example of sampling with replacement is rolling a die three times in a row.

```
# set random seed for reproducibility
set.seed(1)

# rolling a die three times in a row
sample(1:6, 3, replace = T)
```

```
## [1] 2 3 4
```

Of course we could also consider a much bigger number of trials, 10000 say. Doing so, it would be pointless to simply print the results to the console: by default `R` displays up to 1000 entries of large vectors and omits the remainder (give it a go). Eyeballing the numbers does not reveal too much. Instead let us calculate the sample average of the outcomes using `mean()` and see if the result comes close to the expected value $E(D) = 3.5$.

```
# set random seed for reproducibility
set.seed(1)

# compute the sample mean of 10000 die rolls
mean(
    sample(1:6,
           10000,
           replace = T
           )
    )
```

```
## [1] 3.5039
```

We find the sample mean to be fairly close to the expected value. This result will be discussed in Chapter 2.3 in more detail.

Other frequently encountered measures are the variance and the standard deviation. Both are measures of the *dispersion* of a random variable.

Key Concept 2.2
Variance and Standard Deviation

The Variance of the discrete *random variable* Y, denoted σ^2_Y, is

$$\sigma^2_Y = \text{Var}(Y) = E\left[(Y-\mu_y)^2\right] = \sum_{i=1}^{k}(y_i-\mu_y)^2 p_i$$

The standard deviation of Y is σ_Y, the square root of the variance. The units of the standard deviation are the same as the units of Y.

The variance as defined in Key Concept 2.2 *is not* implemented as a function in R. Instead we have the function `var()` which computes the *sample variance*

$$s_Y^2 = \frac{1}{n-1} \sum_{i=1}^{n} (y_i - \overline{y})^2.$$

Remember that s_Y^2 is different from the so called *population variance* of Y,

$$\text{Var}(Y) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \mu_Y)^2,$$

since it measures how the data is dispersed around the sample average $\overline{y}$ instead of the population mean μ_Y. This becomes clear when we look at our die rolling example. For D we have

$$\text{Var}(D) = 1/6 \sum_{i=1}^{6} (d_i - 3.5)^2 = 2.92$$

which is obviously different from the result of s^2 as computed by `var()`.

```
var(1:6)
```

```
## [1] 3.5
```

The sample variance as computed by `var()` is an *estimator* of the population variance.

# You may use this widget to play around with the functions presented above

2.2 Probability Distributions of Continuous Random Variables

Since a continuous random variable takes on a continuum of possible values, we cannot use the concept of a probability distribution as used for discrete random variables. Instead, the probability distribution of a continuous random variable is summarized by its *probability density function* (PDF).

The cumulative probability distribution function (CDF) for a continuous random variable is defined just as in the discrete case. Hence, the cumulative probability distribution of a continuous random variables states the probability that the random variable is less than or equal to a particular value.

For completeness, we present revisions of Key Concepts 2.1 and 2.2 for the continuous case.

Key Concept 2.3
Probabilities, Expected Value and Variance of a Continuous Random Variable

Let $f_Y(y)$ denote the probability density function of Y. Because probabilities cannot be negative, we have $f_Y \geq 0$ for all y. The Probability that Y falls between a and b where $a < b$ is

$$P(a \leq Y \leq b) = \int_a^b f_Y(y)\mathrm{d}y.$$

We further have that $P(-\infty \leq Y \leq \infty) = 1$ and therefore $\int_{-\infty}^{\infty} f_Y(y)\mathrm{d}y = 1$.
As for the discrete case, the expected value of Y is the probability weighted average of its values. Due to continuity, we use integrals instead of sums.
The expected value of Y is defined as

$$E(Y) = \mu_Y = \int y f_Y(y)\mathrm{d}y.$$

The variance is the expected value of $(Y - \mu_Y)^2$. We thus have

$$\text{Var}(Y) = \sigma_Y^2 = \int (y - \mu_Y)^2 f_Y(y)\mathrm{d}y.$$

Let us discuss an example:

Consider the continuous random variable X with probability density function

$$f_X(x) = \frac{3}{x^4}, x > 1.$$

- We can show analytically that the integral of $f_X(x)$ over the real line equals 1.

$$\begin{aligned}
\int f_X(x)\mathrm{d}x =& \int_1^\infty \frac{3}{x^4}\mathrm{d}x && (2.1)\\
=& \lim_{t\to\infty} \int_1^t \frac{3}{x^4}\mathrm{d}x && (2.2)\\
=& \lim_{t\to\infty} -x^{-3}|_{x=1}^t && (2.3)\\
=& -\left(\lim_{t\to\infty} \frac{1}{t^3} - 1\right) && (2.4)\\
=& 1 && (2.5)
\end{aligned}$$

- The expectation of X can be computed as follows:

$$\begin{aligned}
E(X) = \int x \cdot f_X(x)\mathrm{d}x =& \int_1^\infty x \cdot \frac{3}{x^4}\mathrm{d}x && (2.6)\\
=& -\frac{3}{2}x^{-2}|_{x=1}^\infty && (2.7)\\
=& -\frac{3}{2}\left(\lim_{t\to\infty} \frac{1}{t^2} - 1\right) && (2.8)\\
=& \frac{3}{2} && (2.9)
\end{aligned}$$

- Note that the variance of X can be expressed as $\text{Var}(X) = E(X^2) - E(X)^2$. Since $E(X)$ has been computed in the previous step, we seek $E(X^2)$:

$$\begin{aligned} E(X^2) = \int x^2 \cdot f_X(x)\mathrm{d}x = \int_1^\infty x^2 \cdot \frac{3}{x^4}\mathrm{d}x & \qquad (2.10)\\ = -3x^{-1}|_{x=1}^{\infty} & \qquad (2.11)\\ = -3\left(\lim_{t\to\infty}\frac{1}{t} - 1\right) & \qquad (2.12)\\ =3 & \qquad (2.13) \end{aligned}$$

So we have shown that the area under the curve equals one, that the expectation is $E(X) = \frac{3}{2}$ and we found the variance to be $\text{Var}(X) = \frac{3}{4}$. However, this was quite tedious and, as we shall see soon, an analytic approach is not applicable for some probability density functions e.g. if integrals have no closed form solutions.

Luckily, `R` enables us to find the results derived above in an instant. The tool we use for this is the function `integrate()`. First, we have to define the functions we want to calculate integrals for as `R` functions, i.e. the PDF $f_X(x)$ as well as the expressions $x \cdot f_X(x)$ and $x^2 \cdot f_X(x)$.

```
# define functions
f <- function(x) 3/x^4
g <- function(x) x*f(x)
h <- function(x) x^2*f(x)
```

Next, we use `integrate()` and set lower and upper limits of integration to 1 and ∞ using arguments `lower` and `upper`. By default, `integrate()` prints the result along with an estimate of the approximation error to the console. However, the outcome is not a numeric value one can do further calculation with readily. In order to get only a numeric value of the integral, we need to use the `$` operator in conjunction with `value`.

```
# calculate area under curve
AUC <- integrate(f,
                 lower = 1,
                 upper = Inf
                 )
AUC
```

```
## 1 with absolute error < 1.1e-14
```

```
# calculate E(X)
EX <- integrate(g,
                lower = 1,
                upper = Inf
                )
EX
```

```
## 1.5 with absolute error < 1.7e-14
```

```
# calculate Var(X)
VarX <- integrate(h,
                  lower = 1,
                  upper = Inf
                  )$value - EX$value^2
VarX
```

```
## [1] 0.75
```

Although there is a wide variety of distributions, the ones most often encountered in econometrics are the normal, chi-squared, Student t and F distributions. Therefore we will discuss some core `R` functions that allow to do calculations involving densities, probabilities and quantiles of these distributions.

Every probability distribution that `R` handles has four basic functions whose names consist of a prefix followed by a root name. As an example, take the normal distribution. The root name of all four functions associated with the normal distribution is `norm`. The four prefixes are

- `d` for "density" - probability function / probability density function
- `p` for "probability" - cumulative distribution function
- `q` for "quantile" - quantile function (inverse cumulative distribution function)
- `r` for "random" - random number generator

Thus, for the normal distribution we have the `R` functions `dnorm()`, `pnorm()`, `qnorm()` and `rnorm()`.

The Normal Distribution

The probably most important probability distribution considered here is the normal distribution. This is not least due to the special role of the standard normal distribution and the Central Limit Theorem which is treated shortly during the course of this section. Distributions of the normal family have a familiar symmetric, bell-shaped probability density. A normal distribution is characterized by its mean μ and its standard deviation σ what is concisely expressed by $N(\mu, \sigma^2)$. The normal distribution has the PDF

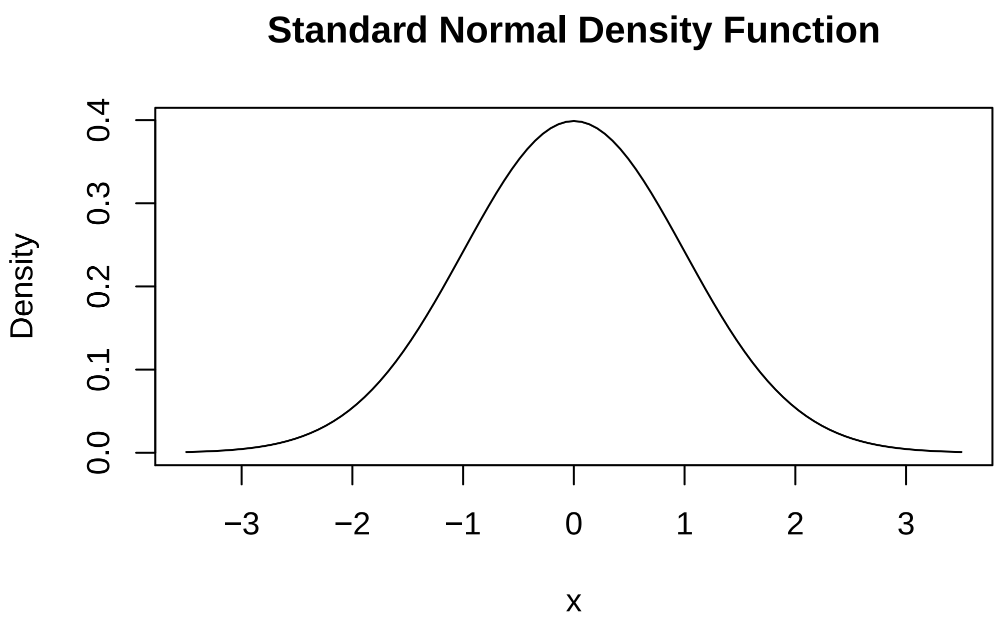
$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-)^2/(2^2)}. \tag{2.14}$$

For the standard normal distribution we have $\mu = 0$ and $\sigma = 1$. Standard normal variates are often denoted by Z. Usually, the standard normal PDF is denoted by ϕ and the standard normal CDF is denoted by Φ. Hence,

$$\phi(c) = \Phi'(c) \quad , \quad \Phi(c) = P(Z \leq c) \quad , \quad Z \sim N(0,1).$$

In `R`, we can conveniently obtain density values of normal distributions using the function `dnorm()`. Let us draw a plot of the standard normal density function using `curve()` in conjunction with `dnorm()`.

```r
# draw a plot of the N(0,1) PDF
curve(dnorm(x),
      xlim=c(-3.5, 3.5),
      ylab = "Density",
      main = "Standard Normal Density Function"
      )
```



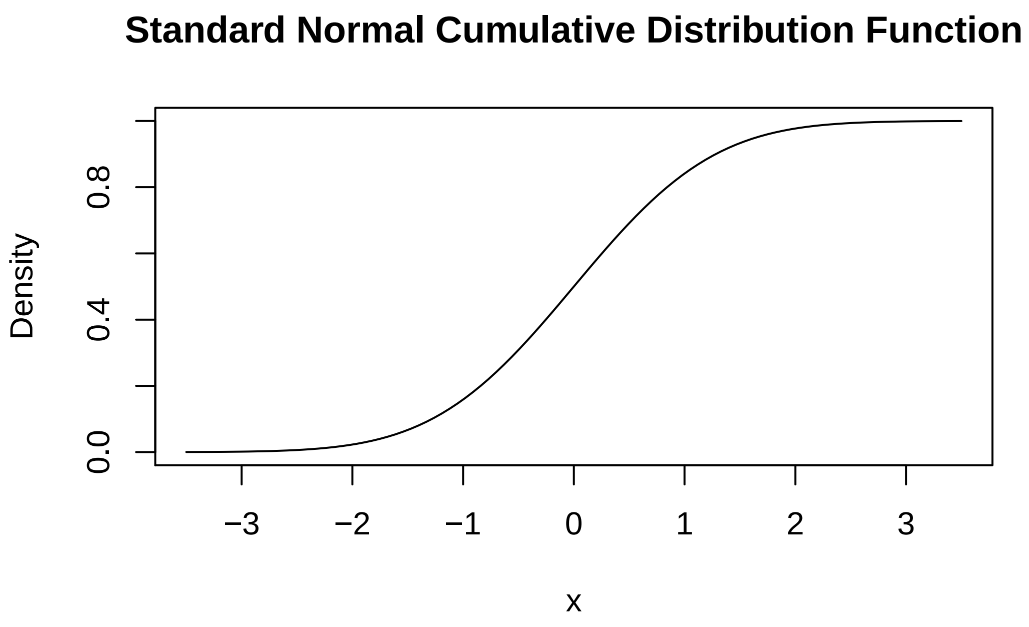
We can obtain the density at different positions by passing a vector of quantiles to `dnorm()`.

```r
# compute denstiy at x=-1.96, x=0 and x=1.96
dnorm(x = c(-1.96, 0, 1.96))
```

```
## [1] 0.05844094 0.39894228 0.05844094
```

Similar to the PDF, we can plot the standard normal CDF using `curve()` and `pnorm()`.

```r
# plot the standard normal CDF
curve(pnorm(x),
      xlim=c(-3.5, 3.5),
      ylab = "Density",
      main = "Standard Normal Cumulative Distribution Function"
      )
```



We can also use R to calculate the probability of events associated with a standard normal variate.

Let us say we are interested in $P(Z \leq 1.337)$. For some continuous random variable Z on $[-\infty, \infty]$ with density function $g(x)$ we would have to determine $G(x)$ which is the anti derivative of $g(x)$ so that

$$P(Z \leq 1,337) = G(1,337) = \int_{-\infty}^{1,337} g(x)\mathrm{d}x.$$

If $Z \sim N(0,1)$, we have $g(x) = \phi(x)$. There is no analytic solution to the integral above and it is cumbersome to come up with an approximation. However, we may circumvent this using R in different ways. The first approach makes use of the function `integrate()` which allows to solve one-dimensional integration problems using a numerical method. For this, we first define the function we want to compute the integral of as a R function `f`. In our example, `f` needs to be the standard normal density function and hence takes a single argument `x`. Following the definition of $\phi(x)$ we define `f` as

```r
# define the standard normal PDF as a R function
f <- function(x) {
  1/(sqrt(2 * pi)) * exp(-0.5 * x^2)
}
```

Let us check if this function enables us to compute standard normal density values by passing it a vector of quantiles.

```r
# define vector of quantiles
quants <- c(-1.96, 0, 1.96)

# compute density values
f(quants)
```

```
## [1] 0.05844094 0.39894228 0.05844094
```

```r
# compare to results produced by dnorm()
f(quants) == dnorm(quants)
```

```
## [1] TRUE TRUE TRUE
```

Notice that the results produced by `f()` are indeed equivalent to those given by `dnorm()`.

Next, we call `integrate()` on `f()` and specify the arguments `lower` and `upper`, the lower and upper limits of integration.

```
# integrate f()
integrate(f,
          lower = -Inf,
          upper = 1.337
          )
```

```
## 0.9093887 with absolute error < 1.7e-07
```

We find that the probability of observing $Z \leq 1.337$ is about 0.9094%.

A second and much more convenient way is to use the function `pnorm()` which also allows calculus involving the standard normal cumulative distribution function.

```
# compute the probability using pnorm()
pnorm(1.337)
```

```
## [1] 0.9093887
```

The result matches the outcome of the approach using `integrate()`.

Let us discuss some further examples:

A commonly known result is that 95% probability mass of a standard normal lies in the interval $[-1.96, 1.96]$, that is in a distance of about 2 standard deviations to the mean. We can easily confirm this by calculating

$$P(-1.96 \leq Z \leq 1.96) = 1 - 2 \times P(Z \leq -1.96)$$

due to symmetry of the standard normal PDF. Thanks to `R`, we can abandon the table of the standard normal CDF again and instead solve this by using the function `pnorm()`.

```
# compute the probability
1 - 2 * (pnorm(-1.96))
```

```
## [1] 0.9500042
```

Now consider a random variable Y with $Y \sim N(5, 25)$. As you should already know from your statistics courses it is not possible to make any statement of probability without prior standardizing as shown in Key Concept 2.4.

Key Concept 2.4
Computing Probabilities Involving Normal Random Variables

Suppose Y is normally distributed with mean μ and variance σ^2:

$$Y \sim N(\mu, \sigma^2)$$

Then Y is standardized by subtracting its mean and dividing by its standard deviation:

$$Z = \frac{Y - \mu}{\sigma}$$

Let c_1 and c_2 denote two numbers whereby $c_1 < c_2$ and further $d_1 = (c_1 - \mu)/\sigma$ and $d_2 = (c_2 - \mu)/\sigma$. Then

$$P(Y \leq c_2) = P(Z \leq d_2) = \Phi(d_2) \tag{2.15}$$
$$P(Y \geq c_1) = P(Z \geq d_1) = 1 - \Phi(d_1) \tag{2.16}$$
$$P(c_1 \leq Y \leq c_2) = P(d_1 \leq Z \leq d_2) = \Phi(d_2) - \Phi(d_1). \tag{2.17}$$

`R` functions that handle the normal distribution can perform this standardization. If we are interested in $P(3 \leq Y \leq 4)$ we can use `pnorm()` and adjust for a mean and/or a standard deviation that deviate from $\mu = 0$ and $\sigma = 1$ by specifying the arguments `mean` and `sd` accordingly. **Attention**: the argument `sd` requires the standard deviation, not the variance!

```r
pnorm(4, mean = 5, sd = 5) - pnorm(3, mean = 5, sd = 5)
```

```
## [1] 0.07616203
```

An extension of the normal distribution in a univariate setting is the multivariate normal distribution. The PDF of two random normal variables X and Y is given by

$$g_{X,Y}(x, y) = \frac{1}{2\pi\sigma_X\sigma_Y\sqrt{1-\rho_{XY}^2}} \tag{2.18}$$
$$\cdot \exp\left\{\frac{1}{-2(1-\rho_{XY}^2)}\left[\left(\frac{x-\mu_x}{\sigma_x}\right)^2 - 2\rho_{XY}\left(\frac{x-\mu_X}{\sigma_X}\right)\left(\frac{y-\mu_Y}{\sigma_Y}\right) + \left(\frac{y-\mu_Y}{\sigma_Y}\right)^2\right]\right\}. \tag{2.19}$$

Equation (2.19) contains the bivariate normal PDF. Admittedly, it is hard to gain insights from this complicated expression. Instead, let us consider the special case where X and Y are uncorrelated standard normal random variables with density functions $f_X(x)$ and $f_Y(y)$ and we assume that they have a joint normal distribution. We then have the parameters $\sigma_X = \sigma_Y = 1$, $\mu_X = \mu_Y = 0$ (due to marginal standard normality) and $\rho_{XY} = 0$ (due to independence). The joint probability density function of X and Y then becomes

$$g_{X,Y}(x, y) = f_X(x)f_Y(y) = \frac{1}{2\pi} \cdot \exp\left\{-\frac{1}{2}\left[x^2 + y^2\right]\right\}, \tag{2.2}$$

the PDF of the bivariate standard normal distribution. The next plot provides an interactive three-dimensional plot of (2.2). By moving the cursor over the plot you can see that the density is rotationally invariant.

The Chi-Squared Distribution

Another distribution relevant in econometric day-to-day work is the chi-squared distribution. It is often needed when testing special types of hypotheses frequently encountered when dealing with regression models.

The sum of M squared independent standard normal distributed random variables follows a chi-squared distribution with M degrees of freedom.

$$Z_1^2 + \cdots + Z_M^2 = \sum_{m=1}^{M} Z_m^2 \sim \chi_M^2 \ \text{ with } \ Z_m \overset{i.i.d.}{\sim} N(0,1)$$

A χ^2 distributed random variable with M degrees of freedom has expectation M, mode at $M-2$ for $n \geq 2$ and variance $2 \cdot M$.

For example, if we have

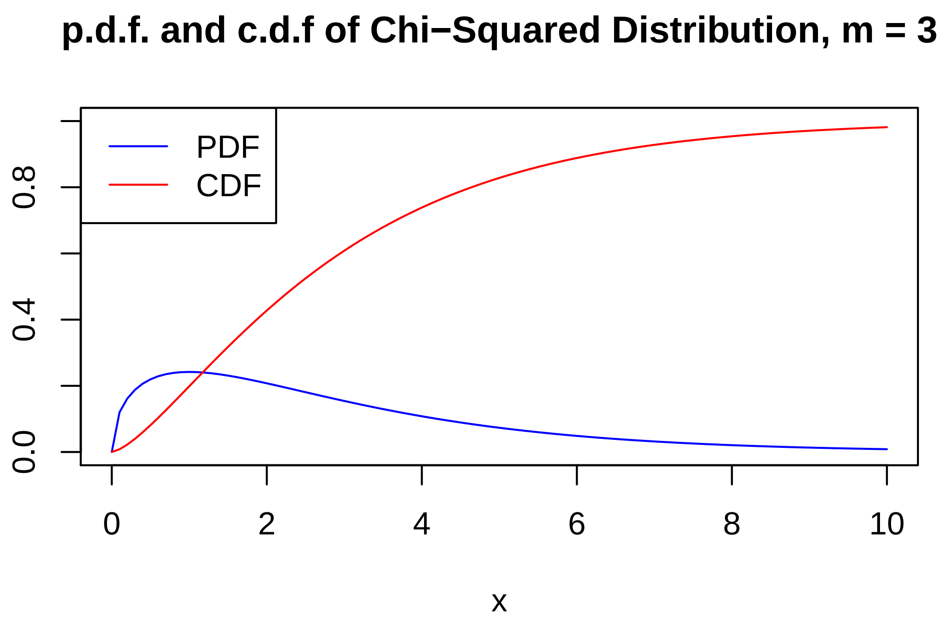
$$Z_1, Z_2, Z_3 \overset{i.i.d.}{\sim} N(0,1)$$

it holds that

$$Z_1^2 + Z_2^2 + Z_3^3 \sim \chi_3^2. \tag{2.3}$$

Using the code below, we can display the PDF and the CDF of a χ_3^2 random variable in a single plot. This is achieved by setting the argument `add = TRUE` in the second call of `curve()`. Further we adjust limits of both axes using `xlim` and `ylim` and choose different colors to make both functions better distinguishable. The plot is completed by adding a legend with help of the function `legend()`.

```r
# plot the PDF
curve(dchisq(x, df=3),
      xlim = c(0, 10),
      ylim = c(0, 1),
      col = "blue",
      ylab = "",
      main = "p.d.f. and c.d.f of Chi-Squared Distribution, m = 3"
      )

# add the CDF to the plot
curve(pchisq(x, df = 3),
      xlim=c(0, 10),
      add = TRUE,
      col = "red"
      )

# add a legend to the plot
legend("topleft",
       c("PDF", "CDF"),
       col = c("blue", "red"),
       lty = c(1, 1)
       )
```

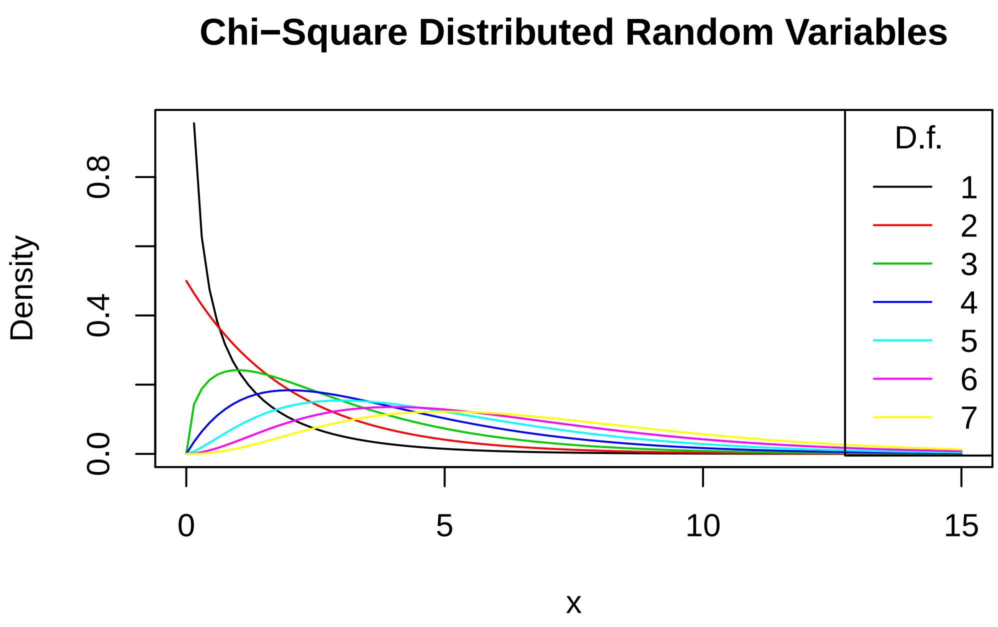


Since the outcomes of a χ^2_M distributed random variable are always positive, the domain of the related PDF and CDF is $\mathbb{R}_{\geq 0}$.

As expectation and variance depend (solely!) on the degrees of freedom, the distribution's shape changes drastically if we vary the number of squared standard normals that are summed up. This relation is often depicted by overlaying densities for different M, see e.g. the Wikipedia Article.

Of course, one can easily reproduce such a plot using `R`. Again we start by plotting the density of the χ^2_1 distribution on the interval $[0, 15]$ with `curve()`. In the next step, we loop over degrees of freedom $m = 2, ..., 7$ and add a density curve for each m to the plot. We also adjust the line color for each iteration of the loop by setting `col = m`. At last, we add a legend that displays degrees of freedom and the associated colors.

```r
# plot the density for m=1
curve(dchisq(x, df = 1),
      xlim=c(0, 15),
      xlab = "x",
      ylab = "Density",
      main = "Chi-Square Distributed Random Variables"
      )

# add densities for m=2,...,7 to the plot using a for loop
for (m in 2:7) {
  curve(dchisq(x, df = m),
        xlim = c(0, 15),
        add = T,
        col = m
        )
}

# add a legend
legend("topright",
       as.character(1:7),
       col = 1:7 ,
       lty = 1,
       title = "D.f."
       )
```



It is evident that increasing the degrees of freedom shifts the distribution to the right (the mode becomes larger) and increases the dispersion (the distribution's variance grows).

The Student t Distribution

Let Z be a standard normal variate, W a random variable that follows a χ^2_M distribution with M degrees of freedom and further assume that Z and W are independently distributed. Then it holds that

$$\frac{Z}{\sqrt{W/M}} =: X \sim t_M$$

and we say that X follows a *Student t distribution* (or simply t distribution) with M degrees of freedom.

As for the χ^2_M distribution, the shape of a t_M distribution depends on M. t distributions are symmetric, bell-shaped and look very similar to a normal distribution, especially when M is large. This is not a coincidence: for a sufficient large M, the t_M distribution can be approximated by the standard normal distribution. This approximation works reasonably well for $M \geq 30$. As we will show later by means of a small simulation study, the t_∞ distribution *is* the standard normal distribution.

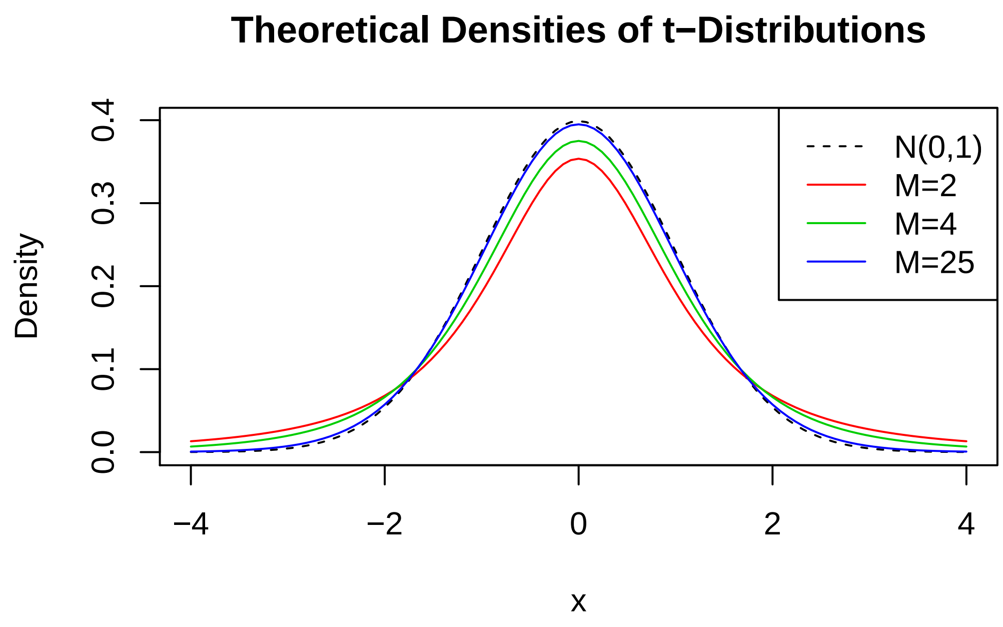
A t_M distributed random variable has an expectation if $M > 1$ and it has a variance if $n > 2$.

$$E(X) = 0 \ , \ M > 1 \tag{2.20}$$

$$\text{Var}(X) = \frac{M}{M-2} \ , \ M > 2 \tag{2.21}$$

Let us graph some t distributions with different M and compare them with the standard normal distribution.

```r
# plot the standard normal density
curve(dnorm(x),
      xlim = c(-4,4),
      xlab = "x",
      lty = 2,
      ylab = "Density",
      main = "Theoretical Densities of t-Distributions"
      )

# plot the t density for m=2
curve(dt(x, df = 2),
      xlim = c(-4, 4),
      col = 2,
      add = T
      )

# plot the t density for m=4
curve(dt(x, df = 4),
      xlim = c(-4, 4),
      col = 3,
      add = T
      )

# plot the t density for m=25
curve(dt(x, df = 25),
      xlim = c(-4, 4),
      col = 4,
      add = T
      )

# add a legend
legend("topright",
       c("N(0,1)","M=2","M=4","M=25"),
       col = 1:4,
       lty = c(2, 1, 1, 1)
       )
```



The plot indicates what has been claimed in the previous paragraph: as the degrees of freedom increase, the shape of the t distribution comes closer to that of a standard normal bell. Already for $M = 25$ we find little difference to the dashed line which is the standard normal density curve. If M is small, we find the distribution to have slightly heavier tails than a standard normal, i.e. it has a "fatter" bell shape.

The F Distribution

Another ratio of random variables important to econometricians is the ratio of two independently χ^2 distributed random variables that are divided by their degrees of freedom M and n. The quantity

$$\frac{W/M}{V/n} \sim F_{M,n} \text{ with } W \sim \chi^2_M \ , \ V \sim \chi^2_n$$

follows an F distribution with numerator degrees of freedom M and denominator degrees of freedom n, denoted $F_{M,n}$. The distribution was first derived by George Snedecor but was named in honor of Sir Ronald Fisher.

By definition, the domain of both PDF and CDF of an $F_{M,n}$ distributed random variable is $\mathbb{R}_{\geq 0}$.

Say we have an F distributed random variable Y with numerator degrees of freedom 3 and denominator degrees of freedom 14 and are interested in $P(Y \geq 2)$. This can be computed with help of the function `pf()`. By setting the argument `lower.tail` to `TRUE` we ensure that `R` computes $1 - P(Y \leq 2)$, i.e. the probability mass in the tail right of 2.

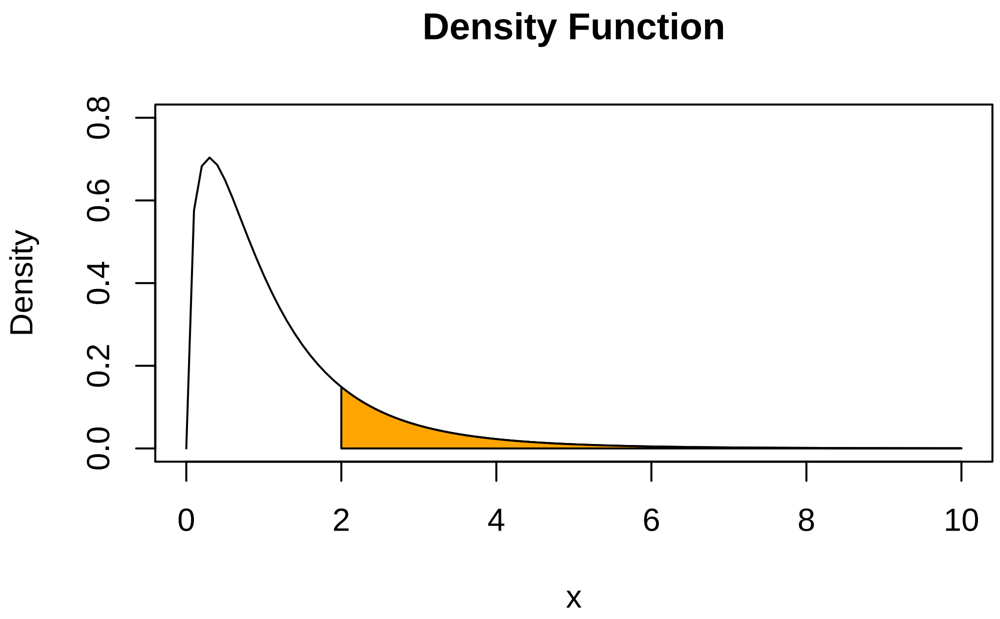
```
pf(2, 3, 13, lower.tail = F)
```

```
## [1] 0.1638271
```

We can visualize this probability by drawing a line plot of the related density function and adding a color shading with `polygon()`.

```
# define coordinate vectors for vertices of the polygon
x <- c(2, seq(2, 10, 0.01), 10)
```

```r
y <- c(0, df(seq(2, 10, 0.01), 3, 14), 0)

# draw density of F_{3, 14}
curve(df(x ,3 ,14),
      ylim = c(0, 0.8),
      xlim = c(0, 10),
      ylab = "Density",
      main = "Density Function"
      )

# draw the polygon
polygon(x, y, col="orange")
```



The F distribution is related to many other distributions. An important special case encountered in econometrics arises if the denominator degrees of freedom are large such that the $F_{M,n}$ distribution can be approximated by the $F_{M,\infty}$ distribution which turns out to be simply the distribution of a χ^2_M random variable divided by its degrees of freedom M,

$$W/M \sim F_{M,\infty} \quad , \quad W \sim \chi^2_M.$$

# You may use this widget to play around with the functions presented above

2.3 Random Sampling and the Distribution of Sample Averages

To clarify the basic idea of random sampling, let us jump back to the die rolling example:

Suppose we are rolling the die n times. This means we are interested in the outcomes of n random processes $Y_i,\ i = 1, ..., n$ which are characterized by the same distribution. Since these outcomes are selected randomly, they are *random variables* themselves and their realizations will differ each time we draw a sample, i.e. each time we roll the die n times. Furthermore, each observation is randomly drawn from the same population, that is the numbers from 1 to 6, and their individual distribution is the same. Hence we say that $Y_1, \dots, Y_n$ are identically distributed. Moreover, we know that the value of any of the Y_i does not provide any information on the remainder of the sample. In our example, rolling a six as the first observation in our sample does not alter the distributions of $Y_2, \dots, Y_n$: all numbers are equally likely to occur. This means that all Y_i are also independently distributed. Thus, we say that $Y_1, \dots, Y_n$ are independently and identically distributed ($i.i.d$). The die example uses the most simple sampling scheme. That is why it is called *simple random sampling*. This concept is condensed in Key Concept 2.5.

Key Concept 2.5
Simple Random Sampling and i.i.d. Random Variables

In simple random sampling, n objects are drawn at random from a population. Each object is equally likely to end up in the sample. We denote the value of the random variable Y for the i^{th} randomly drawn object as Y_i. Since all objects are equally likely to be drawn and the distribution of Y_i is the same for all i, the $Y_i, \dots, Y_n$ are independently and identically distributed (i.i.d.). This means the distribution of Y_i is the same for all $i = 1, \dots, n$ and Y_1 is distributed independently of $Y_2, \dots, Y_n$ and Y_2 is distributed independently of $Y_1, Y_3, \dots, Y_n$ and so forth.

What happens if we consider functions of the sample data? Consider the example of rolling a die two times in a row once again. A sample now consists of two independent random draws from the set $\{1, 2, 3, 4, 5, 6\}$. In view of the aforementioned, it is apparent that any function of these two random variables is also random, e.g. their sum. Convince yourself by executing the code below several times.

```
sum(sample(1:6, 2, replace = T))
```

```
## [1] 6
```

Clearly this sum, let us call it S, is a random variable as it depends on randomly drawn summands. For this example, we can completely enumerate all outcomes and hence write down the theoretical probability distribution of our function of the sample data, S:

We face $6^2 = 36$ possible pairs. Those pairs are

$$(1,1)(1,2)(1,3)(1,4)(1,5)(1,6) \tag{2.22}$$
$$(2,1)(2,2)(2,3)(2,4)(2,5)(2,6) \tag{2.23}$$
$$(3,1)(3,2)(3,3)(3,4)(3,5)(3,6) \tag{2.24}$$
$$(4,1)(4,2)(4,3)(4,4)(4,5)(4,6) \tag{2.25}$$
$$(5,1)(5,2)(5,3)(5,4)(5,5)(5,6) \tag{2.26}$$
$$(6,1)(6,2)(6,3)(6,4)(6,5)(6,6) \tag{2.27}$$

Thus, possible outcomes for S are

$$\{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}.$$

Enumeration of outcomes yields

$$P(S) = \begin{cases} 1/36, & S = 2 \\ 2/36, & S = 3 \\ 3/36, & S = 4 \\ 4/36, & S = 5 \\ 5/36, & S = 6 \\ 6/36, & S = 7 \\ 5/36, & S = 8 \\ 4/36, & S = 9 \\ 3/36, & S = 10 \\ 2/36, & S = 11 \\ 1/36, & S = 12 \end{cases} \tag{2.28}$$

We can also compute $E(S)$ and $\text{Var}(S)$ as stated in Key Concept 2.1 and Key Concept 2.2.

```
# Vector of outcomes
S <- 2:12

# Vector of probabilities
PS <- c(1:6, 5:1)/36

# Expectation of S
ES <- S %*% PS
ES
```

```
##      [,1]
## [1,]    7
```

```
# Variance of S
VarS <- (S - c(ES))^2 %*% PS
VarS
```

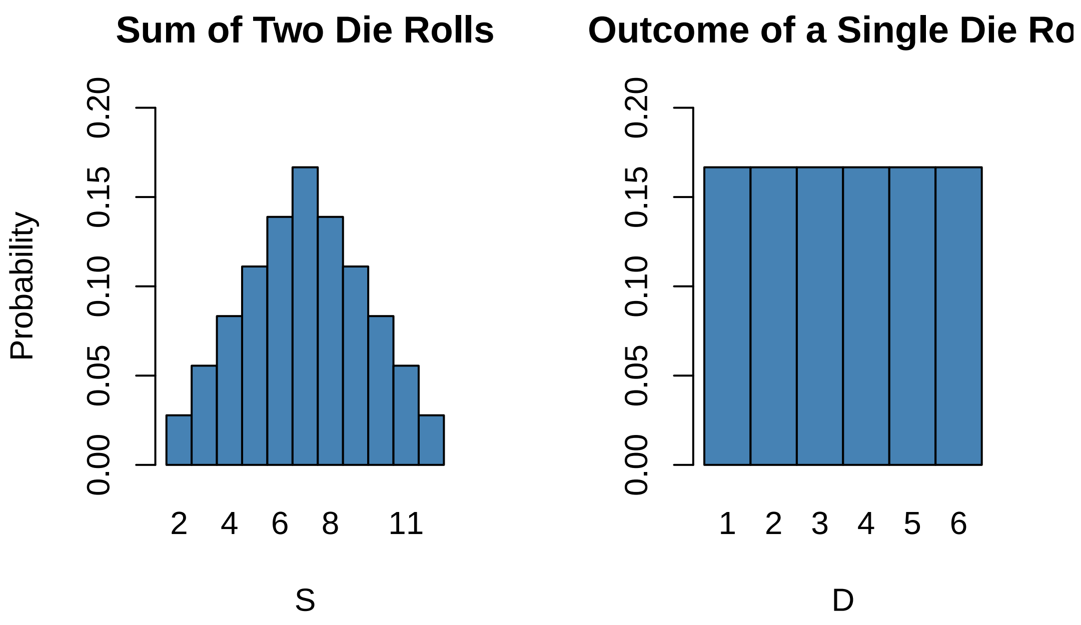
```
##          [,1]
## [1,] 5.833333
```

(The %*% operator is used to compute the scalar product of two vectors.)

So the distribution of S is known. It is also evident that its distribution differs considerably from the marginal distribution, i.e. the distribution of a single die roll's outcome, D . Let us visualize this using bar plots.

```
# divide the plotting area in one row with two columns
par(mfrow = c(1, 2))

# plot the distribution of S
names(PS) <- 2:12

barplot(PS, ylim = c(0, 0.2),
        xlab = "S",
        ylab = "Probability",
        col = "steelblue",
        space = 0,
        main = "Sum of Two Die Rolls"
        )

# plot the distribution of D
```

```r
probability <- rep(1/6, 6)
names(probability) <- 1:6

barplot(probability,
        ylim = c(0, 0.2),
        xlab = "D",
        col = "steelblue",
        space = 0,
        main = "Outcome of a Single Die Roll"
        )
```



Many econometric procedures deal with averages of sampled data. It is almost always assumed that observations are drawn randomly from a larger, unknown population. As demonstrated for the sample function S, computing an average of a random sample also has the effect to make the average a random variable itself. This random variable in turn has a probability distribution which is called the sampling distribution. Knowledge about the sampling distribution of the average is therefore crucial for understanding the performance of econometric procedures.

The *sample average* of a sample of n observations $Y_1, \dots, Y_n$ is

$$\overline{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i = \frac{1}{n}(Y_1 + Y_2 + \cdots + Y_n).$$

$\overline{Y}$ is also called the sample mean.

Mean and Variance of the Sample Mean

Denote μ_Y and σ_Y^2 the mean and the variance of the Y_i and suppose that all observations $Y_1, \dots, Y_n$ are i.i.d. such that in particular mean and variance are the same for all $i = 1, \dots, n$. Then we have that

$$E(\overline{Y}) = E\left(\frac{1}{n}\sum_{i=1}^{n} Y_i\right) = \frac{1}{n}E\left(\sum_{i=1}^{n} Y_i\right) = \frac{1}{n}\sum_{i=1}^{n} E\left(Y_i\right) = \frac{1}{n} \cdot n \cdot \mu_Y = \mu_Y$$

and

$$\begin{aligned}
\text{Var}(\overline{Y}) =& \text{Var}\left(\frac{1}{n}\sum_{i=1}^{n} Y_i\right) && (2.29)\\
=& \frac{1}{n^2}\sum_{i=1}^{n}\text{Var}(Y_i) + \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n}\text{cov}(Y_i, Y_j) && (2.30)\\
=& \frac{\sigma_Y^2}{n} && (2.31)\\
=& \sigma_{\overline{Y}}^2. && (2.32)
\end{aligned}$$

Note that the second summand vanishes since $\text{cov}(Y_i, Y_j) = 0$ for $i \neq j$ due to independence of the observations.

Consequently, the standard deviation of the sample mean is given by

$$\sigma_{\overline{Y}} = \frac{\sigma_Y}{\sqrt{n}}.$$

It is worthwhile to mention that these results hold irrespective of the underlying distribution of the Y_i.

The Sampling Distribution of $\overline{Y}$ when Y Is Normally Distributed

If the $Y_1, \dots, Y_n$ are i.i.d. draws from a normal distribution with mean μ_Y and variance σ_Y^2, the following holds for their sample average $\overline{Y}$:

$$\overline{Y} \sim N(\mu_y, \sigma_Y^2/n) \tag{2.4}$$

For example, if a sample Y_i with $i = 1, \dots, 10$ is drawn from a standard normal distribution with mean $\mu_Y = 0$ and variance $\sigma_Y^2 = 1$ we have

$$\overline{Y} \sim N(0, 0.1).$$

We can use `R`'s random number generation facilities to verify this result. The basic idea is to simulate outcomes of the true distribution of $\overline{Y}$ by repeatedly drawing random samples of 10 observation from the $N(0, 1)$ distribution and computing their respective averages. If we do this for a large number of repetitions, the simulated data set of averages should quite accurately reflect the theoretical distribution of $\overline{Y}$ if the theoretical result holds.

The approach sketched above is an example of what is commonly known as *Monte Carlo Simulation* or *Monte Carlo Experiment*. To perform this simulation in `R`, we proceed as follows:

1. Choose a sample size `n` and the number of samples to be drawn `reps`.
2. Use the function `replicate()` in conjunction with `rnorm()` to draw `n` observations from the standard normal distribution `rep` times. **Note**: the outcome of `replicate()` is a matrix with dimensions `n` $\times$ `rep`. It contains the drawn samples as *columns*.
3. Compute sample means using `colMeans()`. This function computes the mean of each column i.e. of each sample and returns a vector.

```
# Set sample size and number of samples
n <- 10
reps <- 10000

# Perform random sampling
samples <- replicate(reps, rnorm(n)) # 10 x 10000 sample matrix

# Compute sample means
sample.avgs <- colMeans(samples)
```

After performing these steps we end up with a vector of sample averages. You can check the vector property of `sample.avgs`:

```
# Check that sample.avgs is a vector
is.vector(sample.avgs)
```

```
## [1] TRUE
```

```
# print the first few entries to the console
head(sample.avgs)
```

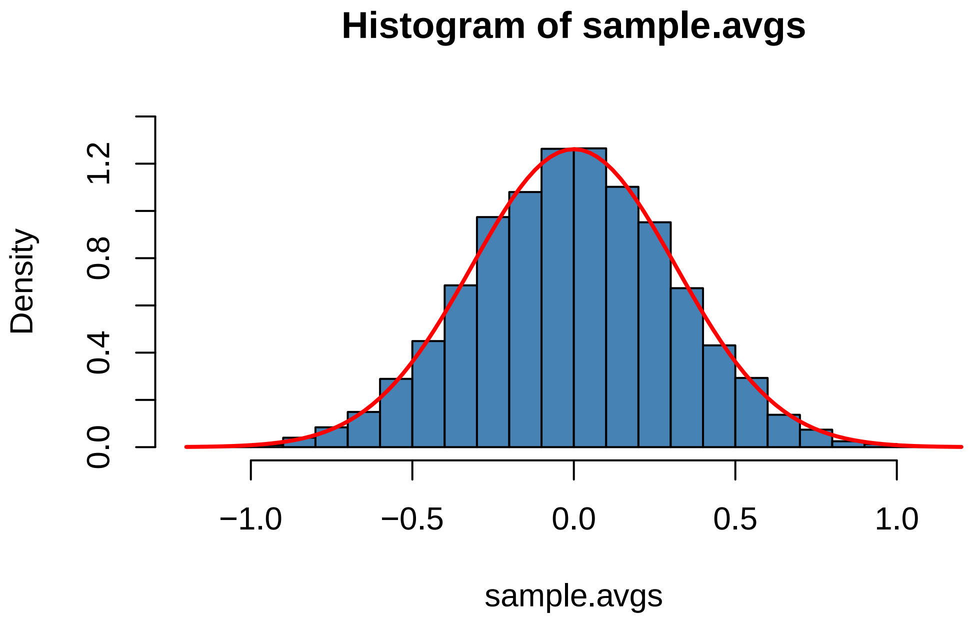
```
## [1] -0.12406767 -0.10649421 -0.01033423 -0.39905236 -0.41897968 -0.90883537
```

A straightforward approach to examine the distribution of univariate numerical data is to plot it as a histogram and compare it to some known or assumed distribution. This comparison can be done with help of a suitable statistical test or by simply eyeballing some graphical representations of these distributions. For our simulated sample averages, we will do the latter by means of the functions `hist()` and `curve()`.

By default, `hist()` will give us a frequency histogram i.e. a bar chart where observations are grouped into ranges, also called bins. The ordinate reports the number of observations falling into each of the bins. Instead, we want it to report density estimates for comparison purposes. This is achieved by setting the argument `freq = FALSE`. The number of bins is adjusted by the argument `breaks`.

Using `curve()`, we overlay the histogram with a red line which represents the theoretical density of a $N(0, 0.1)$ distributed random variable. Remember to use the argument `add = TRUE` to add the curve to the current plot. Otherwise `R` will open a new graphic device and discard the previous plot!

```
# Plot the density histogram
hist(sample.avgs,
     ylim = c(0, 1.4),
     col = "steelblue" ,
     freq = F,
     breaks = 20
     )

# overlay the theoretical distribution of sample averages on top of the histogram
curve(dnorm(x, sd = 1/sqrt(n)),
      col = "red",
      lwd = "2",
      add = T
      )
```



From inspection of the plot we can tell that the distribution of $\overline{Y}$ is indeed very close to that of a $N(0, 0.1)$ distributed random variable so that evidence obtained from the Monte Carlo Simulation supports the theoretical claim.

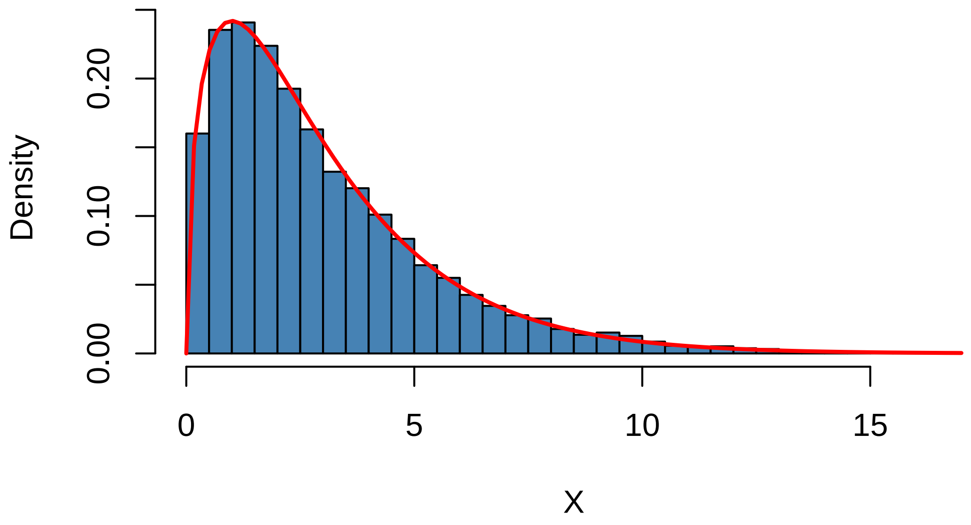
Let us discuss another example where using simple random sampling in a simulation setup helps to verify a well known result. As discussed before, the Chi-squared distribution with m degrees of freedom arises as the distribution of the sum of m independent squared standard normal distributed random variables.

To visualize the claim stated in equation (2.3), we proceed similarly as in the example before:

1. Choose the degrees of freedom `DF` and the number of samples to be drawn `reps`.
2. Draw `reps` random samples of size `DF` from the standard normal distribution using `replicate()`.
3. For each sample, by squaring the outcomes and summing them up column wise. Store the results

Again, we produce a density estimate for the distribution underlying our simulated data using a density histogram and overlay it with a line graph of the theoretical density function of the χ^2_3 distribution.

```r
# Number of repititions
reps <- 10000

# Set degrees of freedom of a chi-Square Distribution
DF <- 3

# Sample 10000 column vectors à 3 N(0,1) R.V.S
Z <- replicate(reps, rnorm(DF))

# Column sums of squares
X <- colSums(Z^2)

# Histogram of column sums of squares
hist(X,
     freq = F,
     col = "steelblue",
     breaks = 40,
     ylab = "Density",
     main = ""
     )

# Add theoretical density
curve(dchisq(x, df = DF),
      type = 'l',
      lwd = 2,
      col = "red",
      add = T
      )
```



Large Sample Approximations to Sampling Distributions

Sampling distributions as considered in the last section play an important role in the development of econometric methods. In general, there are two different approaches in characterizing sampling distributions: an "exact" approach and an "approximate" approach.

The exact approach aims to find a general formula for the sampling distribution that holds for any sample size n. We call this the *exact distribution* or *finite sample distribution*. In the previous examples of die rolling and normal variates, we have dealt with functions of random variables whose sample distributions are *excactly known* in the sense that we can write them down as analytic expressions and do calculations. However, this is not always possible. For $\overline{Y}$, result (2.4) tells us that normality of the Y_i implies normality of $\overline{Y}$ (we demonstrated this for the special case of $Y_i \overset{i.i.d.}{\sim} N(0,1)$ with $n = 10$ using a simulation study that involves simple random sampling). Unfortunately, the *exact* distribution of $\overline{Y}$ is generally unknown and often hard to derive (or even untraceable) if we drop the assumption that the Y_i have a normal distribution.

Therefore, as can be guessed from its name, the "approximate" approach aims to find an approximation to the sampling distribution whereby it is required that the sample size n is large. A distribution that is used as a large-sample approximation to the sampling distribution is also called the *asymptotic distribution*. This is due to the fact that the asymptotic distribution *is* the sampling distribution for $n \to \infty$ i.e. the approximation becomes exact if the sample size goes to infinity. However, there are cases where the difference between the sampling distribution and the asymptotic distribution is negligible for moderate or even small samples sizes so that approximations using the asymptotic distribution are reasonably good.

In this section we will discuss two well known results that are used to approximate sampling distributions and thus constitute key tools in econometric theory: the *law of large numbers* and the *central limit theorem*. The law of large numbers states that in large samples, $\overline{Y}$ is close to μ_Y with high probability. The central limit theorem says that the sampling distribution of the standardized sample average, that is $(\overline{Y} - \mu_Y)/\sigma_{\overline{Y}}$ is asymptotically normally distributed. It is particularly interesting that both results do not depend on the distribution of Y. In other words, being unable to describe the complicated sampling distribution of $\overline{Y}$ if Y is not normal, approximations of the latter using the central limit theorem simplify the development and applicability of econometric procedures enormously. This is a key component underlying the theory of statistical inference for regression models. Both results are summarized in Key Concept 2.6 and Key Concept 2.7.

Key Concept 2.6
Convergence in Probability, Consistency and the Law of Large Numbers

The sample average $\overline{Y}$ converges in probability to μ_Y — we say that $\overline{Y}$ is *consistent* for μ_Y — if the probability that $\overline{Y}$ is in the range $(\mu_Y - \epsilon)$ to $(\mu_Y + \epsilon)$ becomes arbitrary close to 1 as n increases for any constant $\epsilon > 0$. We write this short as

$$\overline{Y} \xrightarrow{p} \mu_Y.$$

Consider the independently and identically distributed random variables $Y_i, i = 1, \dots, n$ with expectation $E(Y_i) = \mu_Y$ and variance $\text{Var}(Y_i) = \sigma^2_Y$. Under the condition that $\sigma^2_Y < \infty$, that is large outliers are unlikely, the law of large numbers states

$$\overline{Y} \xrightarrow{p} \mu_Y.$$

The following application simulates a large number of coin tosses (you may set the number of trials using the slider) with a fair coin and computes the fraction of heads observed for each additional toss. The result is a random path that, as stated by the law of large numbers, shows a tendency to approach the value of 0.5 as n grows.

This interactive application is only available in the HTML version.

The core statement of the law of large numbers is that under quite general conditions, the probability of obtaining a sample average $\overline{Y}$ that is close to μ_Y is high if we have a large sample size.

Consider the example of repeatedly tossing a coin where Y_i is the result of the i^{th} coin toss. Y_i is a Bernoulli distributed random variable with

$$P(Y_i) = \begin{cases} p, & Y_i = 1 \\ 1 - p, & Y_i = 0 \end{cases}$$

where $p = 0.5$ as we assume a fair coin. It is straightforward to show that

$$\mu_Y = p = 0.5.$$

Say p is the probability of observing head and denote R_n the proportion of heads in the first n tosses,

$$R_n = \frac{1}{n} \sum_{i=1}^{n} Y_i. \tag{2.5}$$

According to the law of large numbers, the observed proportion of heads converges in probability to $\mu_Y = 0.5$, the probability of tossing head in a *single* coin toss,

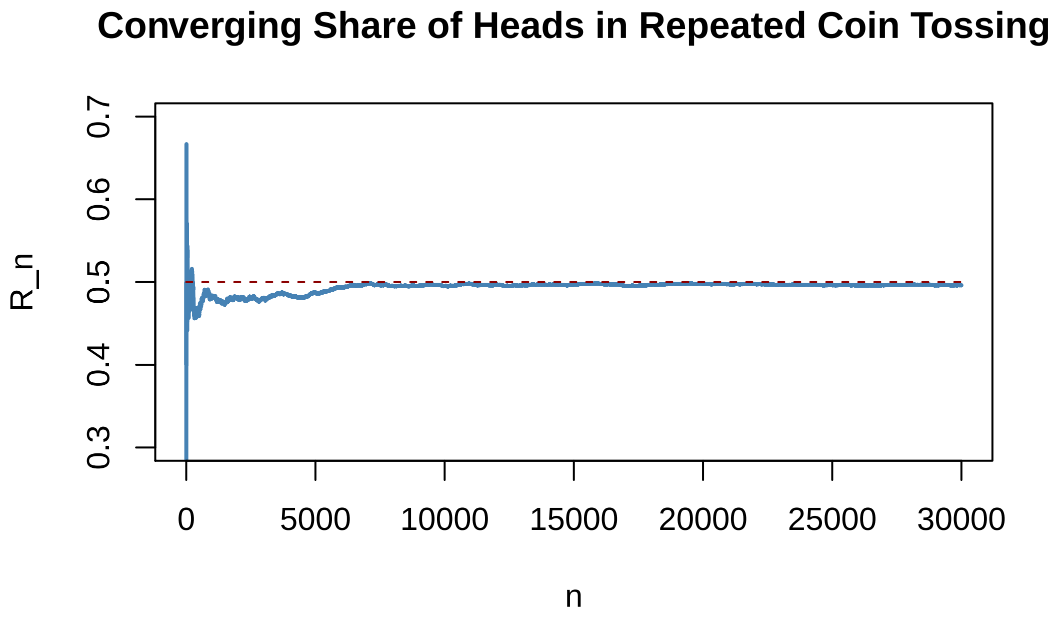
$$R_n \xrightarrow{p} \mu_Y = 0.5 \text{ as } n \to \infty.$$

We can use `R` to compute and illustrate such paths by simulation. The procedure is as follows:

1. Sample `N` observations from the Bernoulli distribution e.g. using `sample()`.
2. Calculate the proportion of heads R_n as in (2.5). A way to achieve this is to call `cumsum()` on the vector of observations `Y` to obtain its cumulative sum and then divide by the respective number of observations.

We continue by plotting the path and also add a dashed line for the benchmark probability $R_n = p = 0.5$.

```r
# set random seed
set.seed(1)

# Set number of coin tosses and simulate
N <- 30000
Y <- sample(0:1, N, replace =T)

# Calculate R_n for 1:N
S <- cumsum(Y)
R <- S/(1:N)

# Plot the path.
plot(R,
     ylim = c(0.3, 0.7),
     type = "l",
     col = "steelblue",
     lwd = 2,
     xlab = "n",
     ylab = "R_n",
     main = "Converging Share of Heads in Repeated Coin Tossing"
     )

# Add a dashed line for R_n = 0.5
lines(c(0, N),
      c(0.5, 0.5),
      col = "darkred",
      lty = 2,
      lwd = 1
      )
```



There are several things to be said about this plot.

- The blue graph shows the observed proportion of heads when tossing a coin n times.

- Since the Y_i are random variables, R_n is a random variate, too. The path depicted is only one of many possible realizations of R_n as it is determined by the 30000 observations sampled from the Bernoulli distribution. Thus, the code chunk above produces a different path every time you execute it (try this below!).

- If the number of coin tosses n is small, we observe the proportion of heads to be anything but close to its theoretical value, $\mu_Y = 0.5$. However, as more and more observation are included in the sample we find that the path stabilizes in neighborhood of 0.5. This is the message to take away: the average of multiple trials shows a clear tendency to converge to its expected value as the sample size increases, just as claimed by the law of large numbers.

Key Concept 2.7
The Central Limit Theorem

Suppose that $Y_1, \dots, Y_n$ are independently and identically distributed random variables with expectation $E(Y_i) = \mu_Y$ and variance $\text{Var}(Y_i) = \sigma^2_Y$ where $0 < \sigma^2_Y < \infty$.<br> The central Limit Theorem (CLT) states that, if the sample size n goes to infinity, the distribution of the standardized sample average

$$\frac{\overline{Y} - \mu_Y}{\sigma_{\overline{Y}}} = \frac{\overline{Y} - \mu_Y}{\sigma_Y/\sqrt{n}}$$

becomes arbitrarily well approximated by the standard normal distribution.
The application below demonstrates the CLT for the sample average of normally distributed random variables with mean 5 and variance 25^2. You may check the following properties:

- The distribution of the sample average is normal.
- As the sample size increases, the distribution of $\overline{Y}$ tightens around the true mean of 5.
- The distribution of the standardized sample average is close to the standard normal distribution for large n.

This interactive application is only available in the HTML version.

According to the central limit theorem, the distribution of the sample mean $\overline{Y}$ of the Bernoulli distributed random variables Y_i, $i = 1, ..., n$ is well approximated by the normal distribution with parameters $\mu_Y = p = 0.5$ and $\sigma^2_{\overline{Y}} = p(1-p) = 0.25$ for large n. Consequently, for the standardized sample mean we conclude that the ratio

$$\frac{\overline{Y} - 0.5}{0.5/\sqrt{n}} \tag{2.6}$$

should be well approximated by the standard normal distribution $N(0, 1)$. We employ another simulation study to demonstrate this graphically. The idea is as follows.

Draw a large number of random samples, 10000 say, of size n from the Bernoulli distribution and compute the sample averages. Standardize the averages as shown in (2.6). Next, visualize the distribution of the generated standardized sample averages by means of a density histogram and compare to the standard normal distribution. Repeat this for different sample sizes n to see how increasing the sample size n impacts the simulated distribution of the averages.

In R, we realized this as follows:

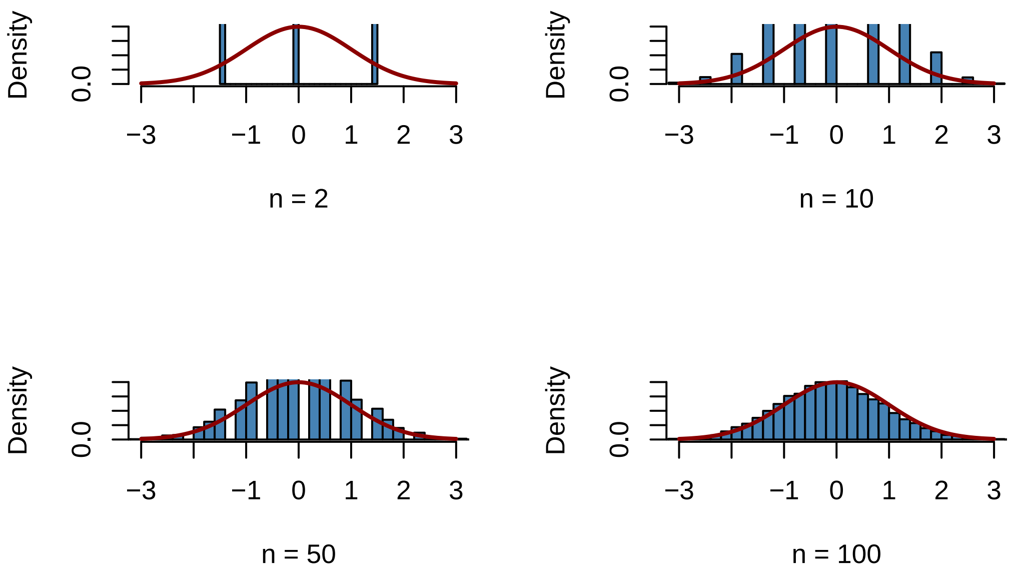
1. We start by defining that the next four subsequently generated figures shall be drawn in a 2×2 array such that they can be easily compared. This is done by calling `par(mfrow = c(2, 2))` before the figures are generated.

2. We define the number of repetitions `reps` as 10000 and create a vector of sample sizes named `sample.sizes`. We consider samples of sizes 2, 10, 50 and 100.

3. Next, we combine two `for()` loops to simulate the data and plot the distributions. The inner loop generates 10000 random samples, each consisting of `n` observations that are drawn from the Bernoulli distribution, and computes the standardized averages. The outer loop executes the inner loop for the different sample sizes `n` and produces a plot for each iteration.

```r
# Subdivide the plot panel into a 2-by-2 array
par(mfrow = c(2, 2))

# Set number of repetitions and the sample sizes
reps <- 10000
sample.sizes <- c(2, 10, 50, 100)
```

```r
# outer loop (loop over the sample sizes)
  for (n in sample.sizes) {

    samplemean <- rep(0, reps) #initialize the vector of sample menas
    stdsamplemean <- rep(0, reps) #initialize the vector of standardized sample menas

# inner loop (loop over repetitions)
    for (i in 1:reps) {
      x <- rbinom(n, 1, 0.5)
      samplemean[i] <- mean(x)
      stdsamplemean[i] <- sqrt(n)*(mean(x) - 0.5)/0.5
    }

# plot the histogram and overlay it with the N(0,1) density for every iteration
    hist(stdsamplemean,
         col = "steelblue",
         breaks = 40,
         freq = FALSE,
         xlim = c(-3, 3),
         ylim = c(0, 0.4),
         xlab = paste("n =", n),
         main = ""
         )

    curve(dnorm(x),
          lwd = 2,
          col = "darkred",
          add = TRUE
          )
  }
```



We see that the simulated sampling distribution of the standardized average tends to deviate strongly from the standard normal distribution if the sample size is small, e.g. for $n = 5$ and $n = 10$. However as n grows, the histograms are approaching the bell shape of a standard normal and we can be confident that the approximation works quite well as seen for $n = 100$.

2.4 Exercises

This interactive part of URFITE is only available in the HTML version

Chapter 3

A Review of Statistics using R

This section reviews important statistical concepts:

- Estimation of unknown population parameters
- Hypothesis testing
- Confidence intervals

Since these types of statistical methods are heavily used in econometrics, we will discuss them in the context of inference about an unknown population mean and discuss several applications in `R`. These `R` applications rely on the following packages which are not part of the base version of `R`:

- `readxl` - allows to import data from *Excel* to `R`.
- `dplyr` - provides a flexible grammar for data manipulation.
- `MASS` - a collection of functions for applied statistics.

Make sure these are installed before you go ahead and try to replicate the examples. The savest way to do so is by checking whether the following code chunk executes without any errors.

```
library(dplyr)
library(MASS)
library(readxl)
```

3.1 Estimation of the Population Mean

Key Concept 3.1
Estimators and Estimates

Estimators are functions of sample data that are drawn randomly from an unknown population. *Estimates* are numeric values computed by estimators based on the sample data. Estimators are random variables because they are functions of *random* data. Estimates are nonrandom numbers.

Think of some economic variable, for example hourly earnings of college graduates, denoted by Y. Suppose we are interested in μ_Y the mean of Y. In order to exactly calculate μ_Y we would have to interview every graduated member of the working population in the economy. We simply cannot do this for time and cost reasons. However, we could draw a random sample of n i.i.d. observations $Y_1, \dots, Y_n$ and estimate μ_Y using one of the simplest estimators in the sense of Key Concept 3.1 one can think of, that is

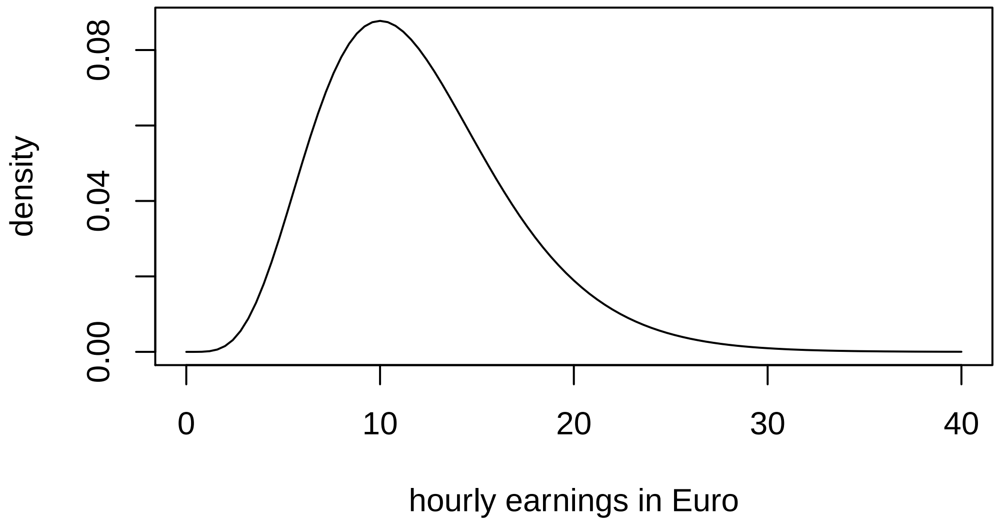
$$\overline{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i,$$

the sample mean of Y. Then again, we could use an even simpler estimator for μ_Y: the very first observation in the sample, Y_1. Is Y_1 a good estimator? For now, assume that

$$Y \sim \chi^2_{12}$$

which is not too unreasonable as hourly income is non-negative and we expect many hourly earnings to be in a range of 5 to 15. Moreover, it is common for income distributions to be skewed to the right — a property of the χ^2_{12} distribution.

```r
# plot the chi_12^2 distribution
curve(dchisq(x, df=12),
      from = 0,
      to = 40,
      ylab = "density",
      xlab = "hourly earnings in Euro"
      )
```



We now draw a sample of $n = 100$ observations and take the first observation Y_1 as an estimate for μ_Y

```r
# set seed for reproducibility
set.seed(1)

# sample from the chi_12^2 distribution, keep only the first observation
rchisq(n = 100, df = 12)[1]
```

```
## [1] 8.257893
```

The estimate 8.26 is not too far away from $\mu_Y = 12$ but it is somewhat intuitive that we could do better: the estimator Y_1 discards a lot of information and its variance is the population variance:

$$\text{Var}(Y_1) = \text{Var}(Y) = 2 \cdot 12 = 24$$

This brings us to the following question: What is a *good* estimator of an unknown parameter in the first place? This question is tackled in Key Concepts 3.2 and 3.3

Key Concept 3.2
Bias, Consistency and Efficiency

Desirable characteristics of an estimator are unbiasedness, consistency and efficiency.

Unbiasedness:
If the mean of the sampling distribution of some estimator $\hat{\mu}_Y$ for the population mean μ_Y equals μ_Y

$$E(\hat{\mu}_Y) = \mu_Y$$

we say that the estimator is unbiased for μ_Y. The *bias* of $\hat{\mu}_Y$ is 0:

$$E(\hat{\mu}_Y) - \mu_Y = 0$$

Consistency:
We want the uncertainty of the estimator μ_Y to decrease as the number of observations in the sample grows. More precisely, we want the probability that the estimate $\hat{\mu}_Y$ falls within a small interval of the true value μ_Y to get increasingly closer to 1 as n grows. We write this as

$$\hat{\mu}_Y \xrightarrow{p} \mu_Y.$$

Variance and efficiency:
We want the estimator to be efficient. Suppose we have two estimators, $\hat{\mu}_Y$ and $\widetilde{\mu}_Y$ and for some given sample size n it holds that

$$E(\hat{\mu}_Y) = E(\widetilde{\mu}_Y) = \mu_Y$$

but

$$\text{Var}(\hat{\mu}_Y) < \text{Var}(\widetilde{\mu}_Y).$$

We then would prefer to use $\hat{\mu}_Y$ as it has a lower variance than $\widetilde{\mu}_Y$, meaning that $\hat{\mu}_Y$ is more *efficient* in using the information provided by the observations in the sample.

3.2 Properties of the Sample Mean

To examine properties of the sample mean as an estimator for the corresponding population mean, consider the following Rexample.

We generate a population `pop` which consists observations Y_i , $i = 1, \dots, 10000$ that origin from a normal distribution with mean $\mu = 10$ and variance $\sigma^2 = 1$. To investigate how the estimator $\hat{\mu} = \bar{Y}$ behaves we can draw random samples from this population and calculate $\bar{Y}$ for each of them. This is easily done by making use of the function `replicate()`. The argument `expr` is evaluated `n` times. In this case we draw samples of sizes $n = 5$ and $n = 25$, compute the sample means and repeat this exactly $n = 25000$ times.

For comparison purposes we store results for the estimator Y_1, the first observation in a sample for a sample of size 5, separately.

```r
# generate a fictive population
pop <- rnorm(10000, 10, 1)
```

```r
# sample form pop and estimate the mean
est1 <- replicate(expr = mean(sample(x = pop, size = 5)), n = 25000)

est2 <- replicate(expr = mean(sample(x = pop, size = 25)), n = 25000)

fo <- replicate(expr = sample(x = pop, size = 5)[1], n = 25000)
```

Check that `est1` and `est2` are vectors of length 25000:

```r
# check if object type is vector
is.vector(est1)
```

```
## [1] TRUE
```

```r
is.vector(est2)
```

```
## [1] TRUE
```

```r
# check length
length(est1)
```

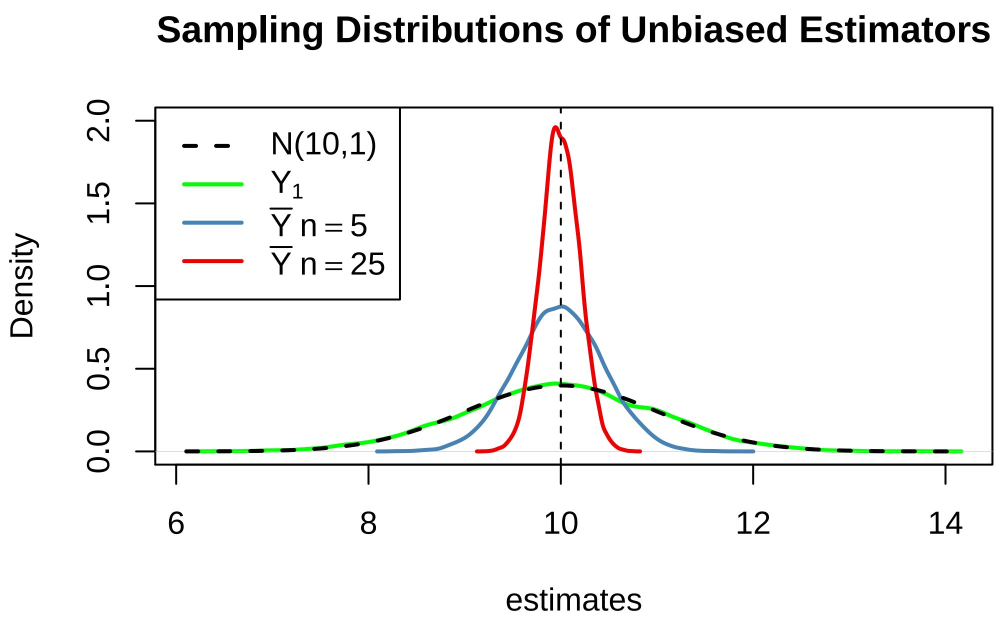
```
## [1] 25000
```

```r
length(est2)
```

```
## [1] 25000
```

The code chunk below produces a plot of the sampling distributions of the estimators $\bar{Y}$ and Y_1 on the basis of the 25000 samples in each case. We also plot the density function of the $N(10, 1)$ distribution.

```r
# plot density estimate Y_1
plot(density(fo),
      col = 'green',
      lwd = 2,
      ylim = c(0, 2),
      xlab = 'estimates',
      main = 'Sampling Distributions of Unbiased Estimators'
      )

# add density estimate for the distribution of the sample mean with n=5 to the plot
lines(density(est1),
     col = 'steelblue',
     lwd = 2,
     bty = 'l'
     )

# add density estimate for the distribution of the sample mean with n=25 to the plot
lines(density(est2),
      col = 'red2',
      lwd = 2
      )

# add a vertical line marking the true parameter
abline(v = 10, lty = 2)

# add N(10,1) density to the plot
curve(dnorm(x, mean = 10),
     lwd = 2,
     lty = 2,
     add = T
     )

# add a legend
legend("topleft",
       legend = c("N(10,1)",
                  expression(Y[1]),
                  expression(bar(Y) ~ n == 5),
                  expression(bar(Y) ~ n == 25)
                  ),
       lty = c(2, 1, 1, 1),
       col = c('black','green', 'steelblue', 'red2'),
       lwd = 2
       )
```



At first, notice that *all* sampling distributions (represented by the solid lines) are centered around $\mu = 10$. This is evidence for the *unbiasedness* of Y_1, $\overline{Y}_{n=5}$ and $\overline{Y}_{n=25}$. Of course, the theoretical density the $N(10, 1)$ distribution is centered at 10, too.

Next, have a look at the spread of the sampling distributions. Several things are remarkable:

- the sampling distribution of Y_1 (green curve) tracks the density of the $N(10, 1)$ distribution (black dashed line) pretty closely In fact, the sampling distribution of Y_1 is the $N(10, 1)$ distribution. This is less surprising if You keep in mind that the Y_1 estimator does nothing but reporting an observation that is randomly selected from a population with $N(10, 1)$ distribution. Hence, $Y_1 \sim N(10, 1)$. Note that this result does not depend on the sample size n: the sampling distribution of Y_1 *is always* the population distribution, no matter how large the sample is.

- Both sampling distributions of $\overline{Y}$ show less dispersion than the sampling distribution of Y_1. This means that $\overline{Y}$ has a lower variance than Y_1. In view of Key Concepts 3.2 and 3.3, we find that $\overline{Y}$ is a more efficient estimator than Y_1. In fact, one can show that this holds for all $n > 1$.

- $\overline{Y}$ shows a behavior that is termed consistency (see Key Concept 3.2). Notice that the blue and the red density curves are much more concentrated around $\mu = 10$ then the green one. As the number of observations is increased from 1 to 5, the sampling distribution tightens around the true parameter. Increasing the sample size to 25 this effect becomes more dominant. This implies that the probability of obtaining estimates that are close to the true value increases with n.

A more precise way to express consitency of an estimator $\hat{\mu}$ for a parameter μ is

$$P(|\hat{\mu} - \mu| < \epsilon) \xrightarrow[n\rightarrow\infty]{p} 1 \quad \text{for any} \quad \epsilon > 0.$$

This expression says that the probability of observing a deviation from the true value μ that is smaller than some arbitrary $\epsilon > 0$ converges to 1 as n grows. Note that consistency does not require unbiasedness.

We encourage you to go ahead and modify the code. Try out different values for the sample size and see how the sampling distribution of $\overline{Y}$ changes!

$\overline{Y}$ is the Least Squares Estimator of μ_Y

Assume you have some observations $Y_1, \dots, Y_n$ on $Y \sim N(10, 1)$ (which is unknown) and would like to find an estimator m that predicts the observations as good as possible. Good means to choose m such that the total deviation between the predicted value and the observed values is small. Mathematically this means we want to find an m that minimizes

$$\sum_{i=1}^{n} (Y_i - m)^2. \tag{3.1}$$

Think of $Y_i - m$ as the committed mistake when predicting Y_i by m. We could just as well minimize the sum of absolute deviations from m but minimizing the sum of squared deviations is mathematically more convenient (and may lead to a different result). That is why the estimator we are looking for is called the *least squares estimator*. As It turns out $m = \overline{Y}$, the estimator of $\mu_Y = 10$ is this wanted estimator.

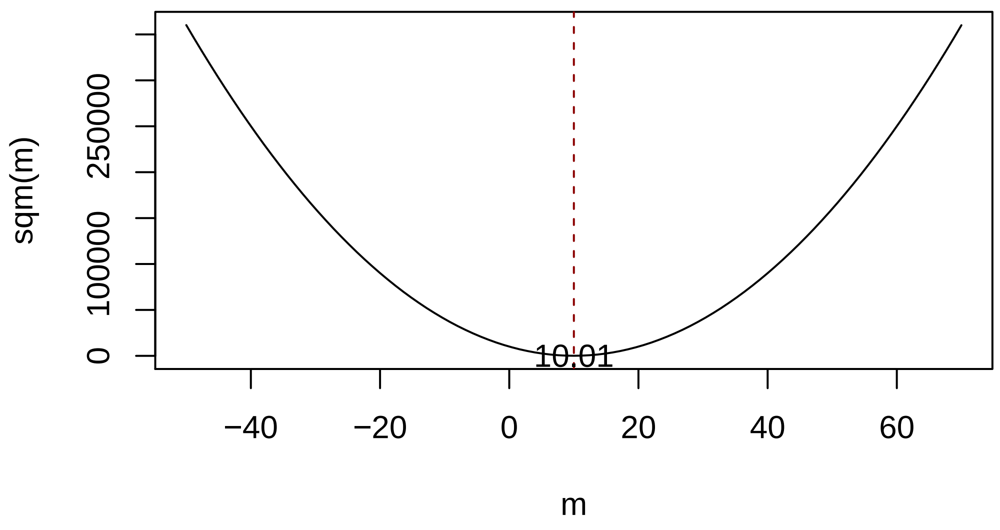
We can show this by generating a random sample of fair size and plotting (3.1) as a function of m.

```r
# define the function and vectorize it
sqm <- function(m) {
 sum((y-m)^2)
}
sqm <- Vectorize(sqm)

# draw random sample and compute the mean
y <- rnorm(100, 10, 1)
mean(y)
```

```
## [1] 10.00543
```

```r
# plot the objective function
curve(sqm(x),
      from = -50,
      to = 70,
      xlab = "m",
      ylab = "sqm(m)"
      )
```

```r
# add vertical line at mean(y)
abline(v = mean(y),
       lty = 2,
       col = "darkred"
       )

# add annotation at mean(y)
text(x = mean(y),
     y = 0,
     labels = paste(round(mean(y), 2))
     )
```



Notice that (3.1) is a quadratic function so there is only one minimum. The plot shows that this minimum lies exactly at the sample mean of the sample data.

Some R functions can only interact with functions that take a vector as an input and evaluate the function body on every values of the vector, for example curve(). We call such functions vectorized functions and it is often a good idea to write vectorized functions yourself although this is cumbersome in some cases. Having a vectorized function in R is never a drawback since these functions work on both single values and vectors.
Let us look at the function sqm() which is nonvectorized

```
sqm <- function(m) {
    sum((y-m)^2) #body of the function
}
```

Providing e.g. c(1,2,3) as the argument m would cause an error since then the operation y-m is invalid: the vecors y and m are of incompatible dimensions. This is why we cannot use sqm() in conjunction with curve().
Here comes Vectorize() into play. It generates a vectorized version of a non-vectorized function.

Why Random Sampling is Important

So far, we assumed (sometimes implicitly) that observed data $Y_1, \dots, Y_n$ are the result of a sampling process that satisfies the assumption of i.i.d. random sampling. It is very important that this assumption is fulfilled when estimating a population mean using $\overline{Y}$. If this is not the case, estimates are biased.

Let us fall back to `pop`, the fictive population of 10000 observations and compute the population mean $\mu_{\texttt{pop}}$:

```r
# compute the population mean of pop
mean(pop)
```

```
## [1] 9.992604
```

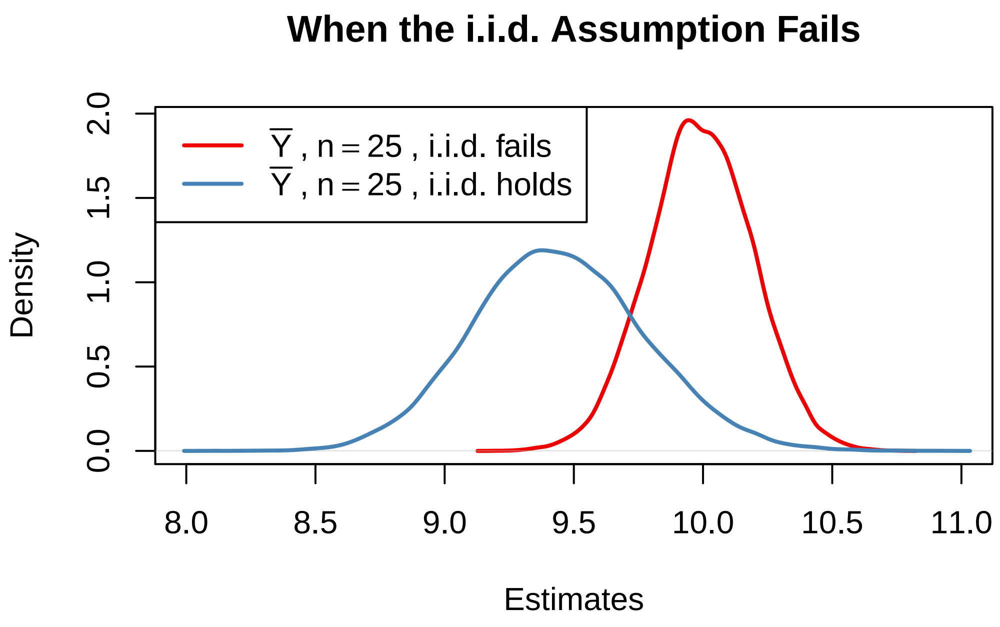
Next we sample 10 observations from `pop` with `sample()` and estimate $\mu_{\texttt{pop}}$ using $\overline{Y}$ repeatedly. However this time we use a sampling scheme that deviates from simple random sampling: instead of ensuring that each member of the population has the same chance to end up in a sample, we assign a higher probability of being sampled to the 2500 smallest observations of the population by setting the argument `prop` to a suitable vector of probability weights:

```r
# simulate outcome for the sample mean when the i.i.d. assumption fails
est3 <-  replicate(n = 25000,
                   expr = mean(sample(x = sort(pop),
                                      size = 10,
                                      prob = c(rep(4, 2500), rep(1, 7500))
                                      )
                               )
                   )

# compute the sample mean of the outcomes
mean(est3)
```

```
## [1] 9.443454
```

Next we plot the sampling distribution of $\overline{Y}$ for this non-i.i.d. case and compare it to the sampling distribution when the i.i.d. assumption holds.

```r
# sampling distribution of sample mean, i.i.d. holds, n=25
plot(density(est2),
      col = 'red2',
      lwd = 2,
      xlim = c(8, 11),
      xlab = 'Estimates',
      main = 'When the i.i.d. Assumption Fails'
     )

# sampling distribution of sample mean, i.i.d. fails, n=25
lines(density(est3),
      col = 'steelblue',
      lwd = 2
      )

# add a legend
legend("topleft",
       legend = c(expression(bar(Y) ~ "," ~ n == 25 ~ ", i.i.d. fails"),
                  expression(bar(Y) ~ "," ~ n == 25 ~ ", i.i.d. holds")
                  ),
       lty = c(1, 1),
       col = c('red2', 'steelblue'),
       lwd = 2
       )
```



We find that in this case failure of the i.i.d. assumption implies that, on average, we *underestimate* μ_Y using $\overline{Y}$: the corresponding distribution of $\overline{Y}$ is shifted to the left. In other words, $\overline{Y}$ is a *biased* estimator for μ_Y if the i.i.d. assumption does not hold.

3.3 Hypothesis Tests Concerning the Population Mean

In this section we briefly review concepts in hypothesis testing and discuss how to conduct hypothesis tests in `R`. We focus on drawing inferences about an unknown population mean.

About Hypotheses and Hypothesis Testing

In a significance test we want to exploit the information contained in a random sample as evidence in favor or against a hypothesis. Essentially, hypotheses are simple questions that can be answered by 'yes' or 'no'. When conducting a hypothesis test we always deal with two different hypotheses:

- The **null hypothesis**, denoted H_0 is the hypothesis we are interested in testing.
- The **alternative hypothesis**, denoted H_1, is the hypothesis that holds if the null hypothesis is false.

The null hypothesis that the population mean of Y equals the value $\mu_{Y,0}$ is written down as

$$H_0 : E(Y) = \mu_{Y,0}.$$

The alternative hypothesis states what holds if the null hypothesis is false. Often the alternative hypothesis chosen is the most general one,

$$H_1 : E(Y) \neq \mu_{Y,0},$$

meaning that $E(Y)$ may be anything but the value under the null hypothesis. This is called a *two-sided* alternative.

For the sake of brevity, we will only consider the case of a two-sided alternative in the subsequent sections of this chapter.

The p-Value

Assume that the null hypothesis is *true*. The p-value is the probability of drawing data and observing a corresponding test statistic that is at least as adverse to what is stated under the null hypothesis as the test statistic actually computed using the sample data.

In the context of the population mean and the sample mean, this definition can be stated mathematically in the following way:

$$p\text{-value} = P_{H_0}\left[|\overline{Y} - \mu_{Y,0}| > |\overline{Y}^{act} - \mu_{Y,0}|\right] \tag{3.2}$$

In (3.2), $\overline{Y}^{act}$ is the actually computed mean of the random sample. Visualized, the p-value is the area in the part of tails of the distribution of $\overline{Y}$ that lies beyond

$$\mu_{Y,0} \pm |\overline{Y}^{act} - \mu_{Y,0}|.$$

Consequently, in order to compute the p-value as in (3.2), knowledge about the sampling distribution of $\overline{Y}$ when the null hypothesis is true is required. However in most cases the sampling distribution of $\overline{Y}$ is unknown. Fortunately, due to the large-sample normal approximation we know that under the null hypothesis

$$\overline{Y} \sim N(\mu_{Y,0}, \sigma^2_{\overline{Y}}) \quad , \quad \sigma^2_{\overline{Y}} = \frac{\sigma^2_Y}{n}$$

and thus

$$\frac{\overline{Y} - \mu_{Y,0}}{\sigma_Y/\sqrt{n}} \sim N(0,1).$$

So in large samples, the p-value can be computed *without* knowledge of the exact sampling distribution of $\overline{Y}$.

Calculating the p-Value When σ_Y Is Known

For now, let us assume that $\sigma_{\overline{Y}}$ is known. Then we can rewrite (3.2) as

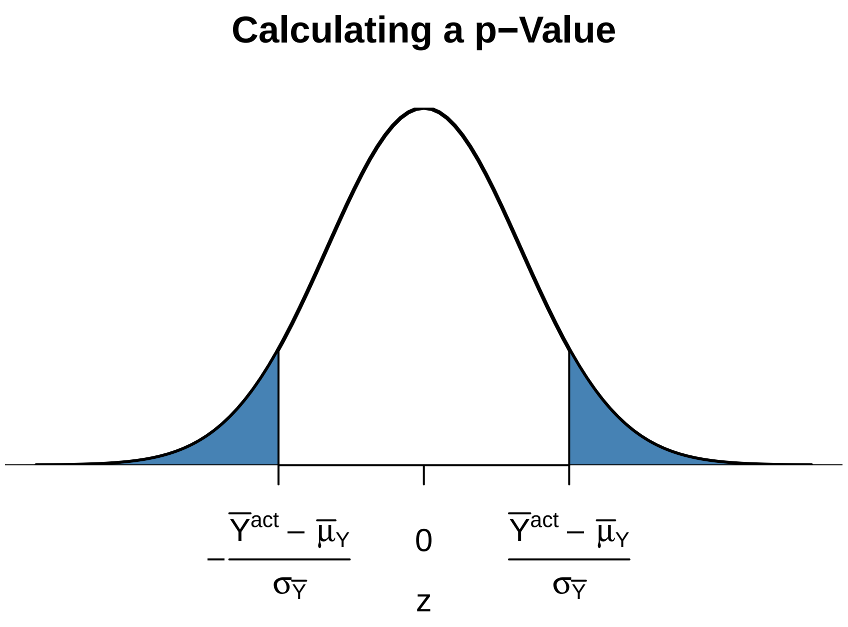
$$\begin{aligned} p\text{-value} &= P_{H_0}\left[\left|\frac{\overline{Y} - \mu_{Y,0}}{\sigma_{\overline{Y}}}\right| > \left|\frac{\overline{Y}^{act} - \mu_{Y,0}}{\sigma_{\overline{Y}}}\right|\right] && (3.3) \\ &= 2 \cdot \Phi\left[-\left|\frac{\overline{Y}^{act} - \mu_{Y,0}}{\sigma_{\overline{Y}}}\right|\right]. && (3.4) \end{aligned}$$

The p-value can be seen as the area in the tails of the $N(0,1)$ distribution that lies beyond

$$\pm \left| \frac{\overline{Y}^{act} - \mu_{Y,0}}{\sigma_{\overline{Y}}} \right| \tag{3.5}$$

We now use R to visualize what is stated in (3.4) and (3.5). The next code chunk replicates Figure 3.1 of the book.

```
# plot the standard normal density on the interval [-4,4]
curve(dnorm(x),
      xlim = c(-4, 4),
      main = 'Calculating a p-Value',
      yaxs = 'i',
      xlab = 'z',
      ylab = '',
      lwd = 2,
      axes = 'F'
      )

# add x-axis
axis(1,
     at = c(-1.5, 0, 1.5),
     padj = 0.75,
     labels = c(expression(-frac(bar(Y)^"act"~-~bar(mu)[Y,0], sigma[bar(Y)])),
                0,
                expression(frac(bar(Y)^"act"~-~bar(mu)[Y,0], sigma[bar(Y)])))
     )

# shade p-value/2 region in left tail
polygon(x = c(-6, seq(-6, -1.5, 0.01), -1.5),
        y = c(0, dnorm(seq(-6, -1.5, 0.01)),0),
        col = 'steelblue'
        )

# shade p-value/2 region in right tail
polygon(x = c(1.5, seq(1.5, 6, 0.01), 6),
        y = c(0, dnorm(seq(1.5, 6, 0.01)), 0),
        col = 'steelblue'
        )
```



Sample Variance, Sample Standard Deviation and Standard Error

If σ^2_Y is unknown, it must be estimated. This can be done efficiently using the sample variance

$$s_y^2 = \frac{1}{n-1}\sum_{i=1}^n (Y_i - \overline{Y})^2. \tag{3.6}$$

Furthermore

$$s_y = \sqrt{\frac{1}{n-1}\sum_{i=1}^n (Y_i - \overline{Y})^2} \tag{3.7}$$

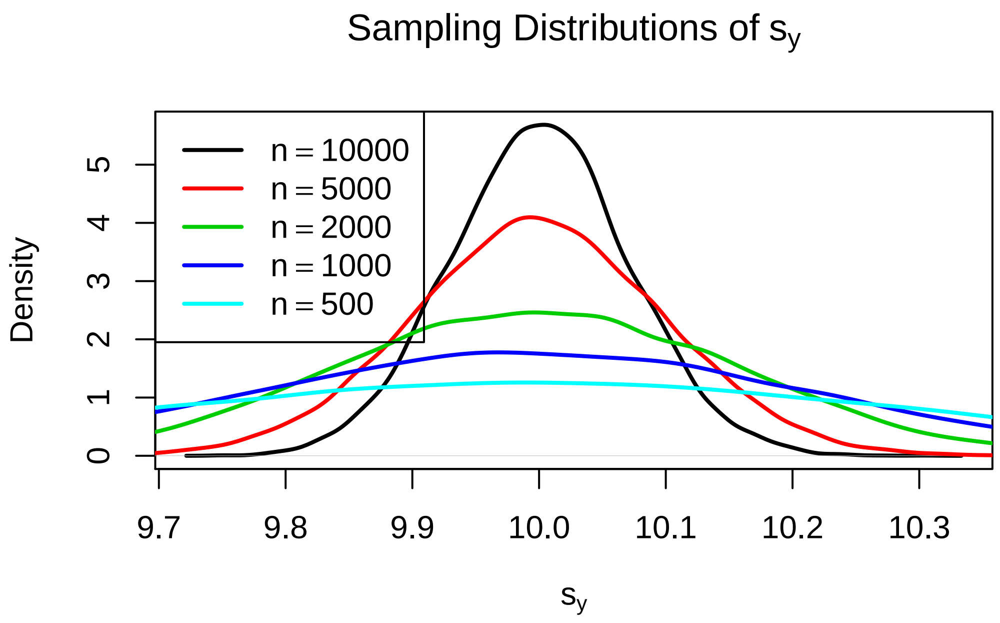
is a suitable estimator for the standard deviation of Y. In R, s_y is implemented in the function `sd()`, use `?sd`.

Using `R` we can get a notion that s_y is a consistent estimator for σ_Y, that is

$$s_Y \xrightarrow{p} \sigma_Y.$$

The idea here is to generate a large number of samples $Y_1, \dots, Y_n$ where, $Y \sim N(10, 10)$ say, estimate σ_Y using s_y and investigate how the distribution of s_Y changes as n gets larger.

```r
# vector of sample sizes
n <- c(10000, 5000, 2000, 1000, 500)

# sample observations, estimate using sd() and plot estimated distributions
s2_y <- replicate(n = 10000, expr = sd(rnorm(n[1], 10, 10)))
plot(density(s2_y),
     main = expression('Sampling Distributions of' ~ s[y]),
     xlab = expression(s[y]),
     lwd = 2
     )

for (i in 2:length(n)) {
  s2_y <- replicate(n = 10000, expr = sd(rnorm(n[i], 10, 10)))
  lines(density(s2_y),
        col = i,
        lwd = 2
        )
}

# add a legend
legend("topleft",
       legend = c(expression(n == 10000),
                  expression(n == 5000),
                  expression(n == 2000),
                  expression(n == 1000),
                  expression(n == 500)
       ),
       col = 1:5,
       lwd = 2
)
```



The plot shows that the distribution of s_Y tightens around the true value $\sigma_Y = 10$ as n increases.

The function that estimates the standard deviation of an estimator is called the *standard error of the estimator*. Key Concept 3.4 summarizes the terminology in the context of the sample mean.

Key Concept 3.4

The Standard Error of $\overline{Y}$

Take an i.i.d. sample $Y_1, \dots, Y_n$. The mean of Y can be consistently estimated using $\overline{Y}$, the sample mean of the Y_i. Since $\overline{Y}$ is a random variable, it has a sampling distribution with variance $\frac{\sigma_Y^2}{n}$.

The standard error of $\overline{Y}$, denoted $SE(\overline{Y})$ is an estimator of the standard deviation of $\overline{Y}$:

$$SE(\overline{Y}) = \hat{\sigma}_{\overline{Y}} = \frac{s_Y}{\sqrt{n}}$$

The caret (^) over σ indicates that $\hat{\sigma}_{\overline{Y}}$ is an estimator for $\sigma_{\overline{Y}}$.

As an example to underpin Key Concept 3.4, consider a sample of $n = 100$ i.i.d. observations of the Bernoulli distributed variable Y with success probability $p = 0.1$ and thus $E(Y) = p = 0.1$ and $\text{Var}(Y) = p(1-p)$. $E(Y)$ can be estimated by $\overline{Y}$ which then has variance

$$\sigma^2_{\overline{Y}} = p(1-p)/n = 0.0009$$

and standard deviation

$$\sigma_{\overline{Y}} = \sqrt{p(1-p)/n} = 0.03.$$

In this case the standard error of $\overline{Y}$ is given by

$$SE(\overline{Y}) = \sqrt{\overline{Y}(1-\overline{Y})/n}.$$

Let us verify whether $\overline{Y}$ and $SE(\overline{Y})$ estimate the respective true values, on average.

```r
# draw 10000 samples of size 100 and estimate the mean of Y and
# estimate the standard error of the sample mean

mean_estimates <- numeric(10000)
se_estimates <- numeric(10000)

for (i in 1:10000) {
  s <- sample(0:1,
              size = 100,
              prob = c(0.9, 0.1),
              replace = T
              )
  mean_estimates[i] <- mean(s)
  se_estimates[i] <- sqrt(mean(s)*(1-mean(s))/100)
}

mean(mean_estimates)
```

```
## [1] 0.099693
```

```r
mean(se_estimates)
```

```
## [1] 0.02953467
```

Both estimators seem to be unbiased for the true parameters.

Calculating the p-value When the Standard Deviation is Unknown

When σ_Y is unknown, the p-value for a hypothesis test concerning μ_Y using $\overline{Y}$ can be computed by replacing $\sigma_{\overline{Y}}$ in (3.4) by the standard error $SE(\overline{Y}) = \hat{\sigma}_Y$. Then,

$$p\text{-value} = 2 \cdot \Phi\left(- \left| \frac{\overline{Y}^{act} - \mu_{Y,0}}{SE(\overline{Y})} \right| \right).$$

This is easily done in R:

```
# sample and estimate, compute standard error and make a hypothesis
samplemean_act <- mean(
  sample(0:1,
         prob = c(0.9, 0.1),
         replace = T,
         size = 100
         )
  )

SE_samplemean <- sqrt(samplemean_act * (1-samplemean_act)/100)

mean_h0 <- 0.1 #true null hypothesis

# compute the p-value
pvalue <- 2 * pnorm(-abs(samplemean_act - mean_h0)/SE_samplemean)
pvalue
```

```
## [1] 0.5382527
```

The t-statistic

In hypothesis testing, the standardized sample average

$$t = \frac{\overline{Y} - \mu_{Y,0}}{SE(\overline{Y})} \tag{3.8}$$

is called t-statistic. This t-statistic plays an important role in testing hypotheses about μ_Y. It is a prominent example of a test statistic.

Implicitly, we already have computed a t-statistic for $\overline{Y}$ in the previous code chunk.

```
# compute a t-statistic for the sample mean
tstatistic <- (samplemean_act - mean_h0) / SE_samplemean
tstatistic
```

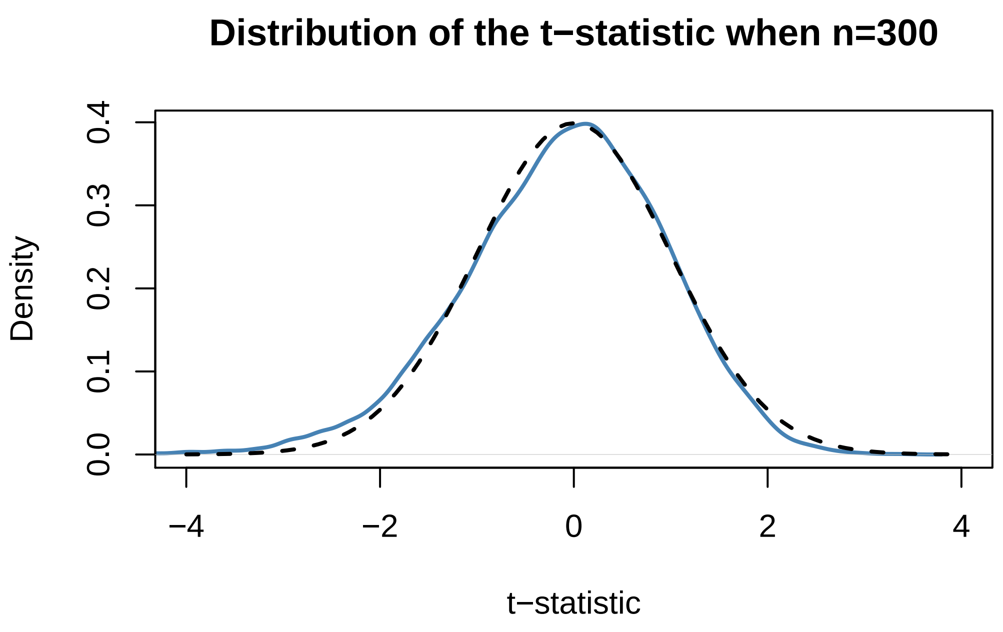
```
## [1] 0.6154575
```

Using Rwe can show that if $\mu_{Y,0}$ equals the true value, that is the null hypothesis is true, (3.8) is approximately distributed $N(0, 1)$ when n is large.

```
# prepare empty vector for t-statistics
tstatistics <- numeric(10000)

# set sample size
n <- 300
```

```r
# simulate 10000 t-statistics
for (i in 1:10000) {
  s <- sample(0:1,
              size = n,
              prob = c(0.9, 0.1),
              replace = T
              )
  tstatistics[i] <- (mean(s)-0.1)/(sqrt(mean(s)*(1-mean(s))/n))
}
```

```r
# plot density and compare to N(0,1) density
plot(density(tstatistics),
     xlab = 't-statistic',
     main = 'Distribution of the t-statistic when n=300',
     lwd = 2,
     xlim = c(-4, 4),
     col = 'steelblue'
     )

# N(0,1) density (dashed)
curve(dnorm(x),
      add = T,
      lty = 2,
      lwd = 2
      )
```



Judging from the plot, the normal approximation works reasonably well for the chosen sample size. This normal approximation has already been used in the definition of the p-value, see (3.8).

Hypothesis Testing with a Prespecified Significance Level

Key Concept 3.5

The Terminology of Hypothesis Testing

In hypothesis testing, two types of mistakes are possible:

1. The null hypothesis *is* rejected although it is true (α-error / type-I-error)
2. The null hypothesis *is not* rejected although it is false (β-error / type-II-error)

The **significance level** of the test is the probability to commit a type-I-error we are willing to accept in advance. E.g. using a prespecified significance level of 0.05, we reject the null hypothesis if and only if the p-value is less than 0.05. The significance level is chosen before the test is conducted.

An equivalent procedure is to reject the null hypothesis if the test statistic observed is, in absolute value terms, larger than the **critical value** of the test statistic. The critical value is determined by the significance level chosen and defines two disjoint sets of values which are called **acceptance region** and **rejection region**. The acceptance region contains all values of the test statistic for which the test does not reject while the rejection region contains all the values for which the test does reject.

The ***p*-value** is the probability that, in repeated sampling under the same conditions, meaning i.i.d. sampling, the same null hypothesis and the same sample size, a test statistic is observed that provides just as much evidence against the null hypothesis as the test statistic actually observed.

The actual probability that the test rejects the true null hypothesis is called the **size of the test**. In an ideal setting, the size does not exceed the significance level.

The probability that the test correctly rejects a false null hypothesis is called **power**.

Reconsider `pvalue` computed further above:

```
# check whether p-value < 0.05
pvalue < 0.05
```

```
## [1] FALSE
```

The condition is not fulfilled so we do not reject the null hypothesis (remember that the null hypothesis is true in this example).

When working with a t-statistic instead, it is equivalent to apply the following rule:

$$\text{Reject } H_0 \text{ if } |t^{act}| > 1.96$$

We reject the null hypothesis at the significance level of 5% if the computed t-statistic lies beyond the critical value of 1.96 in absolute value terms. 1.96 is the 0.05-quantile of the standard normal distribution.

```
# check the critical value
qnorm(p = 0.05)
```

```
## [1] -1.644854
```

```
# check whether the null is rejected using the t-statistic computed further above
abs(tstatistic) > 1.96
```

```
## [1] FALSE
```

Just like using the p-value, we cannot reject the null hypothesis using the corresponding t-statistic. Key Concept 3.6 summarizes the procedure of performing a two-sided hypothesis about the population mean $E(Y)$.

Key Concept 3.6

Testing the Hypothesis $E(Y) = \mu_{Y,0}$ Against the Alternative $E(Y) \neq \mu_{Y,0}$

1. Estimate μ_Y using $\overline{Y}$ and compute the standard error of $\overline{Y}$, $SE(\overline{Y})$.
2. Compute the t-statistic.
3. Compute the p-value and reject the null hypothesis at the 5% level of significance if the p-value is smaller than 0.05 or equivalently, if

$$\left|t^{act}\right| > 1.96.$$

One-sided Alternatives

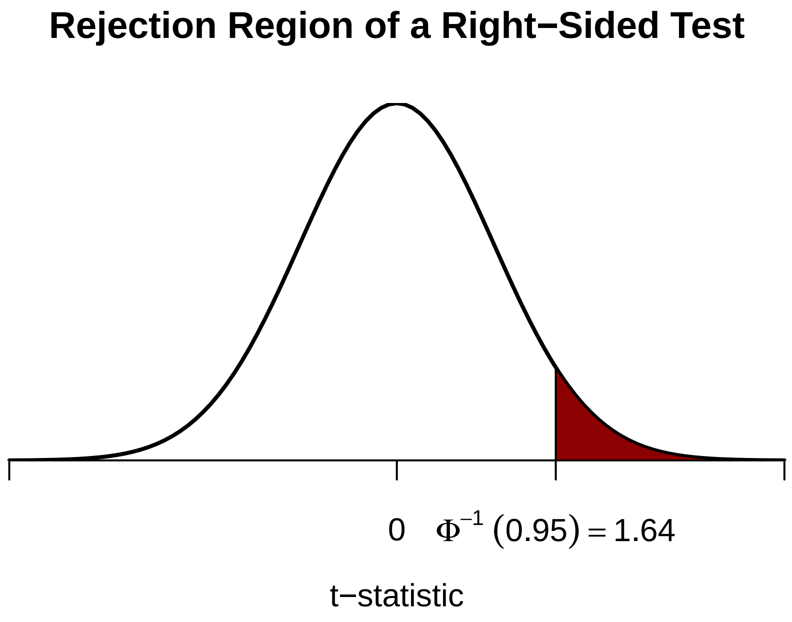
Sometimes we are interested in finding evidence that the mean is bigger or smaller than some value hypothesized under the null. One can come up with many examples here but, to stick to the book, take the presumed wage gap between well and less educated working individuals. Since we hope that this differential exists, a relevant alternative (to the null hypothesis that there is no wage differential) is that good educated individuals earn more, i.e. that the average hourly wage for this group, μ_Y is *bigger* than $\mu_{Y,0}$ the know average wage of less educated workers.

This is an example of a *right-sided test* and the hypotheses pair is chosen to be

$$H_0 : \mu_Y = \mu_{Y,0} \quad \text{vs} \quad H_1 : \mu_Y > \mu_{Y,0}.$$

We reject the null hypothesis if the computed test-statistic is larger than the critical value 1.64, the 0.95-quantile of the $N(0,1)$ distribution. This ensures that $1 - 0.95 = 5\%$ probability mass remains in the area to the right of the critical value. Similar as before we can visualize this in `R` using the function `polygon()`.

```r
# plot the standard normal density on the domain [-4,4]
curve(dnorm(x),
      xlim = c(-4, 4),
      main = 'Rejection Region of a Right-Sided Test',
      yaxs = 'i',
      xlab = 't-statistic',
      ylab = '',
      lwd = 2,
      axes = 'F'
)

# add x-axis
axis(1,
     at = c(-4, 0, 1.64, 4),
     padj = 0.5,
     labels = c('', 0, expression(Phi^-1~(.95)==1.64), '')
)

# shade rejection region in the left tail
polygon(x = c(1.64, seq(1.64, 4, 0.01), 4),
        y = c(0, dnorm(seq(1.64, 4, 0.01)), 0),
        col = 'darkred'
)
```



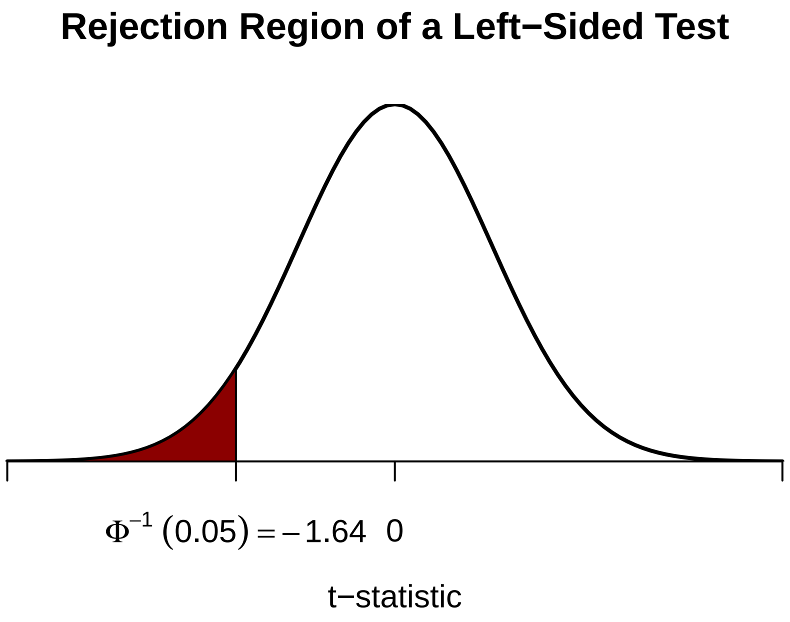
Analogously for the left-sided test we have

$$H_0 : \mu_Y = \mu_{Y,0} \quad \text{vs.} \quad H_1 : \mu_Y < \mu_{Y,0}.$$

The null is rejected if the observed test statistic falls short of the critical value which, for a test at the 0.05 level of significance, is given by -1.64, the 0.05-quantile of the $N(0,1)$ distribution. 5% probability mass lies to the left of the critical value.

It is straightforward to adapt the code chunk above to the case of a left-sided test. We only have to adjust the color shading and the tick marks.

```
# plot the standard normal density on the domain [-4,4]
curve(dnorm(x),
      xlim = c(-4, 4),
      main = 'Rejection Region of a Left-Sided Test',
      yaxs = 'i',
      xlab = 't-statistic',
      ylab = '',
      lwd = 2,
      axes = 'F'
)

# add x-axis
axis(1,
     at = c(-4, 0, -1.64, 4),
     padj = 0.5,
     labels = c('', 0, expression(Phi^-1~(.05)==-1.64), '')
)

# shade rejection region in right tail
polygon(x = c(-4, seq(-4, -1.64, 0.01), -1.64),
        y = c(0, dnorm(seq(-4, -1.64, 0.01)), 0),
        col = 'darkred'
)
```



3.4 Confidence Intervals for the Population Mean

As stressed before, we will never estimate the *exact* value of the population mean of Y using a random sample. However, we can compute confidence intervals for the population mean. In general, a confidence interval for a unknown parameter is a set of values that contains the true parameter with a prespecified probability, the *confidence level*. Confidence intervals are computed using the information available in the sample. Since this information is the result of a random process, confidence intervals are random variables themselves.

Key Concept 3.7 shows how to compute confidence intervals for the unknown population mean $E(Y)$.

Key Concept 3.7

Confidence Intervals for the Population Mean

A 95% confidence interval for μ_Y is a `random variable` that contains the true μ_Y in 95% of all possible random samples. When n is large we can use the normal approximation. Then, 99%, 95%, 90% confidence intervals are

$$\begin{align}
&99\% \text{ confidence interval for } \mu_Y = \left\{ \overline{Y} \pm 2.58 \times SE(\overline{Y}) \right\}, \tag{3.9} \\
&95\% \text{ confidence interval for } \mu_Y = \left\{ \overline{Y} \pm 1.96 \times SE(\overline{Y}) \right\}, \tag{3.10} \\
&90\% \text{ confidence interval for } \mu_Y = \left\{ \overline{Y} \pm 1.64 \times SE(\overline{Y}) \right\}. \tag{3.11}
\end{align}$$

These confidence intervals are sets of null hypotheses we cannot reject in a two-sided hypothesis test at the given level of confidence.

Now consider the following statements.

1. The interval
$$\left\{\overline{Y} \pm 1.96 \times SE(\overline{Y})\right\}$$
covers the true value of μ_Y with a probability of 95%.

2. We have computed $\overline{Y} = 5.1$ and $SE(\overline{Y} = 2.5$ so the interval
$$\{5.1 \pm 1.96 \times 2.5\} = [0.2, 10]$$
covers the true value of μ_Y with a probability of 95%.

While 1. is right (this is exactly in line with the definition above), 2. is completely wrong and none of your lecturers wants to read such a sentence in a term paper, written exam or similar, believe us. The difference is that, while 1. is the definition of a random variable, 2. is one possible *outcome* of this random variable so there is no meaning in making any probabilistic statement about it. Either the computed interval *does cover* μ_Y or it *does not*!

In `R`, testing of hypotheses about the mean of a population on the basis of a random sample is very easy due to functions like `t.test()` from the `stats` package. It produces an object of type `list`. Luckily, one of the most simple ways to use `t.test()` is when you want to obtain a 95% confidence interval for some population mean. We start by generating some random data and calling `t.test()` in conjunction with `ls()` to obtain a breakdown of the output components.

```
# set random seed
set.seed(1)

# generate some sample data
sampledata <- rnorm(100, 10, 10)

# checke type
typeof(t.test(sampledata))
```

```
## [1] "list"
```

```
# display list elements produced by t.test
ls(
  t.test(sampledata)
)
```

```
## [1] "alternative" "conf.int"    "data.name"   "estimate"    "method"
## [6] "null.value"  "p.value"     "parameter"   "statistic"
```

Though we find that many items are reported, at the moment we are only interested in computing a 95% confidence set for the mean.

```
t.test(sampledata)$"conf.int"
```

```
## [1]  9.306651 12.871096
## attr(,"conf.level")
## [1] 0.95
```

This tells us that the 95% confidence interval is

$$[9.31, 12.87]\,.$$

In this example, the computed interval obviously does cover the true μ_Y which we know to be 10.

Let us have a look at the whole standard output produced by `t.test()`.

```r
t.test(sampledata)
```

```
##
##  One Sample t-test
##
## data:  sampledata
## t = 12.346, df = 99, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##   9.306651 12.871096
## sample estimates:
## mean of x
##  11.08887
```

We see that `t.test()` does not only compute a 95% confidence interval but automatically conducts a two-sided significance test of the hypothesis $H_0 : \mu_Y = 0$ at the level of 5% and reports relevant parameters thereof: the alternative hypothesis, the estimated mean, the resulting t-statistic, the degrees of freedom of the underlying t distribution (`t.test()` does not perform the normal approximation) and the corresponding p-value. This is very convenient!

In this example, we come to the conclusion that the population mean *is not* significantly different from 0 (which is correct) at the level of 5%, since $\mu_Y = 0$ is element of the 95% confidence interval

$$0 \in [-0.27, 0.12] .$$

We come to an equivalent result when using the p-value rejection rule since

$$p = 0.456 > 0.05.$$

3.5 Comparing Means from Different Populations

Suppose you are interested in the means of two different populations, denote them μ_1 and μ_2. More specifically you are interested whether these population means are different from each other and plan using a hypothesis test to verify this on the basis of independent sample data from both populations. A suitable pair of hypotheses is

$$H_0 : \mu_1 - \mu_2 = d_0 \quad \text{vs.} \quad H_1 : \mu_1 - \mu_2 \neq d_0 \tag{3.12}$$

where d_0 denotes the hypothesized difference in means. The book teaches us that H_0 can be tested with the t-statistic

$$t = \frac{(\overline{Y}_1 - \overline{Y}_2) - d_0}{SE(\overline{Y}_1 - \overline{Y}_2)} \tag{3.13}$$

where

$$SE(\overline{Y}_1 - \overline{Y}_2) = \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}. \tag{3.14}$$

This is called a two sample t-test. For large n_1 and n_2, (3.13) is standard normal distributed under the null hypothesis. Analogously to the simple t-test we can compute confidence intervals for the true difference in population means:

$$(\overline{Y}_1 - \overline{Y}_2) \pm 1.96 \times SE(\overline{Y}_1 - \overline{Y}_2)$$

is a 95% confidence interval for d. In R, hypotheses as in (3.12) can be tested with `t.test()`, too. Note that `t.test()` chooses $d_0 = 0$ by default. This can be changed by setting the argument `mu` accordingly.

The subsequent code chunk demonstrates how to perform a two sample t-test in R using simulated data.

```
# set random seed
set.seed(1)

# draw data from two different populations with equal mean
sample_pop1 <- rnorm(100, 10, 10)
sample_pop2 <- rnorm(100, 10, 20)

# perform a two sample t-test
t.test(sample_pop1, sample_pop2)
```

```
##
##  Welch Two Sample t-test
##
## data:  sample_pop1 and sample_pop2
## t = 0.872, df = 140.52, p-value = 0.3847
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -2.338012  6.028083
## sample estimates:
## mean of x mean of y
## 11.088874  9.243838
```

We find that the two sample t-test does not reject the (true) null hypothesis that $d_0 = 0$.

3.6 An Application to the Gender Gap of Earnings

This section discusses how to reproduce the results presented in the box *The Gender Gap of Earnings of College Graduates in the United States* of the book.

In order to reproduce Table 3.1 of the book you need to download the replication data which are hosted by Pearson and can be downloaded here. Download the data for Chapter 3 as an excel spreadsheet (`cps_ch3.xlsx`). This data set contains data that range from 1992 to 2008 and earnings are reported in prices of 2008. There are several ways to import the `.xlsx`-files into R. Our suggestion is the function `read_excel()` from the `readxl` package. The package is not part of R's base version and has to be installed manually.

```
# load the 'readxl' package
library(readxl)
```

You are now ready to import the data set. Make sure you use the correct path to import the downloaded file! In our example, the file is saved in a sub folder (`data`) of the working directory. If you are not sure what your current working directory is, use `getwd()`, see also `?getwd()`. This will give you the path that points to the place Ris currently looking for files.

```
# import the data into R
cps <- read_excel(path = 'data/cps_ch3.xlsx')
```

Next, install and load the package `dyplr`. This package provides some handy functions that simplify data wrangling a lot. It makes use of the `%>%` operator.

```
# load the 'dplyr' package
library('dplyr')
```

First, get an overview over the data set. Next, use `%>%` and some functions from the `dplyr` package to group the observations by gender and year and compute descriptive statistics for both groups.

```
# Get an overview of the data structure
head(cps)
```

```
## # A tibble: 6 x 3
##   a_sex  year ahe08
##   <dbl> <dbl> <dbl>
## 1     1  1992  17.2
## 2     1  1992  15.3
## 3     1  1992  22.9
## 4     2  1992  13.3
## 5     1  1992  22.1
## 6     2  1992  12.2
```

```
# group data by gender and year and compute the mean, standard deviation
# and number of observations for each group
avgs <- cps %>%
        group_by(a_sex, year) %>%
        summarise(mean(ahe08),
                  sd(ahe08),
                  n()
                  )

# print results to the console
print(avgs)
```

```
## # A tibble: 10 x 5
## # Groups:   a_sex [?]
##    a_sex  year `mean(ahe08)` `sd(ahe08)` `n()`
##    <dbl> <dbl>         <dbl>       <dbl> <int>
##  1     1  1992          23.3       10.2   1594
##  2     1  1996          22.5       10.1   1379
##  3     1  2000          24.9       11.6   1303
##  4     1  2004          25.1       12.0   1894
##  5     1  2008          25.0       11.8   1838
##  6     2  1992          20.0        7.87  1368
##  7     2  1996          19.0        7.95  1230
##  8     2  2000          20.7        9.36  1181
##  9     2  2004          21.0        9.36  1735
## 10     2  2008          20.9        9.66  1871
```

With the pipe operator `%>%` we simply chain different `R` functions that produce compatible input and output. In the code above, we take the data set `cps` and use it as an input for the function `group_by()`. The output of `group_by` is subsequently used as an input for `summarise()` and so forth.

Now that we have computed the statistics of interest for both genders, we can investigate how the gap in earnings between both groups evolves over time.

```
# split the data set by gender
male <- avgs %>% filter(a_sex == 1)
```

```r
female <- avgs %>% filter(a_sex == 2)

# Rename columns of both splits
colnames(male)   <- c("Sex", "Year", "Y_bar_m", "s_m", "n_m")
colnames(female) <- c("Sex", "Year", "Y_bar_f", "s_f", "n_f")

# Estimate gender gaps, compute standard errors and confidence intervals for all dates
gap <- male$Y_bar_m - female$Y_bar_f

gap_se <- sqrt(male$s_m^2 / male$n_m + female$s_f^2 / female$n_f)

gap_ci_l <- gap - 1.96 * gap_se

gap_ci_u <- gap + 1.96 * gap_se

result <- cbind(male[,-1], female[,-(1:2)], gap, gap_se, gap_ci_l, gap_ci_u)

# print results to the console
print(result, digits = 3)
```

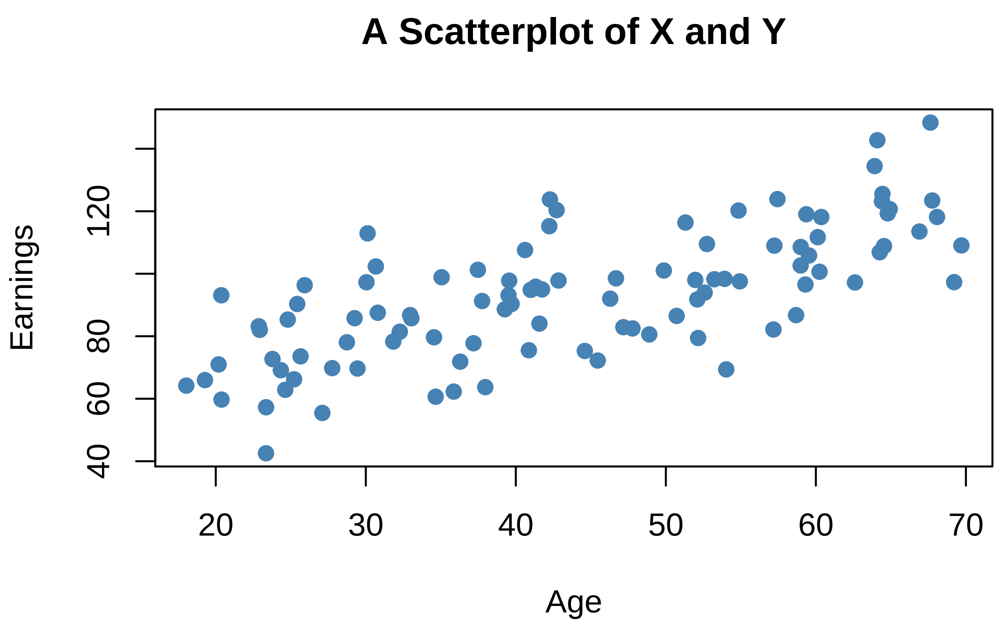
```
##   Year Y_bar_m  s_m  n_m Y_bar_f  s_f  n_f  gap gap_se gap_ci_l gap_ci_u
## 1 1992    23.3 10.2 1594    20.0 7.87 1368 3.23  0.332     2.58     3.88
## 2 1996    22.5 10.1 1379    19.0 7.95 1230 3.49  0.354     2.80     4.19
## 3 2000    24.9 11.6 1303    20.7 9.36 1181 4.14  0.421     3.32     4.97
## 4 2004    25.1 12.0 1894    21.0 9.36 1735 4.10  0.356     3.40     4.80
## 5 2008    25.0 11.8 1838    20.9 9.66 1871 4.10  0.354     3.41     4.80
```

We observe virtually the same results as the ones presented in the book. The computed statistics suggest that there *is* a gender gap in earnings. Note that we can reject the null hypothesis that the gap is zero for all periods. Further, estimates of the gap and limits of the 95% confidence intervals indicate that the gap has been quite stable over the recent past.

3.7 Scatterplots, Sample Covariance and Sample Correlation

A scatter plot represents two dimensional data, for example n observation on X_i and Y_i, by points in a Cartesian coordinate system. It is very easy to generate scatter plots using the `plot()` function in `R`. Let us generate some fictional data on age and earnings of workers and plot it.

```r
# set random seed
set.seed(123)

# generate data set
X <- runif(n = 100,
           min = 18,
           max = 70
           )
Y <- X + rnorm(n=100, 50, 15)

# plot observations
plot(X,
     Y,
     type = "p",
     main = "A Scatterplot of X and Y",
     xlab = "Age",
     ylab = "Earnings",
     col = "steelblue",
     pch = 19
     )
```



The plot shows positive correlation between age and earnings. This is in line with the assumption that older workers earn more than those who joined the working population recently.

Sample Covariance and Correlation

By now you should be familiar with the concepts of variance and covariance. If not, we recommend you to work your way through Chapter 2 of the book.

Just like the variance, covariance and correlation of two variables are properties that relate to the (unknown) joint probability distribution of these variables. We can estimate covariance and correlation by means of suitable estimators using a random sample (X_i, Y_i), $i = 1, \dots, n$.

The sample covariance

$$s_{XY} = \frac{1}{n-1} \sum_{i=1}^{n} (X_i - \overline{X})(Y_i - \overline{Y})$$

is an estimator for the population variance of X and Y whereas the sample correlation

$$r_{XY} = \frac{s_{XY}}{s_X s_Y}$$

can be used to estimate the population correlation, a standardized measure for the strength of the linear relationship between X and Y. See Chapter 3.7 in the book for a more detailed treatment of these estimators.

As for variance and standard deviation, these estimators are implemented as `R` functions in the `stats` package. We can use them to estimate population covariance and population correlation of the fictional data on age and earnings.

```
# compute sample covariance of X and Y
cov(X, Y)
```

```
## [1] 213.934
```

```
# compute sample correlation between X and Y
cor(X, Y)
```

```
## [1] 0.706372
```

```
# equivalent way to compute the sample correlation
cov(X, Y)/(sd(X) * sd(Y))
```

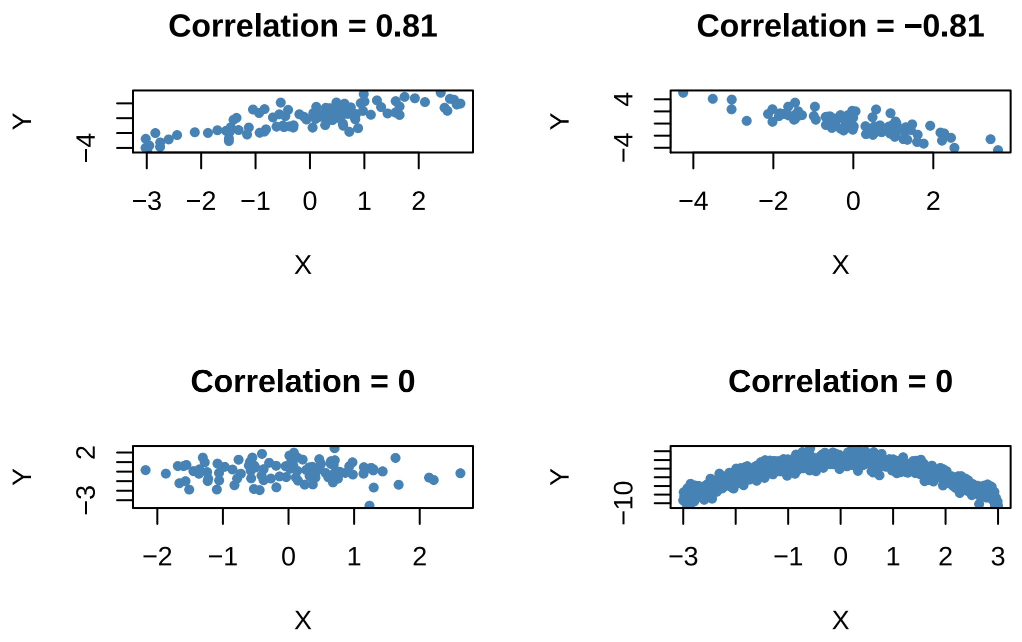
```
## [1] 0.706372
```

The estimates indicate that X and Y are moderately correlated.

The next code chunk uses the function `mvnorm()` from package `MASS` to generate bivariate sample data with different degree of correlation.

```
library(MASS)

# set random seed
set.seed(1)

# positive correlation (0.81)
example1 <- mvrnorm(100,
                    mu = c(0, 0),
                    Sigma = matrix(c(2, 2, 2, 3), ncol = 2),
                    empirical = TRUE
                    )

# negative correlation (-0.81)
example2 <- mvrnorm(100,
                    mu = c(0, 0),
                    Sigma = matrix(c(2, -2, -2, 3), ncol = 2),
                    empirical = TRUE
                    )

# no correlation
example3 <- mvrnorm(100,
                    mu = c(0, 0),
                    Sigma = matrix(c(1, 0, 0, 1), ncol = 2),
                    empirical = TRUE
                    )

# no correlation (quadratic relationship)
X <- seq(-3, 3, 0.01)
Y <- -X^2 + rnorm(length(X))

example4 <- cbind(X, Y)

# divide plot area as 2-by-2 array
par(mfrow = c(2, 2))
```

```r
# plot data sets
plot(example1, col = 'steelblue', pch = 20, xlab = 'X', ylab = 'Y',
     main = "Correlation = 0.81")

plot(example2, col = 'steelblue', pch = 20, xlab = 'X', ylab = 'Y',
     main = "Correlation = -0.81")

plot(example3, col = 'steelblue', pch = 20, xlab = 'X', ylab = 'Y',
     main = "Correlation = 0")

plot(example4, col = 'steelblue', pch = 20, xlab = 'X', ylab = 'Y',
     main = "Correlation = 0")
```



3.8 Exercises

This interactive part of URFITE is only available in the HTML version.

Bibliography

Heiss, F. (2016). *Using R for Introductory Econometrics.* CreateSpace Independent Publishing Platform.

Kleiber, C. and Zeileis, A. (2008). *Applied econometrics with R.* Springer.

Stock, J. and Watson, M. (2015). *Introduction to Econometrics, Third Update, Global Edition.* Pearson Education Limited.

Wooldridge, J. (2016). *Introductory econometrics.* Cengage Learning, sixth editon edition.