Applied Statistics – Exercise 4

Problems

T=Theoretical Exercise, R=R Exercise

#1. (T) Consider the continuous random variable X with the following probability density function

$$f(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1/2 & \text{if } 0 \leq x \leq 1 \\ 1/6 & \text{if } 1 \leq x \leq 2 \end{cases}$$

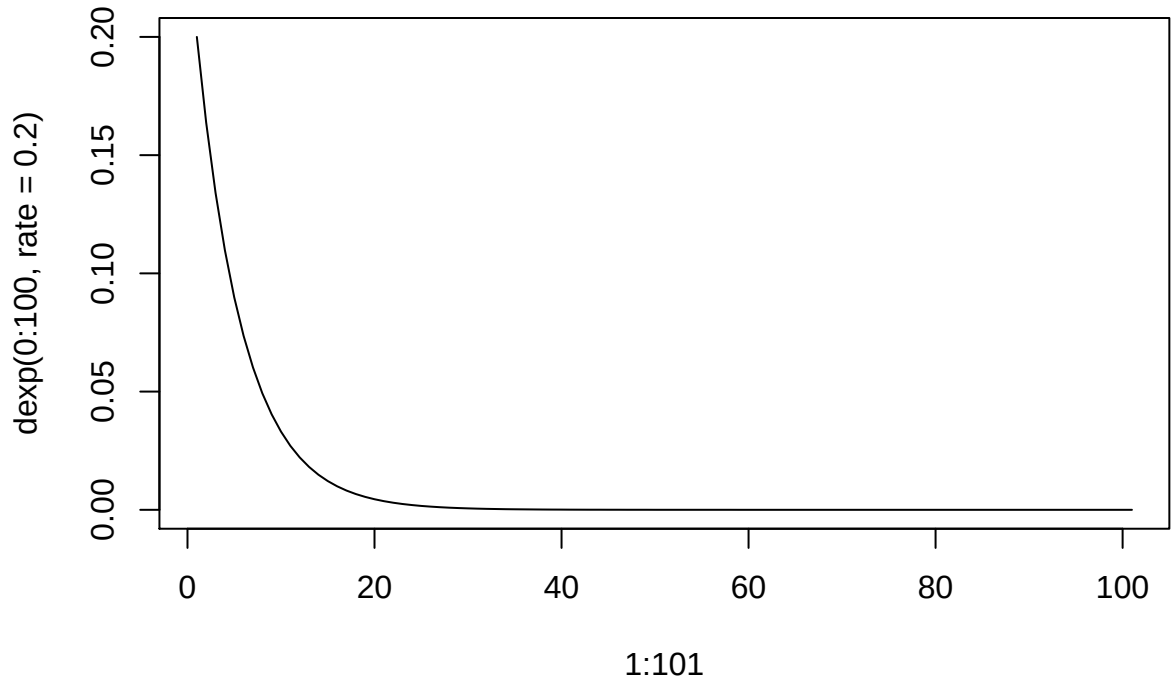
a) Draw, with pen and paper, the probability density function of X, i.e. $f(x)$.

b) Determine the expression of the distribution function $F(x)$ of X, and draw it.

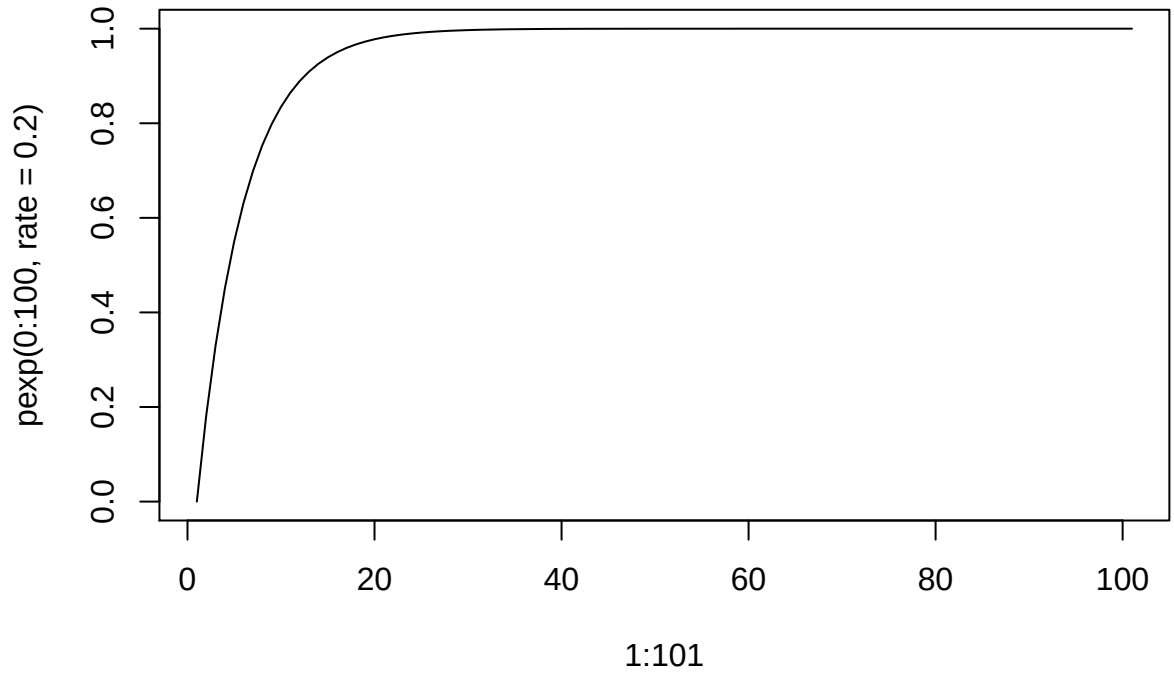
#2. (T) The following exercises are about cumulative distribution functions and probability distribution funcions.

a) Let X be a random variable with a $Exp(0.2)$ exponential distribution. Compute $P(X > 5.5)$.

```
plot(1:101, dexp(0:100, rate=.2), type='l')
```




```
plot(1:101, pexp(0:100, rate=.2),type='l')
```

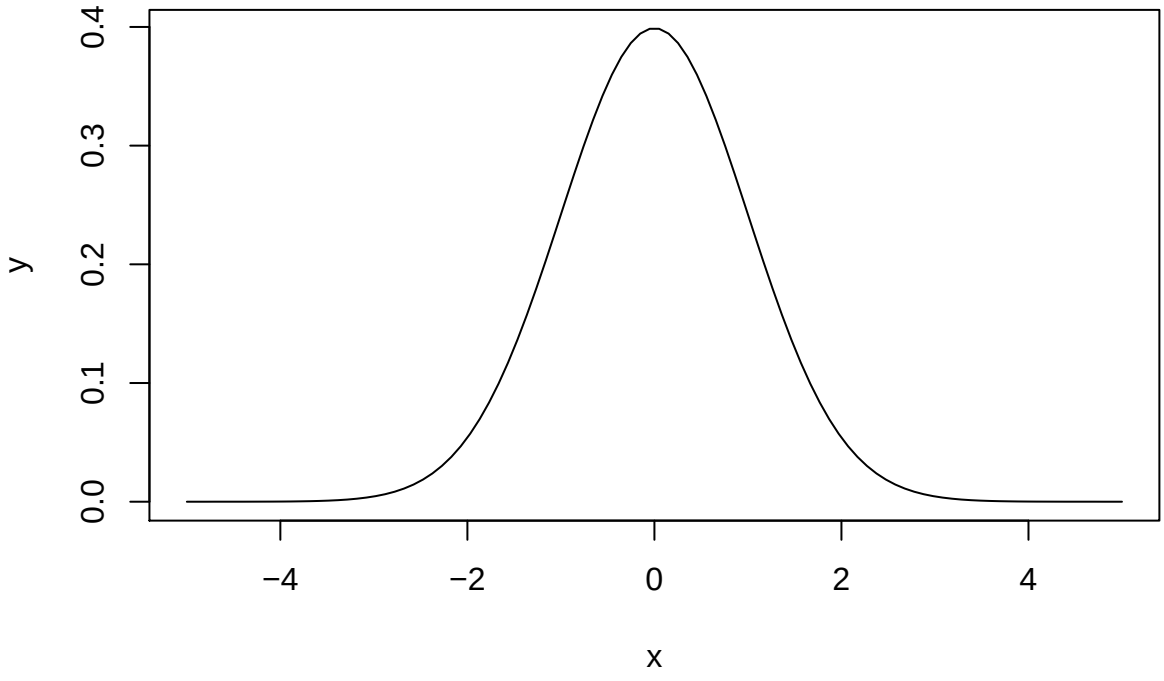



```
pexp(5.5, rate=.2)
```

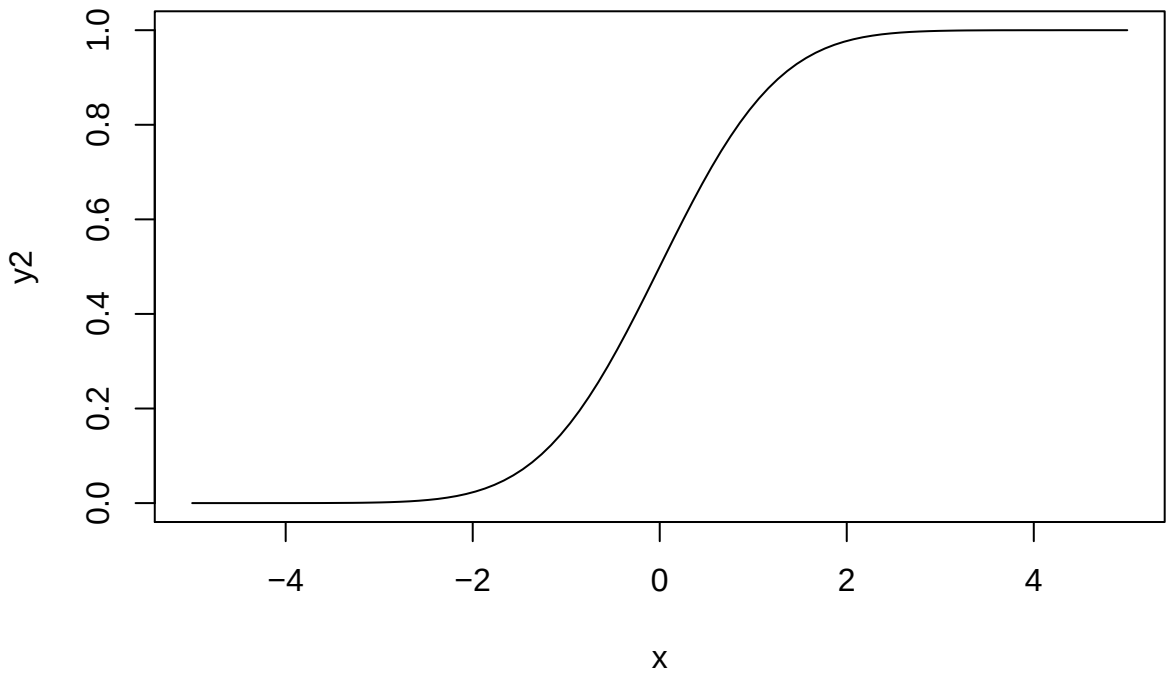
```
## [1] 0.6671289
```

b) Let X be a random variable with a $N(0,1)$ normal distribution. Compute $P(-0.2 < X < 0.4)$. You can use the table at the end of the book. In this case, note that the distribution $N(0,1)$ is symmetric around 0; use this fact to your advantage when you use the table.

```
x <- seq(-5, 5, length=100)
y <- dnorm(x) # standard normal distribution parameters by default
y2 <- pnorm(x)
plot(x, y, type="l", lwd=1)
```



```
plot(x, y2, type="l", lwd=1)
```




```
pnorm(.4)-pnorm(-.2)
```

```
## [1] 0.2346815
```

c) Consider the standard normal distribution, i.e. $N(0,1)$. What is the 15th percentile of this distribution? (Note that this is equivalent to the 0.15 quantile.)

```
qnorm(.15)
```

```
## [1] -1.036433
```

#3. (T) Consider a random variable X with uniform distribution $U(0,1)$. In class we have seen how this can be used to simulate the flip of a coin.

a) Use the random variable X to simulate the rolls of a dice with 4 faces, similarity to what we did in class. Each of the possible outcomes $1, 2, 3, 4$ must have equal probability.

b) Consider the random variable $Y = \lfloor 4X + 1 \rfloor$. That is, we take a random number generated according to the distribution of X, we multiply it by 4, we add 1, and we round it down to the nearest integer. What are the possible outcomes of Y ? What is the probability of each of these outcomes? Discuss the relationship between Y and the random variable you built in point (a).

#4. (R) This exercise introduces some random number generation facilities of R. As we saw in the last exercise session, in R there are several functions related to distributions: the names of such functions follow the pattern `xdistname`, where `distname` is the name of the distribution (e.g. geom, norm, binom. . .), and the prefix `x` is one of `d`, `p`, `q`, and `r`, with the following meaning:

- `d` gives the value of the probability mass function (or the probability distribution function in the case of continuous random variables)
- `p` gives the value of the cumulative distribution function
- `q` returns quantiles
- `r` returns a number sampled from the distribution

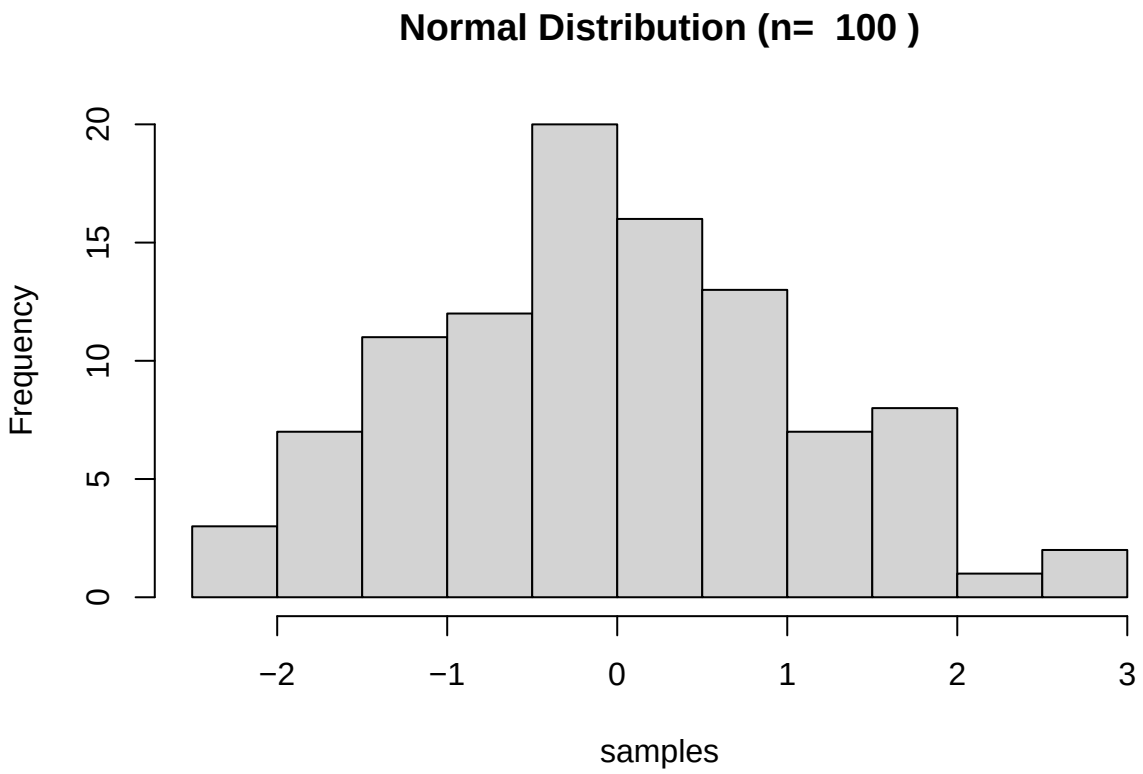
For simulation we are often interested in the last version of the function, the one prefixed with `r`. For instance, to generate numbers distributed according to the normal distribution, we use the function `rnorm`.

```
n <- 100
samples <- rnorm(n)
```

We can visualize how the sampled points are distributed using a *histogram*, which is a plot comprised of several columns: the height of each column is proportional to the number of points falling into the corresponding interval. To plot the histogram of a vector samples, you just need to call `hist(samples)`. In the following, the questions you have to answer with your R code are in *italic* font.

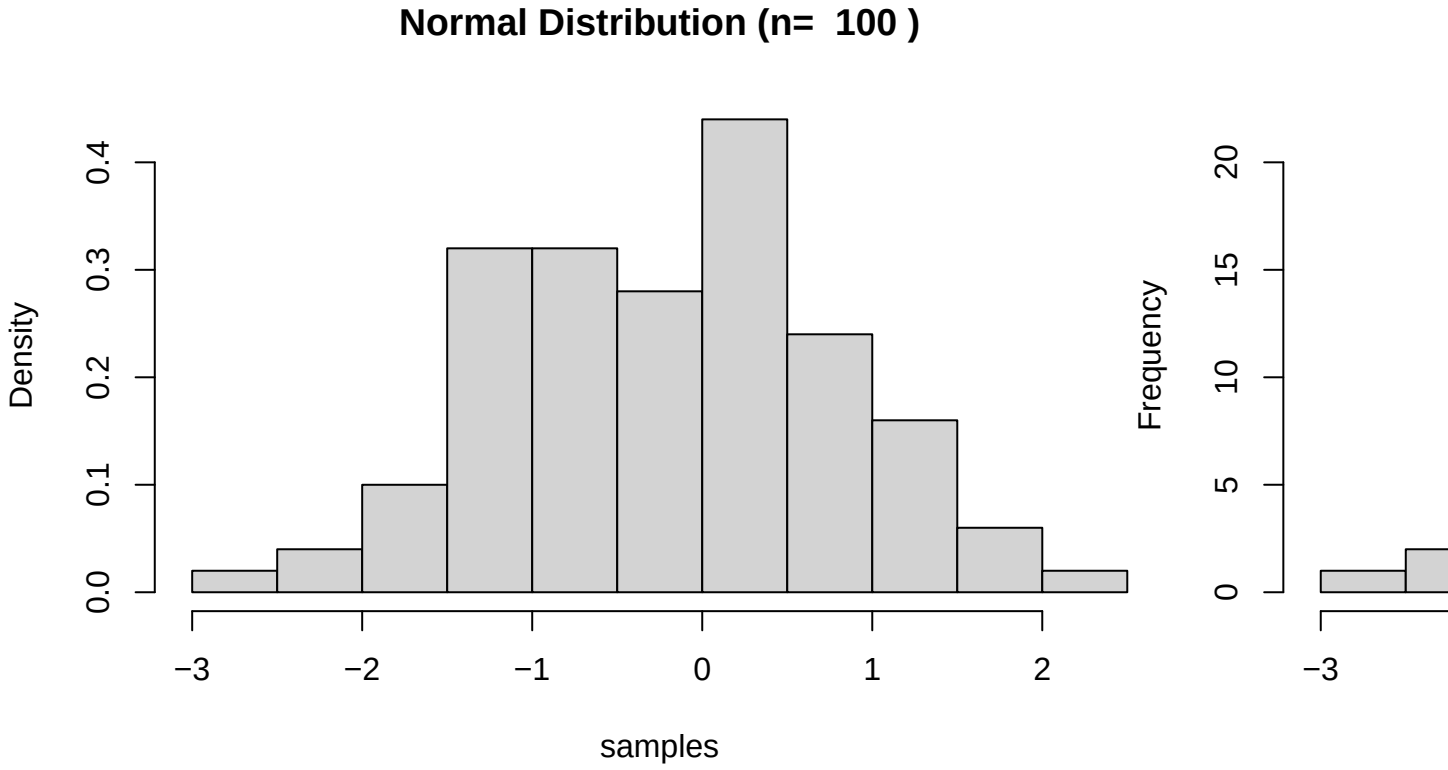
a) In a code block in your Rmarkdown file, define a variable `n` as above and sample `n` points using the rnorm distribution. *Then, plot the result using the hist function.* You can look up the documentation of any function in the help panel on the right of the RStudio window. *Does running the code block more than once change the result?*

```
n <- 100
samples <- rnorm(n)
hist(samples, main=paste('Normal Distribution (n= ', n, ')'))
```



b) The `hist` function takes several parameters. Among these, there is one called `freq`. Setting `freq = TRUE` does not change the output with respect to the default: the height of each column is still proportional to the overall count of points falling into each interval. Setting it to `FALSE` (as in `hist(samples, freq = FALSE)`) rescales the columns: each column is rescaled so that the sum of the areas of the columns is equal to 1. *Try to change the parameter and observe how the output plot changes.*

```
n <- 100
samples <- rnorm(n)
for (freq in c(FALSE, TRUE)) {
 hist(samples, main=paste('Normal Distribution (n= ', n, ')'), freq=freq)
}
```



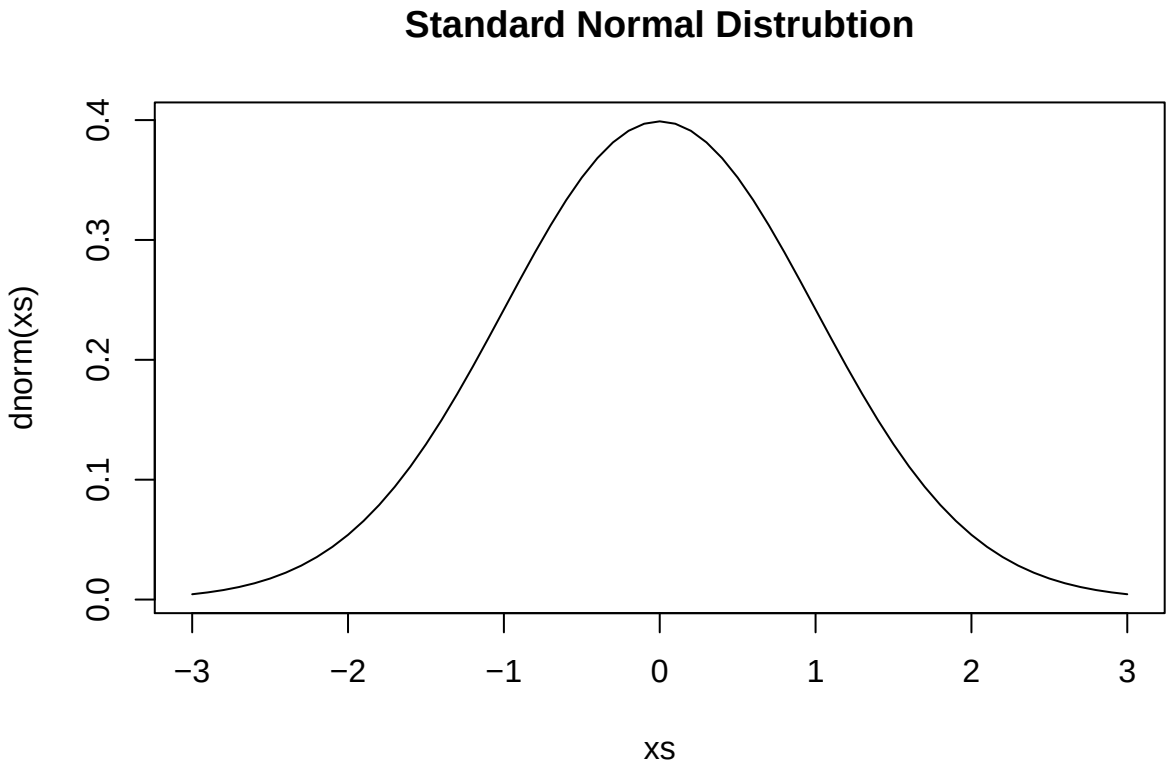
c) When `freq=FALSE`, we have that the total area of the rectangles is 1. This sounds similar to the definition of probability density function (page 57 in Dekking et al.), doesn't it? Let's try to overlay the plot of the probability density function (PDF) of a $N(0,1)$ random variable on top of our histogram. We can use the `dnorm(xs)` function to get the values of the (PDF) of the normal distribution corresponding to values `xs`. These values `xs` should approximately comprise the values we see on the `x`-axis of the histogram we plotted previously, and should have a fine enough resolution to be useful to plot a smooth curve of the PDF. A vector `xs` with these characteristics can be built using the following code

```
xs <- -30:30 / 10
xs
```

```
##  [1] -3.0 -2.9 -2.8 -2.7 -2.6 -2.5 -2.4 -2.3 -2.2 -2.1 -2.0 -1.9 -1.8 -1.7 -1.6
## [16] -1.5 -1.4 -1.3 -1.2 -1.1 -1.0 -0.9 -0.8 -0.7 -0.6 -0.5 -0.4 -0.3 -0.2 -0.1
## [31]  0.0  0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9  1.0  1.1  1.2  1.3  1.4
## [46]  1.5  1.6  1.7  1.8  1.9  2.0  2.1  2.2  2.3  2.4  2.5  2.6  2.7  2.8  2.9
## [61]  3.0
```

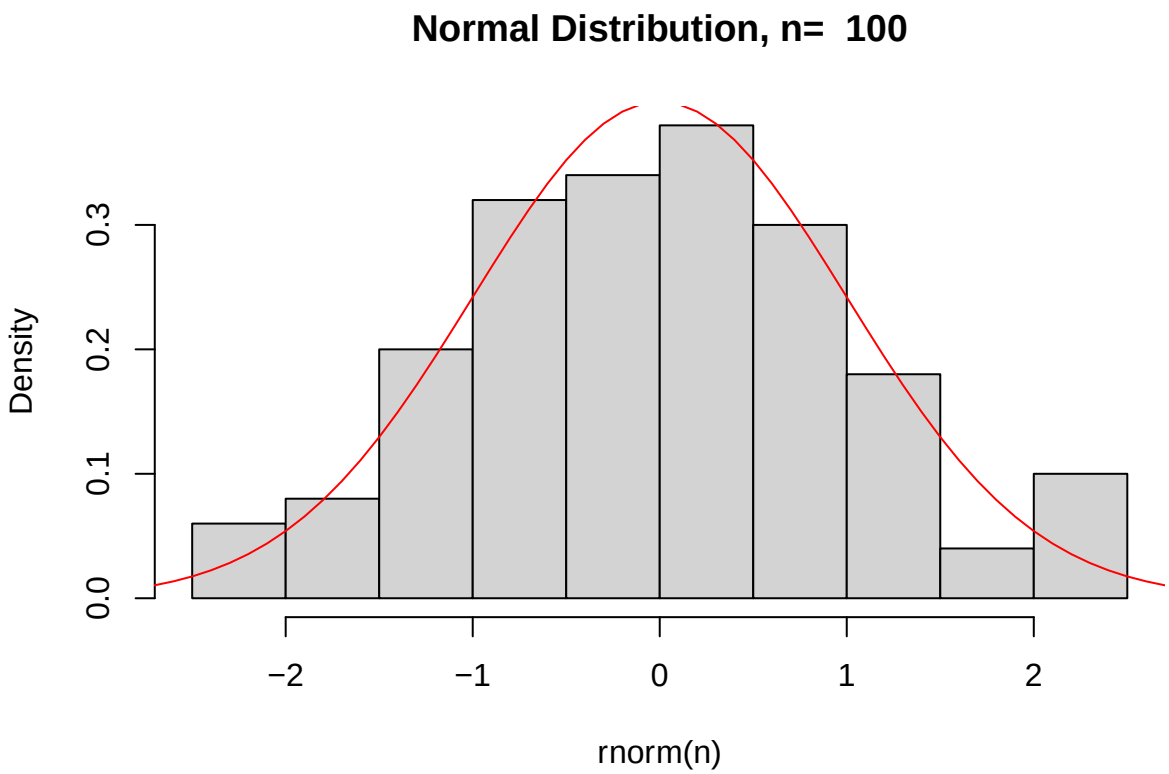
As you can see, the resulting vector ranges from -3 to 3 with increments of 0.1. *Using this vector* ***xs*** *as the input to the function* ***dnorm****, get the values of the probability distribution function, and plot them using the* ***lines*** *function.*

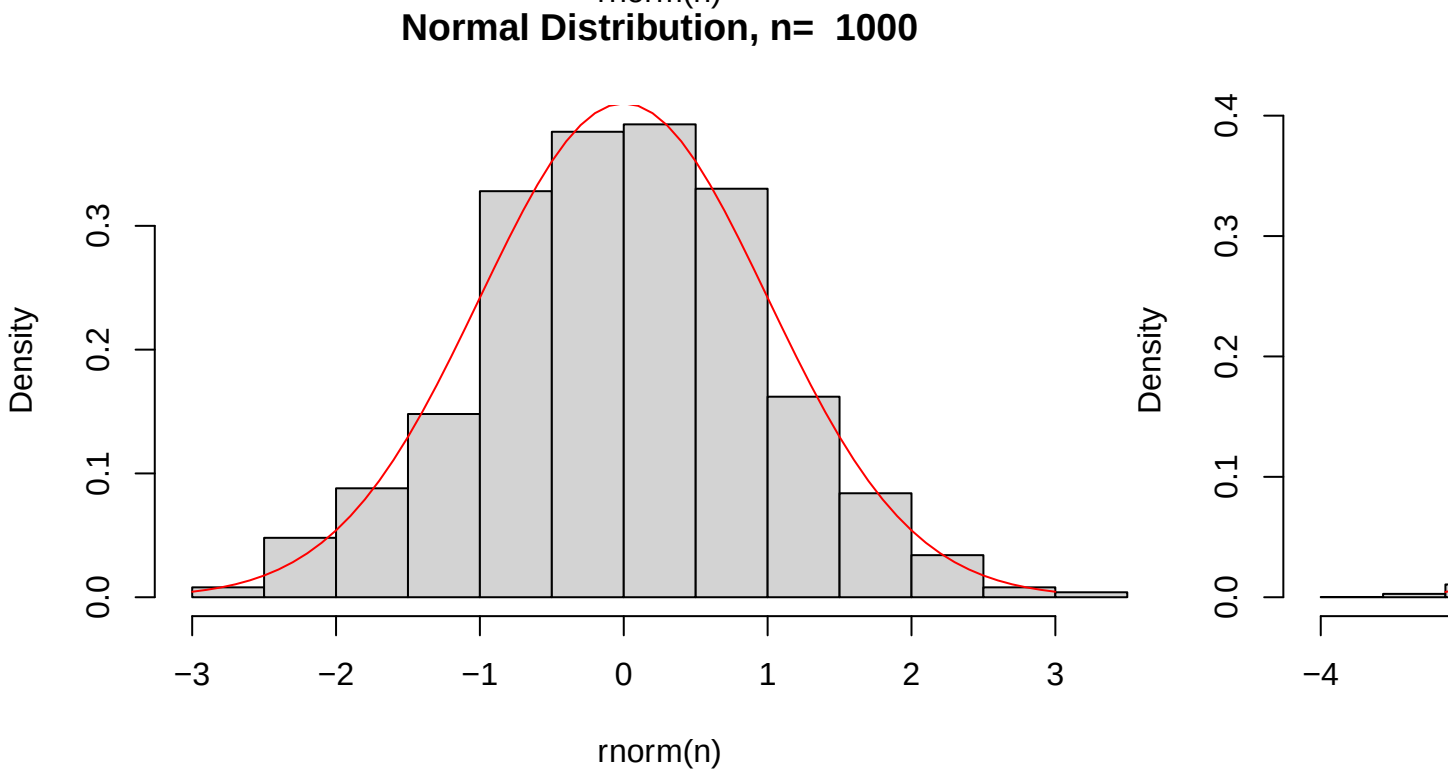
```
plot(xs, dnorm(xs), type='l', main='Standard Normal Distrubtion')
```



d) To overlay the theoretical distribution and the histogram in the same plot, simple place the calls to hist and lines one after the other in the same code block. *Do this plot: is the histogram similar to the theoretical distribution function? Change the value n you defined earlier to some larger value. How does the plot change?* Bear in mind that if you defined the variable n in a different code block, you will need to re-execute it before running the block containing the plotting code. Just to be sure, you can re-execute all the code blocks.

```
for (n in c(100,1000,10000)) {
   hist(rnorm(n), main=paste('Normal Distribution, n= ', n), freq=FALSE)
   lines(xs, dnorm(xs), type='l', col='red')
}
```





e) Re-evaluating the code blocks many times is cumbersome and error prone. It is way better to define a function that takes as a parameter the number of samples you want, and encapsulates all the sampling and plotting code. Such a function in R can be defined as follows

```
sample.and.plot.norm <- function(n) {
  samples <- rnorm(n)
```

```
  hist(samples, freq=FALSE)
  xs <- -30:30 / 10
  p <- dnorm(xs)
  lines(xs, p, col='red')
}
```

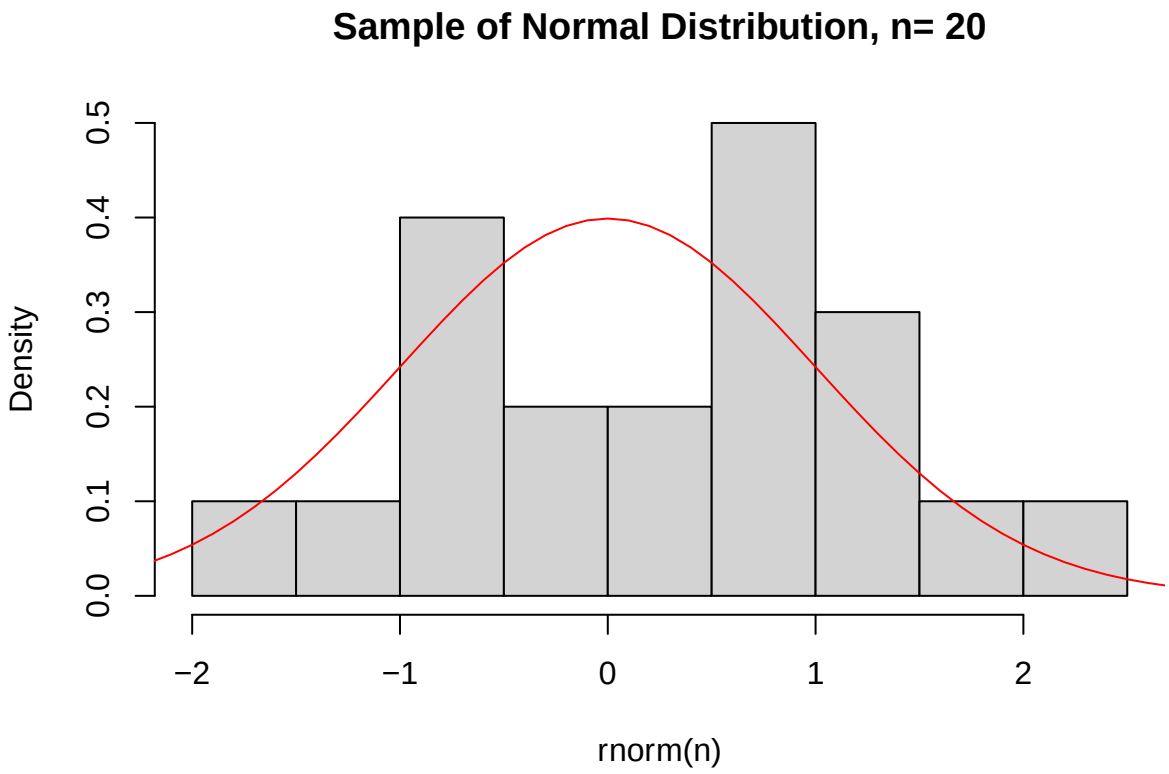
And then can be called, even in another code block

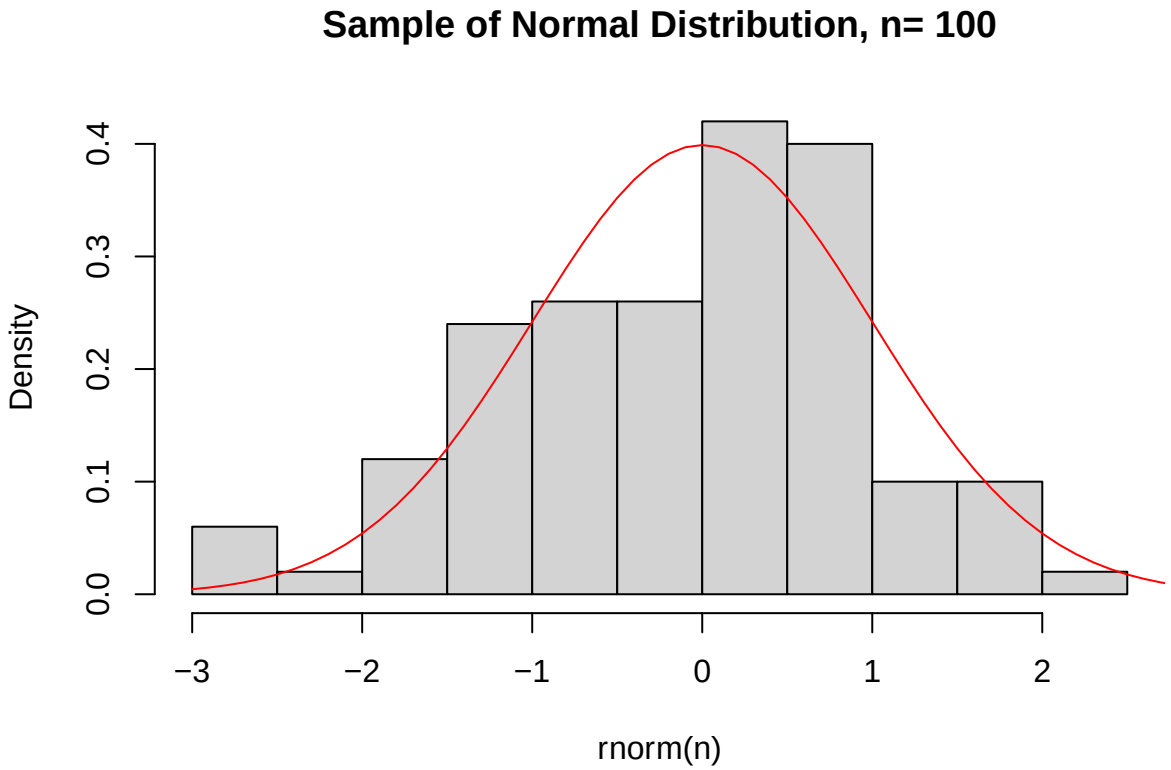
```
sample.and.plot.norm(100)
```

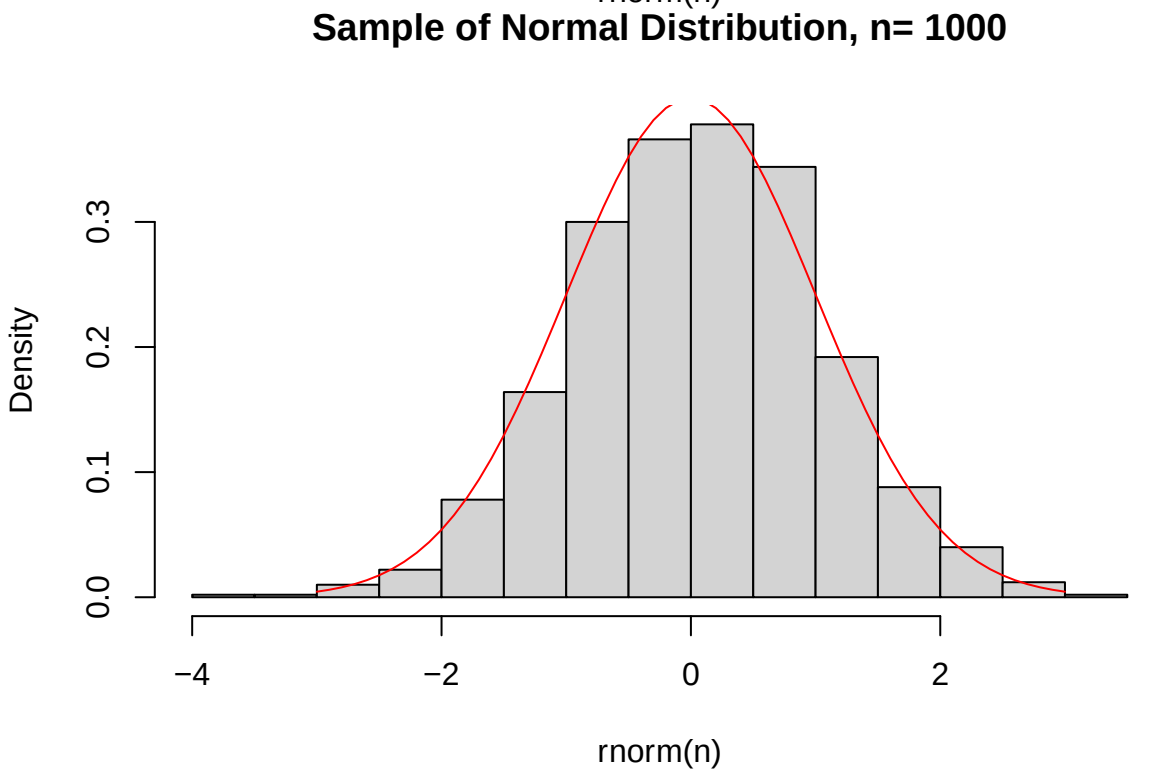
Define a function encapsulating your code, and call it several times with different parameters. Do you get plots similar to the ones you were getting before?

```
sample.and.plot.norm <- function(n) {
  hist(rnorm(n), freq=FALSE, main=paste('Sample of Normal Distribution, n=', n))
  lines(-30:30/10, dnorm(-30:30/10), col='red')
}
```

```
for (n in c(20,100,1000)){
  sample.and.plot.norm((n))
}
```







#5. (R) In this exercise, you will produce a plot similar to the previous exercise, but relative to the exponential distribution.

a) Write a function `sample.and.plot.exp` that takes the number of samples `n` as a parameter. In the body of the function, add code to

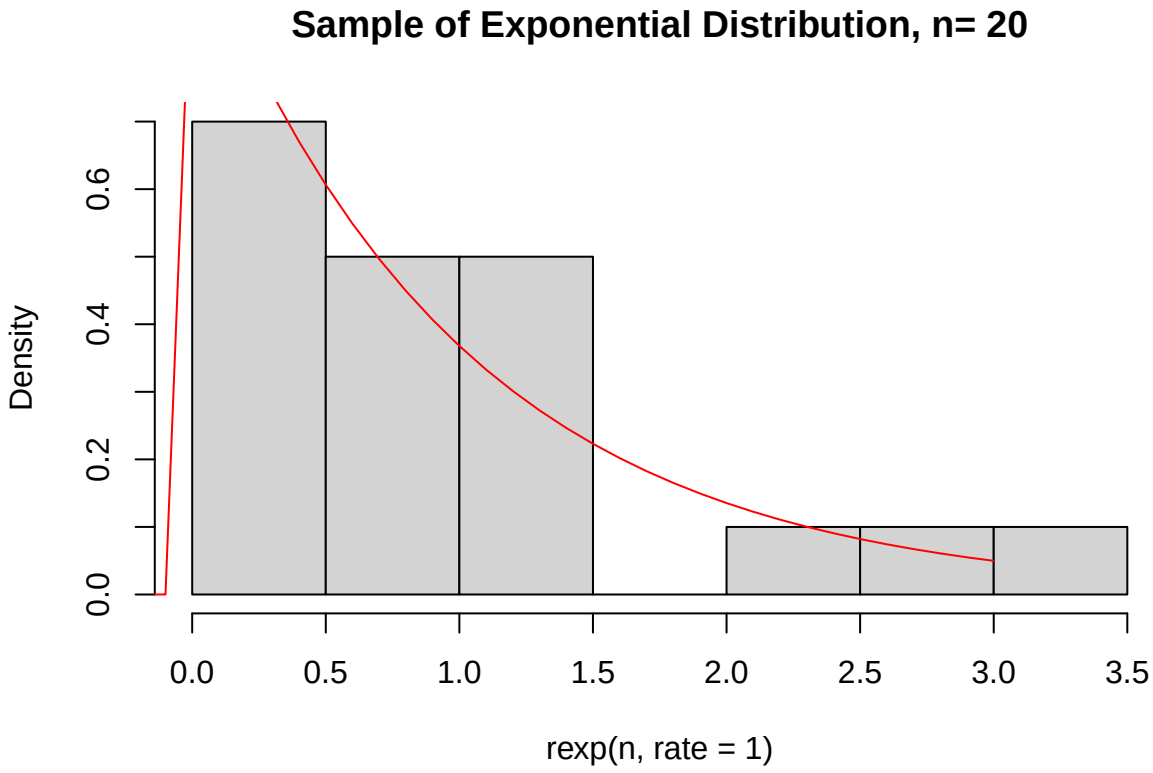
- Sample `n` numbers from the exponential distribution with parameter $\lambda = 1$.

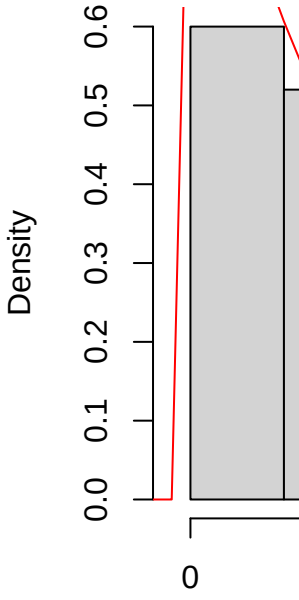
- Plot the histogram of the sampled numbers
- Overlay the plot of the exponential probability distribution function to the histogram.

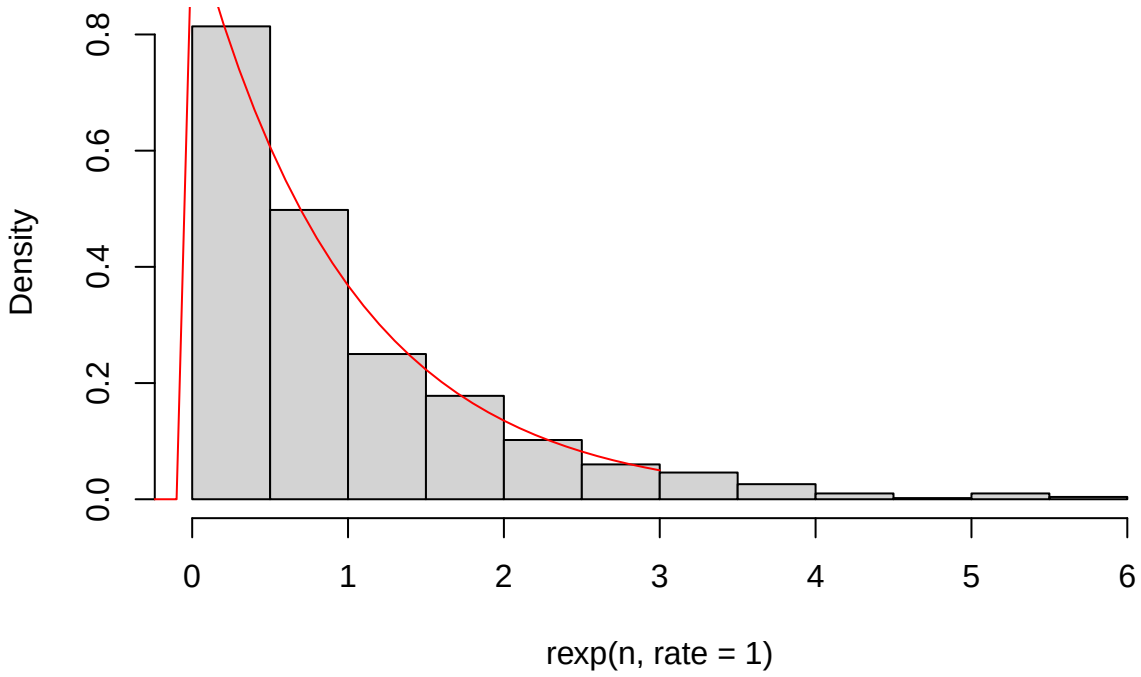
```
sample.and.plot.exp <- function(n) {
   hist(rexp(n, rate=1), freq=FALSE, main=paste('Sample of Exponential Distribution, n=', n))
   lines(-30:30/10, dexp(-30:30/10), col='red')
}
```

b) Call the function sample.and.plot.exp with several different values of n. How does it change the appearance of the plot?

```
for (n in c(20,100,1000)){
   sample.and.plot.exp((n))
}
```







#6. (R) Let X be a continuous random variable with the probability density function

$$f(x) = \begin{cases} 2x & \text{if } 0 \leq x \leq 1 \\ 0 & \text{otherwise.} \end{cases}$$

a) Derive the distribution function of $f(x)$.

$$f(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x^2 & \text{if } 0 \leq x \leq 1 \\ 1 & \text{if x} > 1 \end{cases}$$

b) Describe the principle how you can simulate the random variables following the probability density $f(x)$ when you have an access to a random number generator of uniformly distributed random numbers.

We can draw a random sample from any (continuous) distribution using $u \in U(0,1)$ by $x = F^{inv}(u)$. Here x is a realisation of the random variable X with a cumulative distribution function $F(x)$.

c) Write a program that uses stochastic simulation to draw 100 independent samples from $f(x)$.

```
draw.sample <- function() {
   u <- runif(1, min=0, max=1) # random sample from U(0,1)
   return(sqrt(u)) # the inverse of x^2 is the root of x
}
```

```
draw.sample()
```

```
## [1] 0.6899707
```

```
plot.realisation <- function(n) {
   samples <- c()
   for (i in 1:n) {
      samples <- append(draw.sample(), samples)
```

```
  }
  hist(samples, main=paste('Realisation of f(x), n=', n))
}
```

```
plot.realisation(1000)
```