Practice Lecture 1: Basics of R and probability distributions

Erlis Ruli (ruli@stat.unipd.it)

14 October 2021

1 R Basics

The `R` (R Core Team, 2020) programming language and environment is designed for statistical analysis. It is open, free (see https://www.R-project.org/) and is written and maintained by a very active community of statisticians. A major design feature is extendability. `R` makes it very straightforward to code up statistical methods in a way that is easy to distribute and for others to use. The first place to look for information on getting started with `R` is http://cran.r-project.org/manuals.html. I will assume that you have installed `R`, and have at least discovered the function `q()` for quitting `R`.

You might find it easier to work with `R` if you use an IDE. For this I recommend R Studio (https://rstudio.com/); the free Desktop edition is enough.

The following command creates the vector named "a", and assigns to this vector the numbers $1, 2, \ldots, 10$.

```
> a <- 1:10
```

The symbol "<-" is an assignment symbol, the colon ":" is a function that creates regular sequence of integers and the symbols "<" on the left is the `R` prompt, just like the $ in a unix terminal. Square brackets are used for subsetting vector elements (In `R`, indexing of objects starts from 1 and not from 0 as for instance it happens in `C`). For instance, the third element of a, that is a_3 is

```
> a[3]
```

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

The symbol "## [1]" says that the `R` output has only one row and the value delivered in this case is 3. We can select more than one element at time by, for instance the first three elements of the vector a are

```
> a[1:3]
```

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

Elements of a such as the 2nd, 3rd and 7th can be extracted by

```
> a[c(2,3,7)]
```

```
## [1] 2 3 7
```

where we used the function `c`, which stands for "concatenate". We could also have used the concatenate function to define a, such as

```
> a <- c(1,2,3,4,5,6,7,8,9,10)
```

Matrices are built by the function `matrix`. For example, here is a 5×2 matrix

```
> A <- matrix(c(1:10), ncol=2)
```

We will use the terms "`R` function" and "`R` command" almost interchangeably. Here we are actually creating internally a vector such as the vector a above and then we reshape it to fill a 5×2 matrix named A. See `?matrix` for further details.

Accessing matrix elements is done via a pair of square brackets with double index notation:

```
> # (1,2) element of A is
> A[1,2]
```

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

We see here also how to place comments in R, i.e. any text preceded by the "#".

2 Generating random variates from a given distribution

2.1 The uniform

Let us start from the uniform random variable (r.v.). Recall that, if $U \sim \text{Unif}(0,1)$, then U has p.d.f. equal to one in the interval [0,1] and zero elsewhere. Here is the plot of the p.d.f. of a uniform r.v.

```
> par(mfrow = c(1,2))
> plot(dunif, xlim=c(-1,2),
+      ylab="f(x)", main = "The p.d.f of the Unif(0,1) r.v.")
> plot(punif, xlim=c(-1,2),
+      ylab="F(x)", main = "The d.f of the Unif(0,1) r.v.")
```

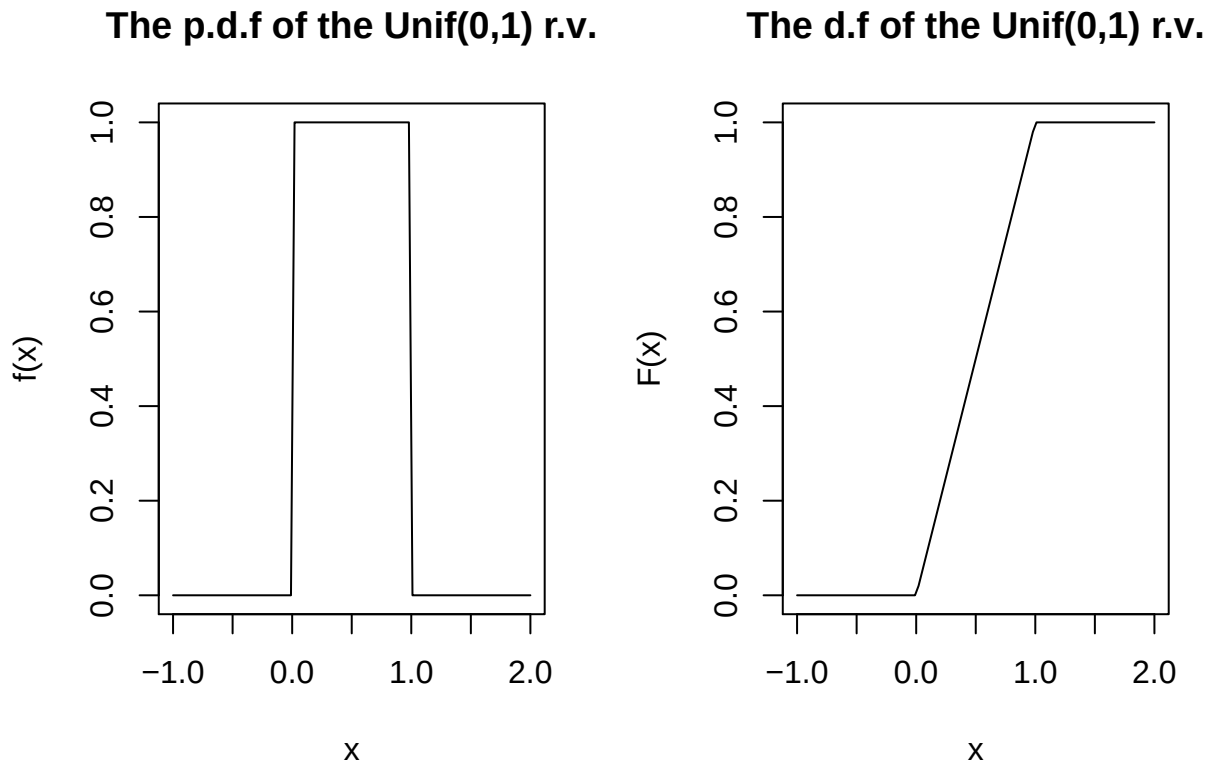


Figure 1: The p.d.f (left) and d.f. of the Unif(0,1) r.v..

To create the figure on the left, we used the function `dunif` which is R's built-in function for the p.d.f. of the $\text{Unif}(0,1)$ r.v. Then we used the function `plot` which is a generic R function for creating figures of many types. Built-in R function for dealing with probability distibutions can be categorised into four groups:

- functions starting by the letter `d`, e.g. dunif, dnorm, etc., give the probability density function of the r.v.;
- functions starting by the letter `r`, e.g. runif, `rnorm`, etc., are used for generating random draws;

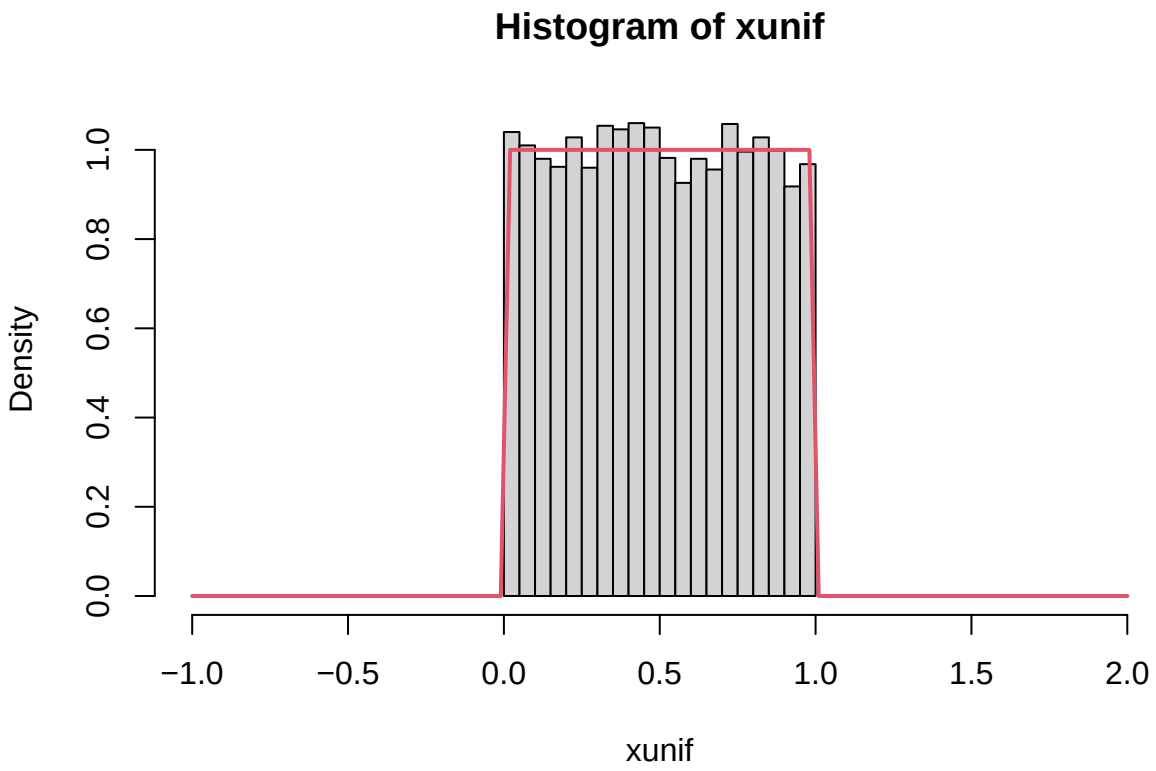
- functions starting by the letter `p`, e.g. `punif`, `pnorm`, etc., give the distribution function;
- functions starting by the letter `q`, e.g. `qunif`, `qnorm`, etc., give the quantile function.

We now use the `runif` command to draw random samples of size 10^4. To make sure you get the figure below use the same random seed as mine.

```
> set.seed(2020)
> xunif <- runif(1e+4)
```

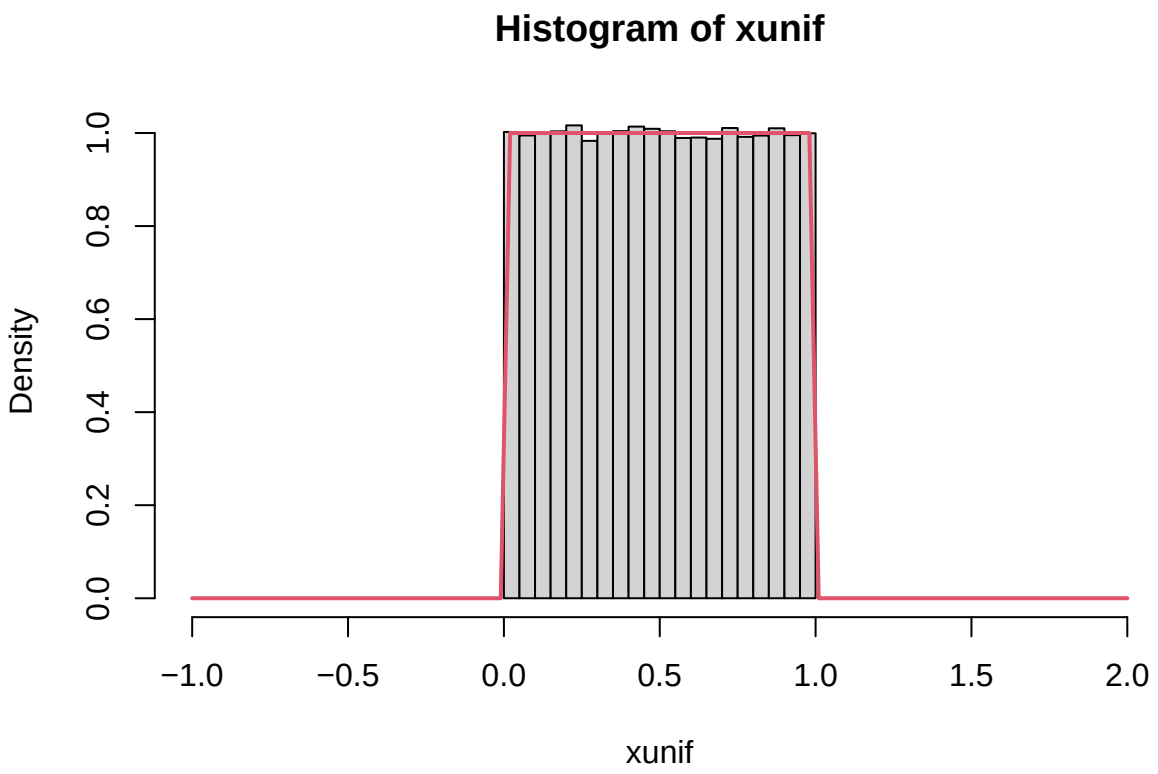
Let's see how uniform are our draws by constructing a scaled histogram (using option `freq=FALSE`) and then let's compare the latter with the the p.d.f. of the Unif$(0,1)$ r.v. We will see the histogram in the next Lecture, but for the time being...

```
> hist(xunif, xlim=c(-1,2), freq = FALSE)
> plot(dunif, xlim=c(-1,2), add=TRUE, col=2, lwd=2)
```



The histogram seem quite close to the p.d.f.. Changing the random seed (or simply not using a random seed) produces a slightly different histogram, each time we generate new random sample. Let us now increase the sample size

```
> set.seed(2020)
> xunif <- runif(3e+5)
> hist(xunif, xlim=c(-1,2), freq = FALSE)
> plot(dunif, xlim=c(-1,2), add=TRUE, col=2, lwd=2)
```



The histogram is now extremely close to the p.d.f.. Thus the fealing is that for the sample size going to infinity the histogram tends to converge to the p.d.f. thus the sample we have got by runif is actually a random sample which has distribution U(0,1).

2.2 Monte Carlo approximations

Approximations of features of a probability distribution by means of simulations are generally known as Monte Carlo approximations. For instance, above we approximated the density of the Unif(0,1) distribution by a histogram. This approximations are very useful when the involved probability distribution is difficult to work with.

Here is how this method works for approximating features of a distribution such as the expected value or the probability that the r.v. is whithin a given interval. For simplicity, we focus here on the Unif(0,1) distribution but note that the technique applies to any probability distribution.

First note that the Unif(0,1) distribution has expected value

$$E(X) = \int_0^1 x 1_{[0,1]}(x) dx = \frac{1}{2}.$$

Then we generate N samples $x_1, \ldots, x_N$ from the Unif(0,1) distribution. This sample is also called Monte Carlo sample. Finally, we replace the integral by the sum divided by N, i.e.

$$\int_0^1 x 1_{[0,1]}(x) dx \approx \frac{1}{N} \sum_{i=1}^{n} x_i.$$

Thanks to the LLN, as $N \to \infty$ the average $\frac{1}{N} \sum_{i=1}^{n} X_i$ will converge in probability to the true value $\mu = \frac{1}{2}$.

As another example, suppose we wish to compute $P(1/2 < X < 2/3)$, when $X \sim \text{Unif}(0, 1)$. Again this can be computed exactly by

$$P(1/2 < X < 2/3) = \int_{1/2}^{2/3} 1_{[0,1]}(x)dx = \frac{1}{6}.$$

To figure out how to approximate this quantity via a Monte Carlo method, first write this probability in the form of an expectation. In particular, let $1_{(1/2,2/3)}(x)$ be an indicator function which takes value 1 if $x \in (1/2, 2/3)$ and zero otherwise. Then we have that

$$\begin{aligned} P(1/2 < X < 2/3) &= \int_{1/2}^{2/3} 1_{[0,1]}(x)dx \\ &= \int_0^1 1_{(1/2,2/3)}(x)1_{[0,1]}(x)dx \\ &= E[1_{(1/2,2/3)}(X)]. \end{aligned}$$

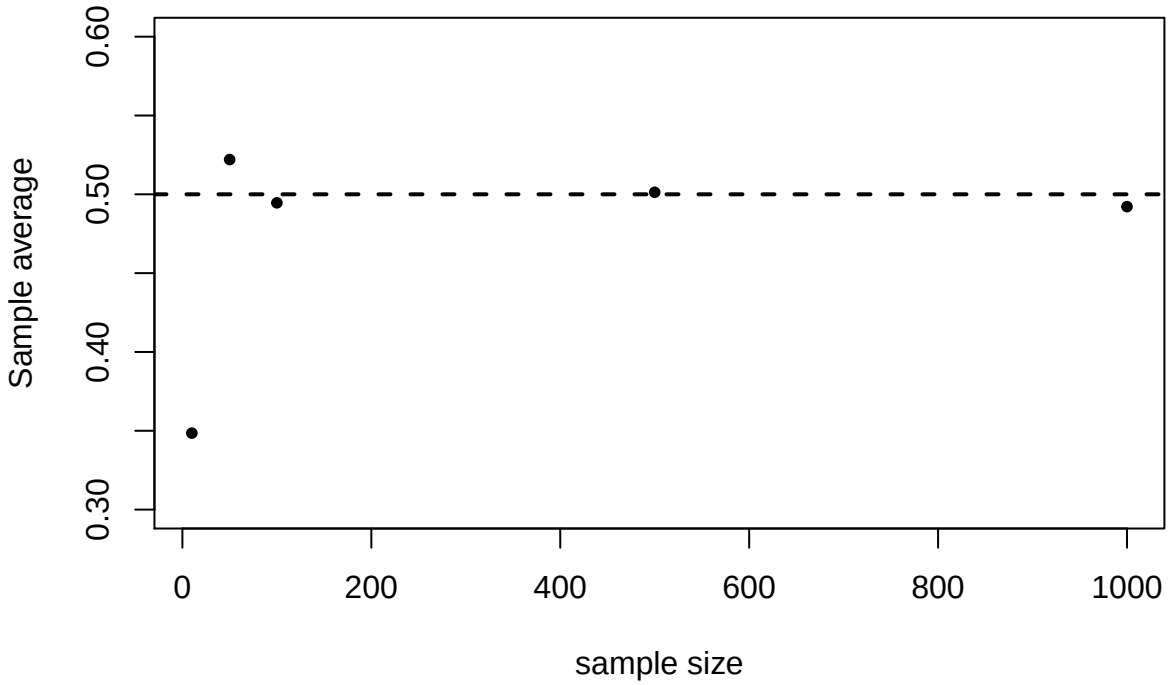
Thus to approximate the expectation we again draw N samples $x_1, \ldots, x_N$ from the Unif(0,1) distribution and replace the integral by the sum divided by N

$$E[1_{(1/2,2/3)}(X)] \approx \frac{1}{N}\sum_{i=1}^{N} 1_{(1/2,2/2)}(x_i),$$

Note that $1_{(1/2,2/2)}(x_i)$ gives only 1's and 0's thus the sum over N gives the propoprtion or the relative frequency of 1's. Thanks to the LLN, as $N \to \infty$ this relative frequency will converge to the true probability $\frac{1}{6}$.

As a numerical illustration let's see how closer we can get to the expected value of the Unif(0,1) as N increases. By the LLN we know that the average converges to the true mean for the sample size going to infinity. Let's see if that's the case with our random draws.

```
> set.seed(2020)
> xunif1 <- runif(10)
> set.seed(2020)
> xunif2 <- runif(50)
> set.seed(2020)
> xunif3 <- runif(1e+2)
> set.seed(2020)
> xunif4 <- runif(5e+2)
> set.seed(2020)
> xunif5 <- runif(1e+3)
> averages <- c(mean(xunif1), mean(xunif2), mean(xunif3),mean(xunif4),mean(xunif5))
> plot(x = c(10,50,1e+2,5e+2,1e+3), y=averages, ylim=c(0.3, 0.6), pch=20,
+      ylab = "Sample average",
+      xlab="sample size")
> abline(h = 1/2, lwd=2, lty=2)
```



We note that the sample averages get closer to 1/2, as the sample size increases, just as we suspected.

3 The Gaussian distribution

Gaussian or normal random draws can be performed by the command `rnorm`. For instance, 10^3 draws from the standard normal distribution can be obtained by

```
> set.seed(2020)
> # generate 1000 r. draws from the N(0,1) dist
> xnorm <- rnorm(1e+3)
```

If we want random draws from say N(1/2,2) then we have two options: either we resort to the options of the `rnorm`, i.e.

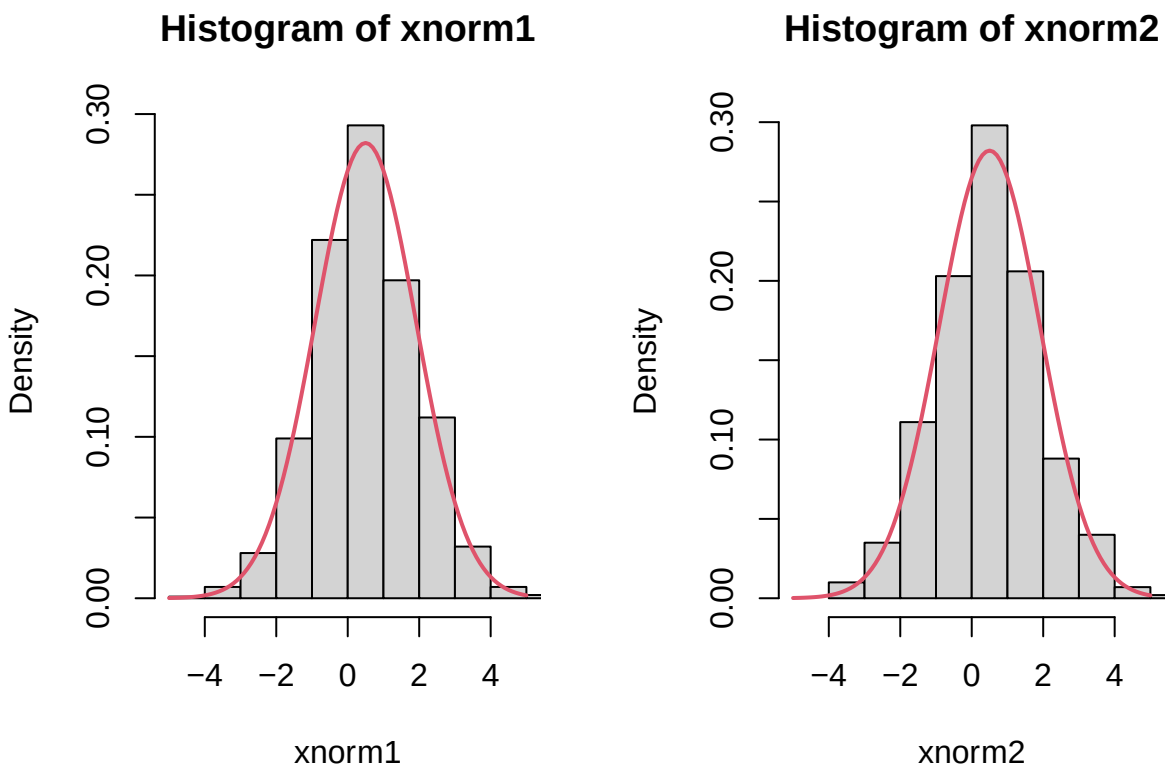
```
> # generate 1000 r. draws from the N(1/2,3) dist
> xnorm1 <- rnorm(1e+3, mean = 1/2, sd = sqrt(2))
```

where note that `R` the option `sd` is used to specify the standard deviation of the required normal distribution. The second alternative is to resort to the properties of the normal distribution (see L0) and thus

```
> mu <- 1/2
> st.dev <- sqrt(2)
> xnorm2 <- mu + st.dev*xnorm
```

We can comapre the random draws using a histogram with the true p.d.f. as below

```
> par(mfrow = c(1,2))
> hist(xnorm1, xlim=c(-5,5), freq = FALSE)
> plot(function(x) dnorm(x, mean = 1/2, sd=sqrt(2)),
+      xlim=c(-5,5), add=TRUE, col=2, lwd=2)
> hist(xnorm2, xlim=c(-5,5), freq = FALSE)
> plot(function(x) dnorm(x, mean = 1/2, sd=sqrt(2)),
+      xlim=c(-5,5), add=TRUE, col=2, lwd=2)
```



Here we see also the use of the command `function`. This command is used to define user-defined functions. For instance, suppose we want to define the function $f(x) = (x^3 + \log(x))/x$. The `R` code for this is

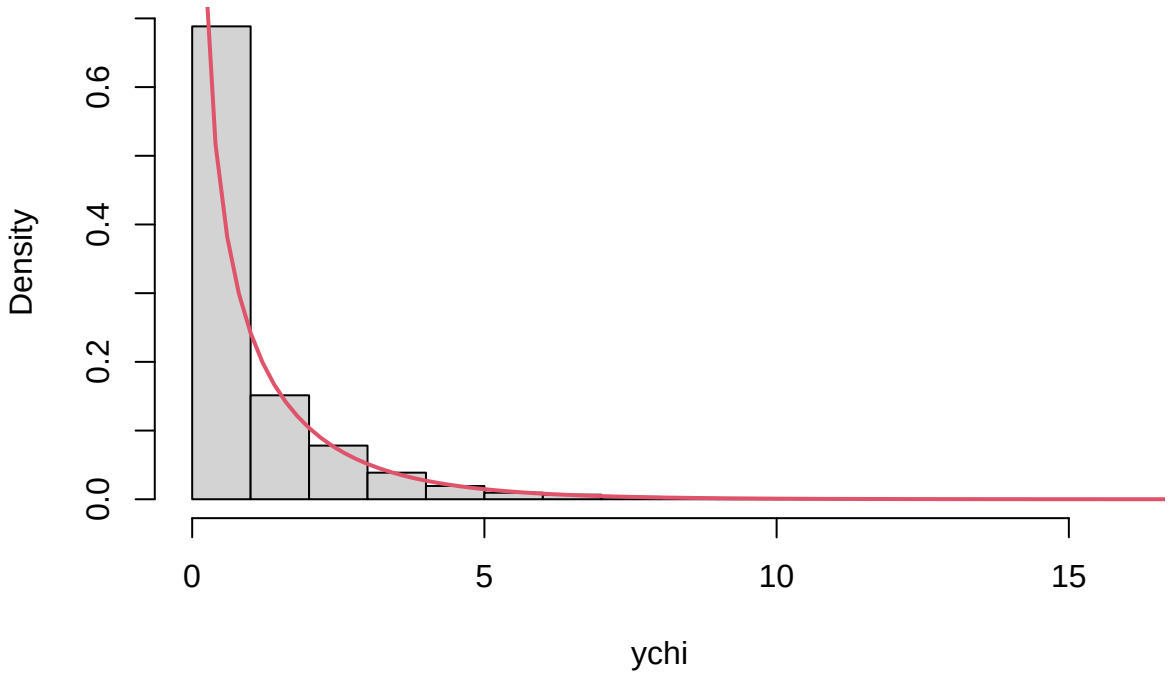
```
> fx <- function(x) {
+   out1 <- x^3 + log(x)
+   out2 <- out1/x
+   return(out2)
+ }
```

Functions in `R` can have more than one arguments. These can be of any type, vectors, matrices, lists, etc. and the return objects can be of any type. The only constraint is that functions cannot return multiple objects.

4 Transformation of a r.v.

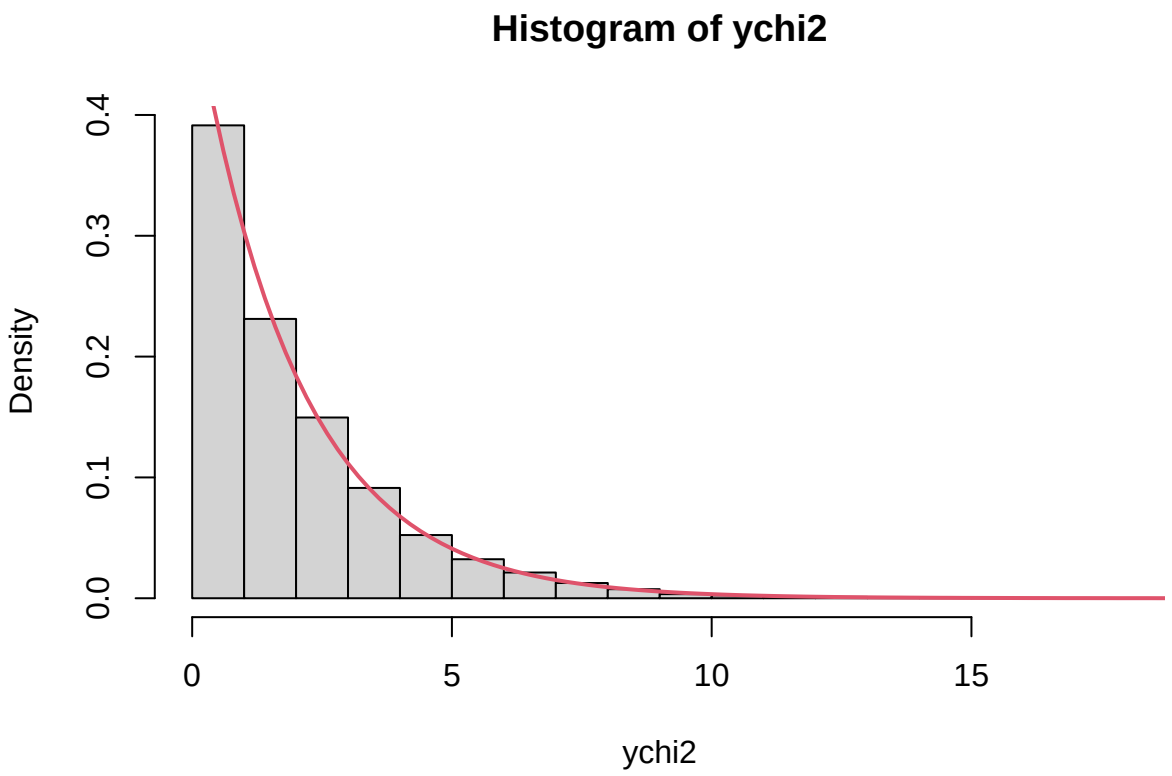
From Lecture 1 we know that if $Z \sim \mathrm{N}(0,1)$, then $Y = Z^2 \sim \chi^2_1$. Let's check this by simulation. The idea is to generate a large sample from the $\mathrm{N}(0,1)$ distribution, square all the values and then compare these squarred values by means of an histogram with the theoretical distribution, i.e. χ^2_1.

```
> set.seed(2020)
> znorm <- rnorm(1e+4)
> ychi <- znorm^2
> hist(ychi, freq = FALSE)
> plot(function(x) dchisq(x, df=1), add=TRUE, lwd=2, col=2, xlim=c(0,20))
```



We see that the histogram is quite close to the theoretical p.d.f., given by the function `dchisq`. Te be convinced it is not just luck, lets try sums of two r.v.. Recall that if $Z_1 \sim \mathrm{N}(0,1), Z_2 \sim \mathrm{N}(0,1)$ and Z_1 is independent from Z_2, then $Y = Z_1^2 + Z_2^2 \sim \chi_2^2$. So we have

```
> set.seed(2020)
> znorm1 <- rnorm(1e+4)
> znorm2 <- rnorm(1e+4)
> ychi2 <- znorm1^2+ znorm2^2
> hist(ychi2, freq = FALSE)
> plot(function(x) dchisq(x, df=2), add=TRUE, lwd=2, col=2, xlim=c(0,20))
```



This tells us that if we wanted to generate values from the χ^2_2 distribution, we can generate two random draws from the $N(0,1)$, square each of them and then sum. Of course we do not need to take such a long way, since `R` has the function `rchisq` which we can be use to generate samples from χ^2_p, for any p.

Exercise: Generate 10^4 random draws from $N(0,1)$, take the exponential and plot the relative histogram. This distribution is called log-normal (see its p.d.f in the help of the comand `dlnorm` or at https://en.wikipedia.org/wiki/Log-normal_distribution). Make sure that the histogram is close to the theoretical distribution. Mathematically, you are required to show that if $Z \sim N(0,1)$ then $Y = \exp(Z) \sim \log\mathrm{N}(\mu, \sigma)$ with parameters μ, σ to be defined from those of Z.

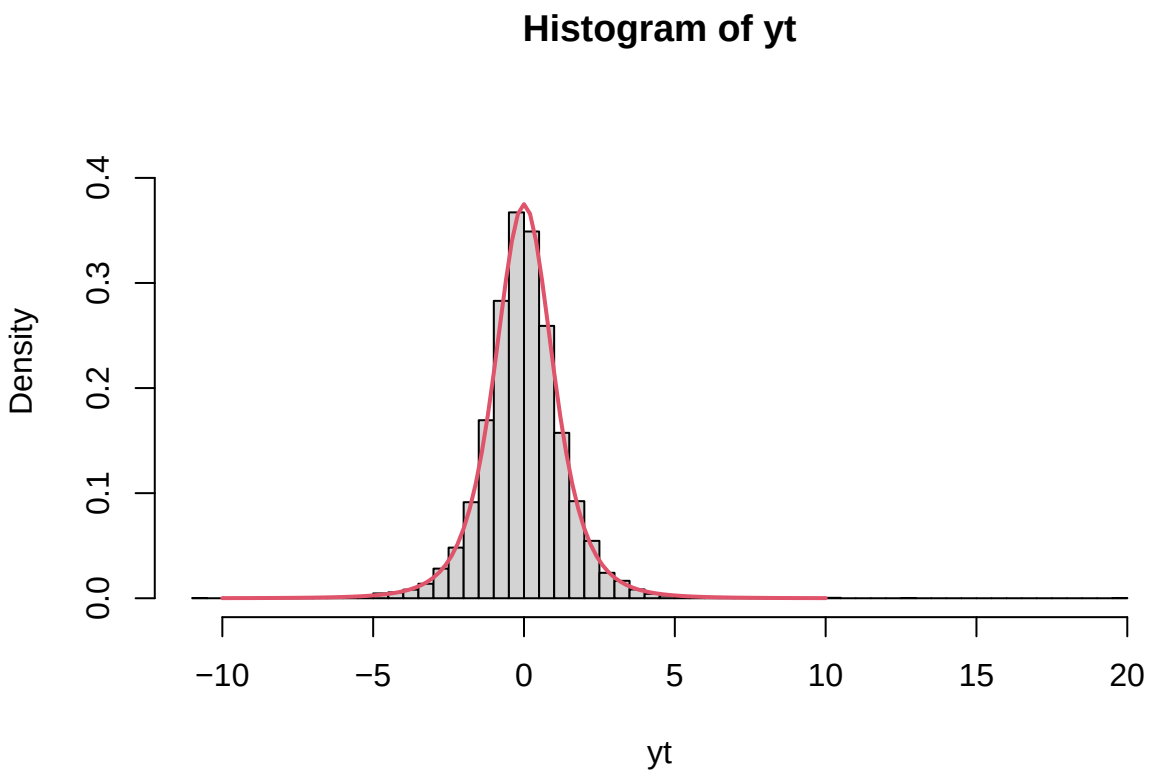
5 Generating t-Student random variates

The t-Student r.v. with ν degrees of freedom is defined (see L1) by $T = \frac{Z}{\sqrt{X/\nu}}$, where $Z \sim \mathrm{N}(0,1)$ and $X \sim \chi^2_\nu$. This definition immediately gives us a way for generating t-Student random samples. That is

```
> # drawing from t-Student distribution with nu=4
> # using its definition
> set.seed(2020)
> nu <- 4
> znorm <- rnorm(1e+4)
> xchi <- rchisq(1e+4, df=4)
> yt <- znorm/sqrt(xchi/nu)
```

Now we compare the samples with the theoretical p.d.f.

```
> hist(yt, freq = FALSE, breaks = 50, ylim=c(0,0.45))
> plot(function(x) dt(x, df=nu), add=TRUE, lwd=2, col=2, xlim=c(-10,10))
```



To have a clearer picture, we increased the number of bars with respect to the default option, by specifying the option `breaks=50`. Of course, `R` has a built-in function for generating t-Student random draws, it's called `rt`. Above we used the function `dt`, which gives the p.d.f of the a t-Student r.v..

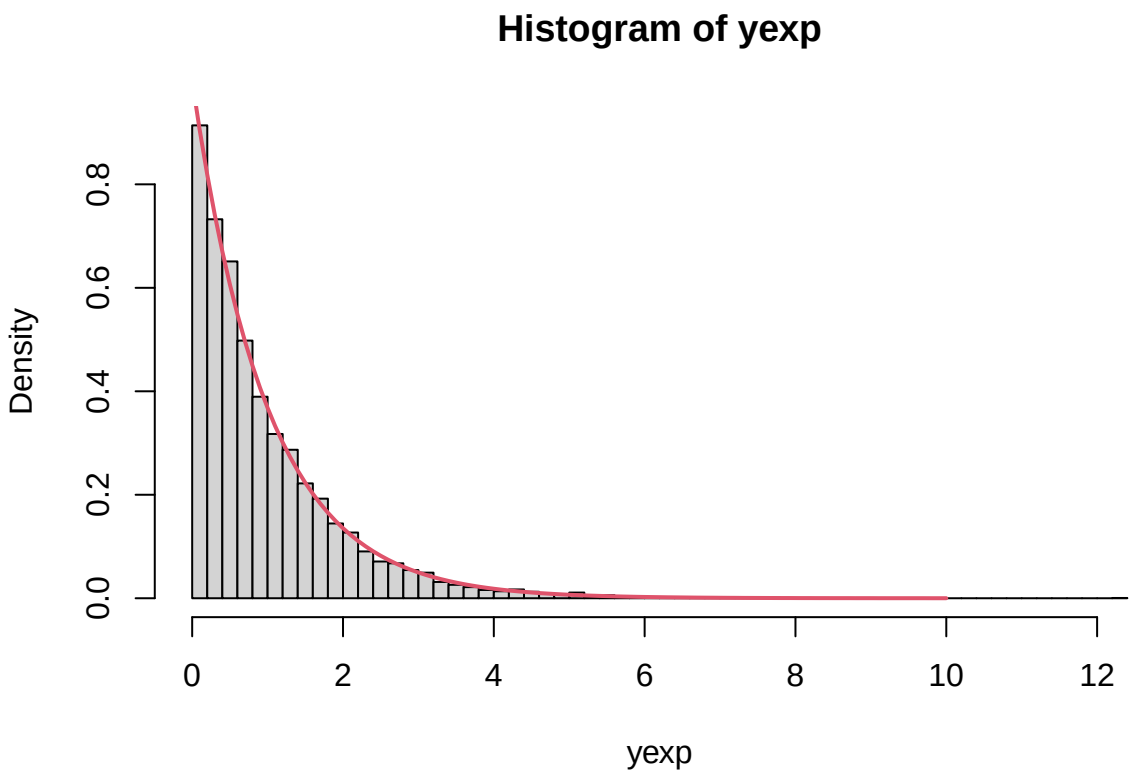
6 The inverse transform sampling method

We illustrate the inverse transform sampling (ITS) method by the example of an Exp(1) r.v.. We saw that the d.f. for this r.v. is $F(x) = 1 - \exp(-x)$ and thus the quantile function is $F^{-1}(p) = -\log(1-u)$. Here is the code for drawing 10^4 samples with the ITS method.

```
> set.seed(2020)
> unif <- runif(1e+4)
> yexp <- -log(1-unif)
> # alternatively we can also use
> # yexp1 <- -log(unif)
> # becasue U ~ Unif(0,1), then also 1-U ~ Unif(0,1)
```

Again, we comapre the distribution of the samples with the theoretical one by means of a histogram.

```
> hist(yexp, freq = FALSE, breaks = 50)
> plot(dexp, add=TRUE, lwd=2, col=2, xlim=c(0,10))
```



Of course, R has a built-in function for generating Exp(1) random draws, it's called `rexp`.

Exercise: Generate 10^4 samples from the Exp(3/2) distribution by the ITS method.

7 The Central Limit Theorem

7.1 Convergence of binomial r.v.

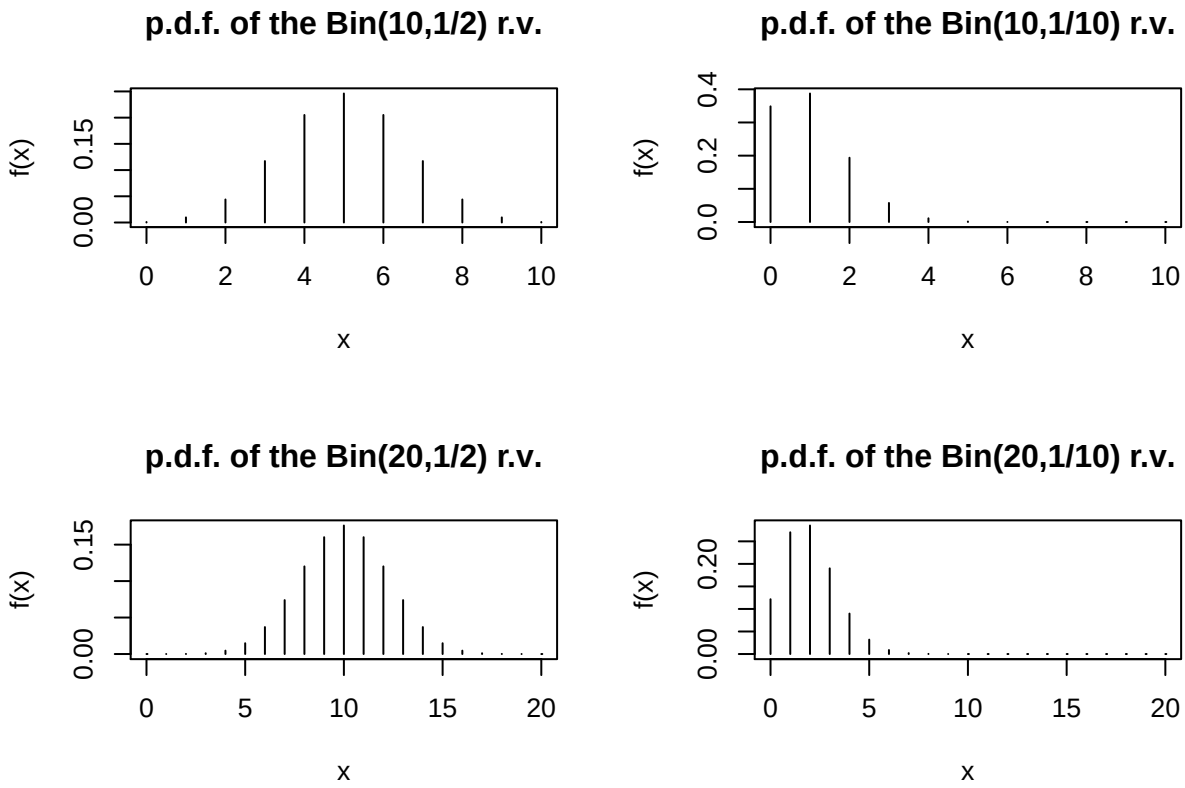
From theory we know that if $Y \sim \text{Bin}(n, \theta)$, then $\frac{Y-E(Y)}{\sqrt{var(Y)}} = \frac{Y-n\theta}{\sqrt{n\theta(1-\theta)}} \xrightarrow{d} \text{N}(0,1)$. Recall also that the CLT applies here because Y is the sum of n Bernoulli r.v..

Let us see how this works in practice. Consider the following four scenarios

- $n = 10, \theta = 1/2$
- $n = 10, \theta = 1/10$
- $n = 20, \theta = 1/2$
- $n = 20, \theta = 1/10$

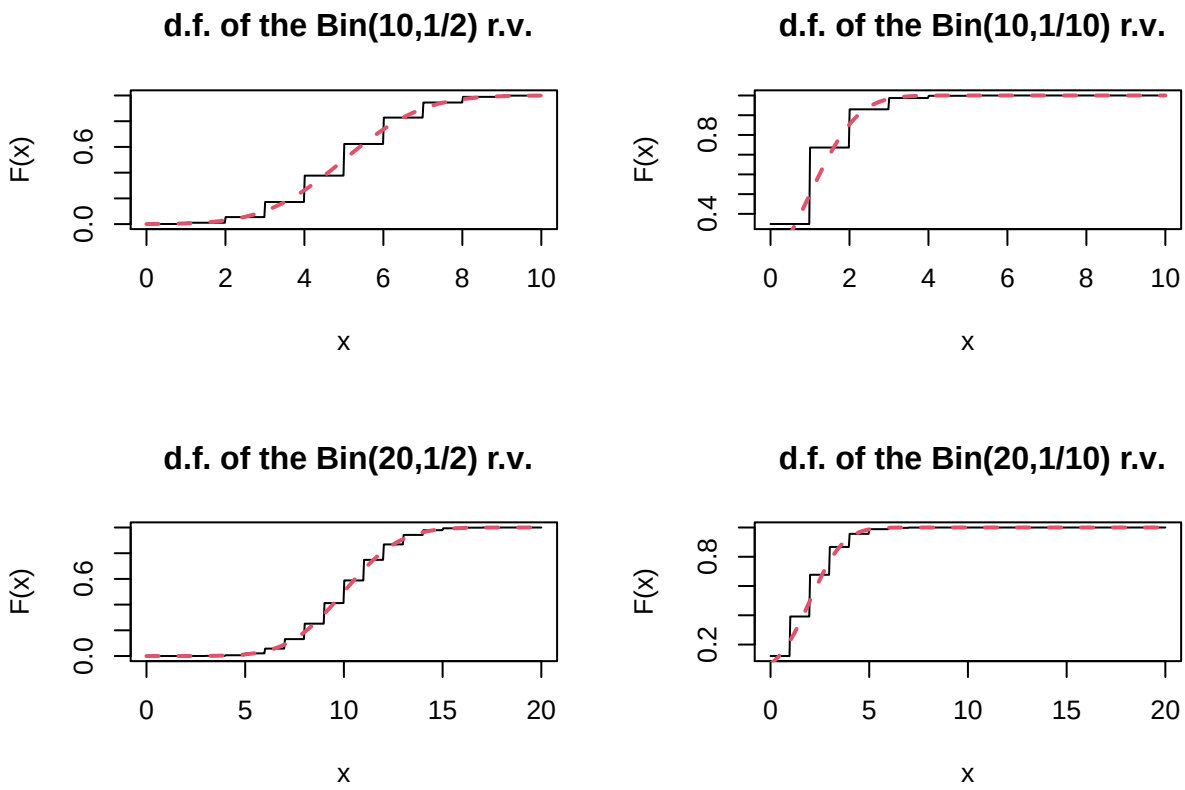
We consider first the probability density functions for each scenario. As we see from the plots of the p.d.f. the closer the success probability to the boundary of the parameter space, the skewed is the p.d.f.

```
> par(mfrow=c(2,2))
> plot(x = 0:10, dbinom(0:10, size=10, prob=1/2), type="h",
+      ylab="f(x)", xlab="x", main="p.d.f. of the Bin(10,1/2) r.v.")
> plot(x = 0:10, dbinom(0:10, size=10, prob=1/10), type="h",
+      ylab="f(x)", xlab="x", main="p.d.f. of the Bin(10,1/10) r.v.")
> plot(x = 0:20, dbinom(0:20, size=20, prob=1/2), type="h",
+      ylab="f(x)", xlab="x", main="p.d.f. of the Bin(20,1/2) r.v.")
> plot(x = 0:20, dbinom(0:20, size=20, prob=1/10), type="h",
+      ylab="f(x)", xlab="x", main="p.d.f. of the Bin(20,1/10) r.v.")
```



In the following plot we show the d.f. of the binomial r.v. and the d.f. of the the normal distbution as implied by the CLT

```
> par(mfrow = c(2,2))
> plot(function(x) pbinom(x, size=10, prob=1/2),xlim=c(0,10), n=500,
+      ylab="F(x)", main = " d.f. of the Bin(10,1/2) r.v.")
> plot(function(x) pnorm(x, mean=10*1/2, sd=sqrt(10*1/4)),xlim=c(0,10),
+      add=TRUE, col=2, lwd=2, lty=2)
>
> plot(function(x) pbinom(x, size=10, prob=1/10),xlim=c(0,10), n=500,
+      ylab="F(x)", main = " d.f. of the Bin(10,1/10) r.v.")
> plot(function(x) pnorm(x, mean=10*1/10, sd=sqrt(10*9/100)),xlim=c(0,10),
+      add=TRUE, col=2, lwd=2, lty=2)
>
>
> plot(function(x) pbinom(x, size=20, prob=1/2),xlim=c(0,20), n=500,
+      ylab="F(x)", main = " d.f. of the Bin(20,1/2) r.v.")
> plot(function(x) pnorm(x, mean=20*1/2, sd=sqrt(20*1/4)),xlim=c(0,20),
+      add=TRUE, col=2, lwd=2, lty=2)
>
> plot(function(x) pbinom(x, size=20, prob=1/10),xlim=c(0,20), n=500,
+      ylab="F(x)", main = " d.f. of the Bin(20,1/10) r.v.")
> plot(function(x) pnorm(x, mean=20*1/10, sd=sqrt(20*9/100)),xlim=c(0,20),
+      add=TRUE, col=2, lwd=2, lty=2)
```



As expected, the larger the sample size, the closer is the $Bin(n, \theta)$ d.f. to the normal d.f. For values of θ near the boundary points, i.e. $\theta \approx 0$ or $\theta \approx 1$, the normal approximation is less accurate.

8 The multivariate normal distribution

For obvious geometrical reasons we focus here on the bivariate normal distribution. Thus we have a $X = (X_1, X_2)$.

We first define an R function that computes the density of the bivariate normal distribution

```
> # real-valued function with arguments:
> # x: vector (2 x 1), all other argumens are scalars
> dbvnorm <- function(x, mu1, mu2, sigma1.2, sigma2.2, sigma12){
+   # build the covariance matrix Sigma
+   Sigma = matrix(c(sigma1.2, sigma12, sigma12,sigma2.2),ncol=2)
+
+   # build the vector of means
+   mu = c(mu1, mu2)
+
+   # compute a first part the of p.d.f, aka the "kernel"
+   out1 = exp(-0.5 * t(x-mu) %*% solve(Sigma) %*% (x-mu))
+
+   # scale the kernel by the normalising constant
+   out2 = out1/((2*pi)^(2/2) * sqrt(det(Sigma)))
+
+   # return the output
+   return(out2)
+
+   # you can also omit print() by writing just
```

```
+   # out2
+ }
```

We have two options for drawing a bivariate distribution, either via a perspective plot or via countur levels. In the following we see both approaches.

Firstly, we need to construct a regular bivariate grid and evaluete our newly defined function over it. We have to re-arrange the grid values in order to have as input an $(m \times 2)$ matrix. The output will be an $(m \times 1)$ vector which has to be re-arranged into a square matrix.

```
> mu1 <- 1
> mu2 <- 2
>
> # reguular univ. grid on 1st dimension
> x1 <- seq(-2,5, len=100)
> head(x1)
```

```
## [1] -2.000000 -1.929293 -1.858586 -1.787879 -1.717172 -1.646465
```

```
> # reguular univ. grid on 2nd dimension
> x2 <- seq(-1, 6, len=100)
> head(x2)
```

```
## [1] -1.0000000 -0.9292929 -0.8585859 -0.7878788 -0.7171717 -0.6464646
```

```
> # regular bivariate grid expanded row-wise
> x1x2 <- expand.grid(x1, x2)
> head(x1x2)
```

```
##        Var1 Var2
## 1 -2.000000   -1
## 2 -1.929293   -1
## 3 -1.858586   -1
## 4 -1.787879   -1
## 5 -1.717172   -1
## 6 -1.646465   -1
```

We fix also $\Sigma = [\sigma_{ij}]$ by setting with $\text{var}(X_1) = \sigma_1^2 = 1$, $\text{var}(X_2) = \sigma_2^2 = 2$ and $\text{cov}(X_1, X_2) = \sigma_{12} = 1$. Thus

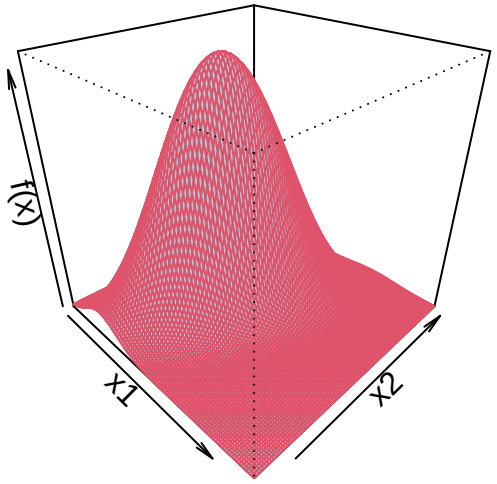
```
> sigma1.2 <- 1
> sigma2.2 <- 2
> sigma12 <- 1
> Sigma <- matrix(c(1,1,1,2),ncol=2)
```

Now we are ready to evaluate the bivariate normal p.d.f. over the grid. For this, we can either use a for/while loop or use `apply` function. We take the former approach here.

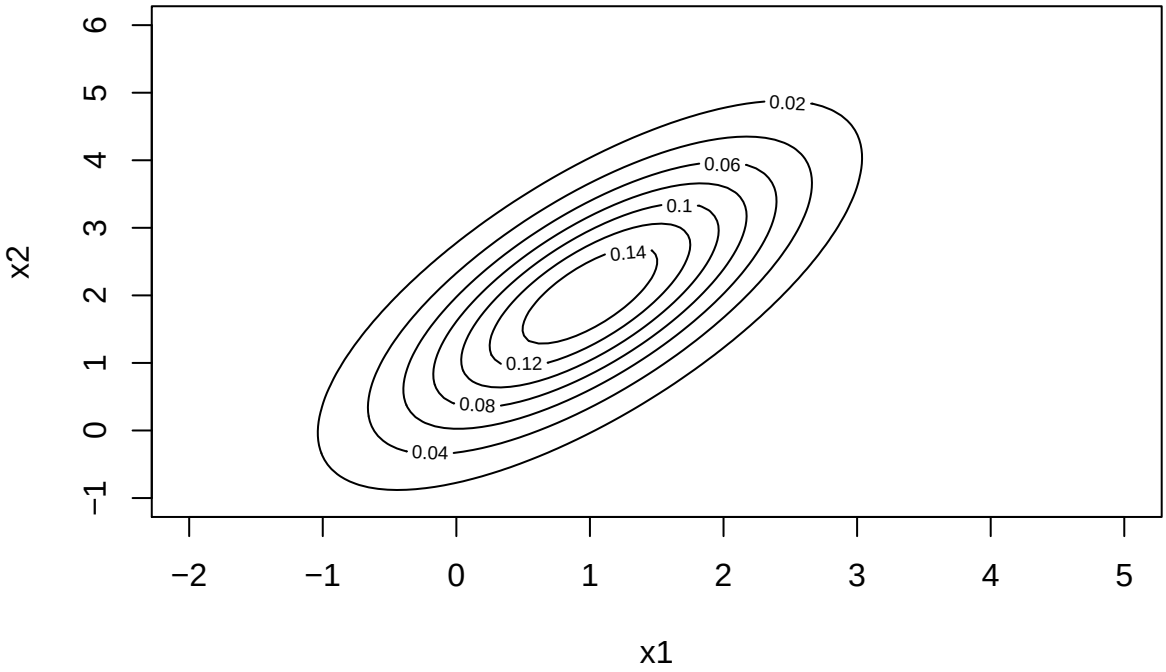
```
> z.pdf <- apply(x1x2,MARGIN = 1,
+                function(x) dbvnorm(x,
+                                    mu1=mu1,
+                                    mu2=mu2,
+                                    sigma1.2=sigma1.2,
+                                    sigma2.2=sigma2.2,
+                                    sigma12=sigma12))
> # we reorder the object to  100 x 100 matrix
> z.pdf.mat <- matrix(z.pdf, ncol=100, nrow=100, byrow = F)
```

Now we are ready for the plots

```
> persp(x1,x2,z.pdf.mat, zlab = "f(x)",
+         col="lightblue",theta=45,phi=30,
+   xlab = "x1", ylab = "x2", shade=0.01, border = 2)
```




```
> contour(x1,x2,z.pdf.mat, xlab="x1", ylab="x2")
```



Wee see that the contours are ellipses with main principal axis having positive slope. The center of the ellipse is at the point μ.