STATS 406 Fall 2016: Lab 02

1 Density function of normal distributions

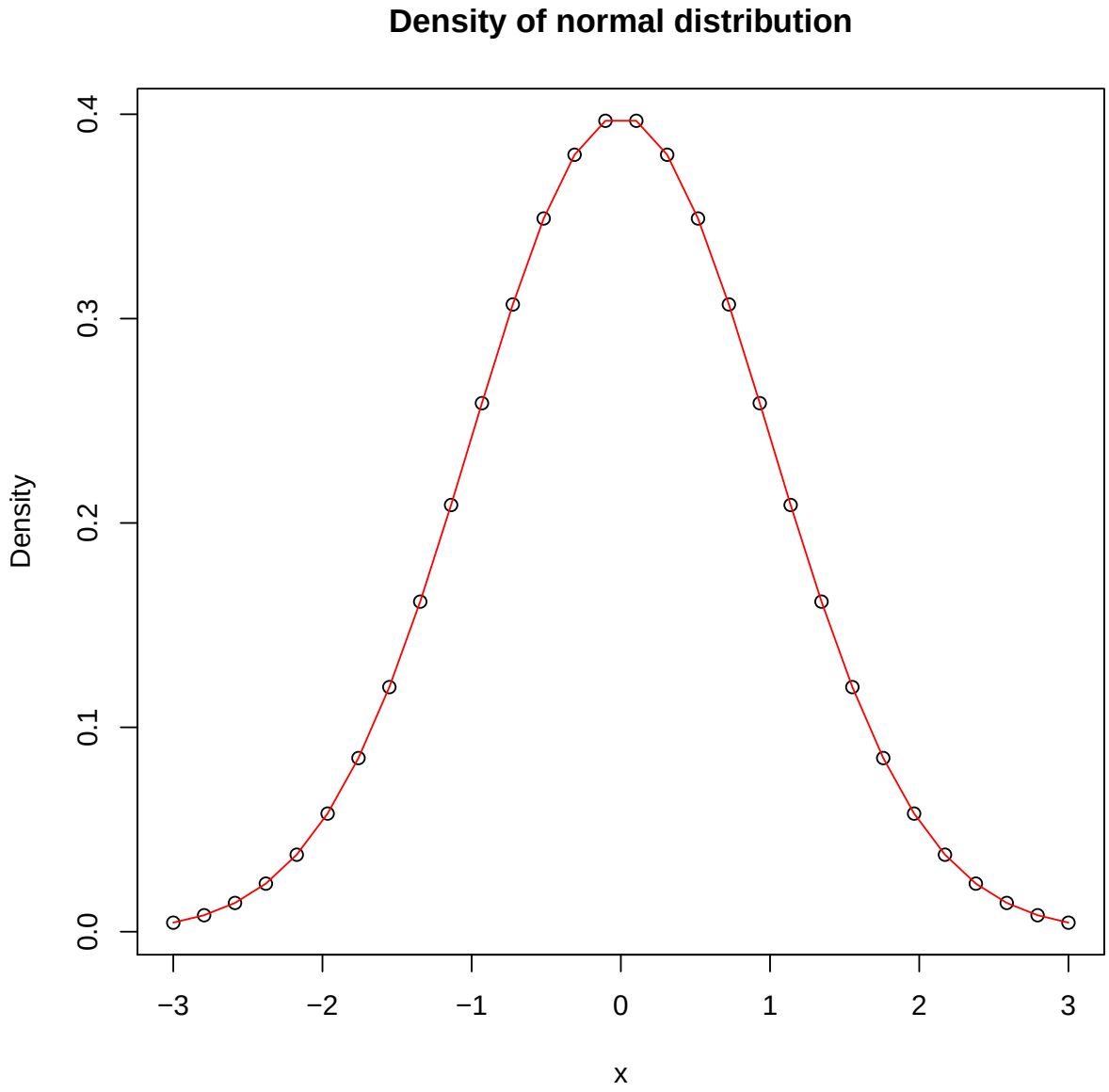
In R, functions are a symbolic representation of an operation to be carried out on variables known as inputs. Functions are useful when you plan to apply a certain operation to a range of inputs which are cumbersome to be declared beforehand. The syntax for declaring a function can be best understood with an example:

The probability density of a normal distribution $N(\mu, \sigma^2)$ is defined as

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\{-\frac{(x-\mu)^2}{2\sigma^2}\}.$$

There is a function `dnorm()` in R that can produce values of this density. Now without using that function, write your own function to calculate $f(x)$. Then plot the densities of $N(0, 1)$ from your own function on a grid of 30 points from -3 to 3, and compare with the result using the internal function `dnorm()`.

```
normal_density_function <- function(x, mean, sd){
  d = 1 / sqrt(2*pi*sd^2) * exp(-(x - mean)^2 / (2*sd^2))
  return(d)
}
# Generate a grid with 30 points between -3 and 3
x = seq(-3, 3, length=30)
# Compute the corresponding densities
y1 = normal_density_function(x=x, mean=0, sd=1)
# Compute the densities from R internal function dnorm()
y2 = dnorm(x=x, mean=0, sd=1)
# Plot the result of function normal_density_function() with points
plot(x, y1, main='Density of normal distribution',
xlab='x', ylab='Density', type='p')
# Plot the result of function normal_density_function() with curve
lines(x, y2, lty=1, col="red")
```



The return statement within a function is used when you want some value returned by the function. Sometimes you just want a function to do something (say print/plot something), but not return a value. There is no need to use the `return()` command in those situations. Also, the variables used in a function are local and not accessible unless you are returning them as output.

2 Coin flipping game

In a fair coin flipping game, let $B_1, B_2, \ldots$ denote the result where $B_i = 1$ if it's heads and 0 if it's tails. Let N be the number of experiments when the first time you get heads, i.e.

$$N = \min\{n \geq 1 : B_n = 1\}.$$

Suppose you are given a number m, write a piece of R code that runs the game for m times and prints $N_1, \ldots, N_m$.
Hint: you can use the command `rbinom(1,1,0.5)` to simulate the result of flipping a coin. The function `rbinom(n, size, prob)` generates binomial random variable.

```
m = 10
for(k in 1:m){
  # initialize count n and result B
  n = 0
  B = 0
  # stop at the first time that 1 occurs
  while(B != 1){
    n = n + 1
    # generate a fair coin flip
    B = rbinom(1,1,0.5)
  }
  print(n)
}
```

3 Another coin flipping game

In a fair coin flipping game, two players A and B bet on the first two consecutive results. Specifically, if the first two consecutive are heads, then A wins. If the first two consecutive are tails, then B wins. Suppose you are given a number m, write a piece of R code that simulates the game for m times and prints the winner at each time. And calculate the proportion of times that A wins.

```
m = 100
# save the times that A wins
count = 0
for(k in 1:m){
  # generates the first flip
```

```
  coin_old = rbinom(1,1,0.5)
  # generates the second flip
  coin = rbinom(1,1,0.5)
  # stops when two consecutive results appear
  while(coin_old != coin){
    # save the last flip
    coin_old = coin
    # generate a new flip
    coin = rbinom(1,1,0.5)
  }
  # check the results
  if(coin==1){
    print("A wins")
    count = count + 1
  }else{
    print("B wins")
  }
}
# print the proportion of times A wins
print(count/m)
```

4 Student performance data set

Download the "student-mat.csv" file from Canvas. This is a data set about students' math grades and other information. Make sure that the data is located in your current working directory in R. You can use the command `getwd()` and `setwd()` to check and change the directory.

1. Read the data into R by using command

   `read.table("student-mat.csv",header=T,sep=";")`.

   Check the help document in R to see what the parameters mean.

2. Look at the variables `G1, G2, G3` which denote the first period, second period, and final math grades of the students. Create a new data frame consisting of only two columns: the average of the three grades; the weighted average of the three grades with weights 0.25, 0.25, 0.5. You can use either for loops or apply to solve this.

3. Save the new data frame into a csv file "student_new.csv".

```r
# read the data
student = read.table("student-mat.csv",header=T,sep=";")
n = nrow(student)
# create a new data frame and specify the column names
student_new = data.frame(matrix(0,nrow=n,ncol=2))
names(student_new) = c("avg","weightedAvg")

# use for loop
for(i in 1:n){
  tmp = as.numeric(student[i,c("G1","G2","G3")])
  student_new$avg[i] = mean(tmp)
  student_new$weightedAvg[i] = sum(tmp*c(0.25,0.25,0.5))
}

# use apply
student_new$avg = apply(student[,c("G1","G2","G3")],1,mean)
# define the weighted sum function
student_new$weightedAvg = apply(student[,c("G1","G2","G3")],1,
function(x) {sum(x*c(0.25,0.25,0.5))})

# write the new data frame into csv file
write.table(student_new,file="student_new.csv",sep=",",
col.names = T,row.names = F)
```