Exercises

SCR week 5

Exercises part 1

1.1. Density plots

Read paragraph 12.1.4 of the book of Norman, pp. 264 - 265. In this exercise you will create the graph shown on page 265, but using a different dataset, called "tobit.csv" (see `0_data/tobit.csv`). This data set contains test scores of 200 children. The variables you will use in this exercise are called `read` and `math`.

a

Read in the `tobit` dataset.

Answer:

```
testscores <- read.csv("0_data/tobit.csv")
```

b

Inspect the data set by using a few `R` functions that explore the structure of the data, or can give a summary of the data.

Answer:

```
str(testscores)
```

```
## 'data.frame':    200 obs. of  6 variables:
##  $ X   : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ id  : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ read: int  34 39 63 44 47 47 57 39 48 47 ...
##  $ math: int  40 33 48 41 43 46 59 52 52 49 ...
##  $ prog: Factor w/ 3 levels "academic","general",..: 3 3 2 2 2 2 2 2 3 1 ...
##  $ apt : int  352 449 648 501 762 658 800 613 531 528 ...
```

```
summary(testscores) #inspect the data set
```

```
##        X                id              read            math      
##  Min.   :  1.00   Min.   :  1.00   Min.   :28.00   Min.   :33.00  
##  1st Qu.: 50.75   1st Qu.: 50.75   1st Qu.:44.00   1st Qu.:45.00  
##  Median :100.50   Median :100.50   Median :50.00   Median :52.00  
##  Mean   :100.50   Mean   :100.50   Mean   :52.23   Mean   :52.65  
##  3rd Qu.:150.25   3rd Qu.:150.25   3rd Qu.:60.00   3rd Qu.:59.00  
##  Max.   :200.00   Max.   :200.00   Max.   :76.00   Max.   :75.00  
##          prog          apt        
##  academic  : 45   Min.   :352.0  
```

```
##  general   :105   1st Qu.:575.5
##  vocational: 50   Median :633.0
##                   Mean   :640.0
##                   3rd Qu.:705.2
##                   Max.   :800.0
```

c

Like in chapter 12.1.4 of Matloff, use `density()` to get two seperate nonparametric density estimates of the `read` and `math` scores. Make sure you set the `from` and `to` arguments correctly. Use the information you got from **b** to set these. E.g.: if scores run, say from 28 to 76, you might want to have the density run from 20 to 84.

Answer:

```
d_read <- density(testscores$read, from = 20, to = 84, bw = "SJ")
d_math <- density(testscores$math, from = 20, to = 84, bw = "SJ")
# the from and to of the d_read will, in our example determine the plotting window,
# so the range for d_math can be set to the same.
# We can check with summary() or range() if all the scores fall within this range.
```

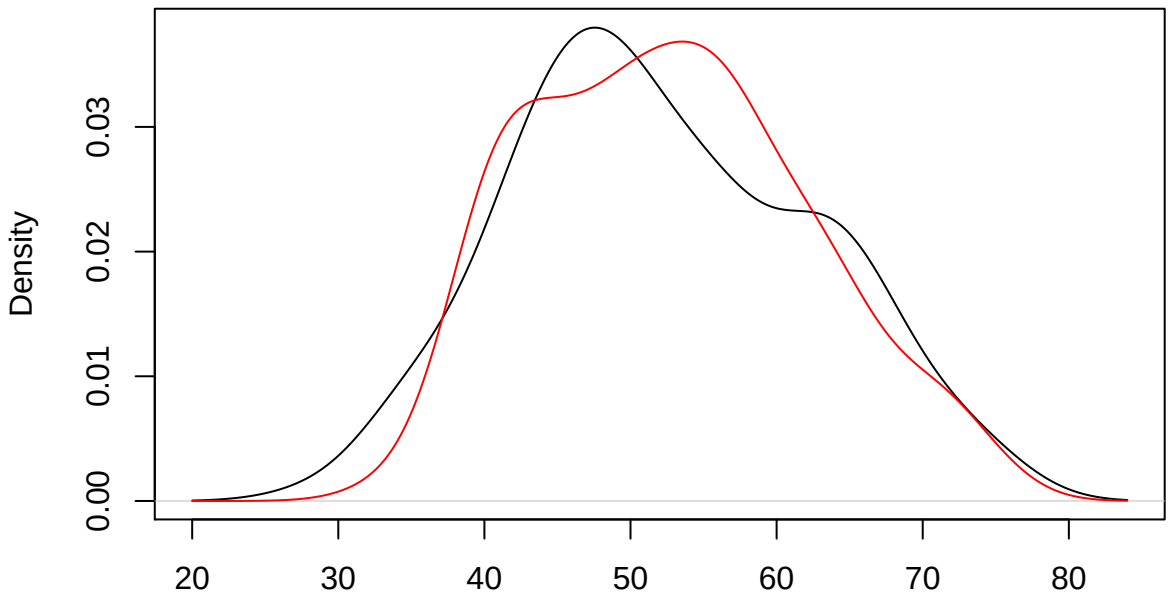
d

Put both densities into a single plot, make sure you give the different density estimates a different colour. Any interesting differences?

Answer:

```
plot(d_read, main = "", xlab = "")
lines(d_math, col = 'red')
```

Or : par(new=TRUE)



It looks as though both might be bi-modal. But that's hard to tell from these estimates.

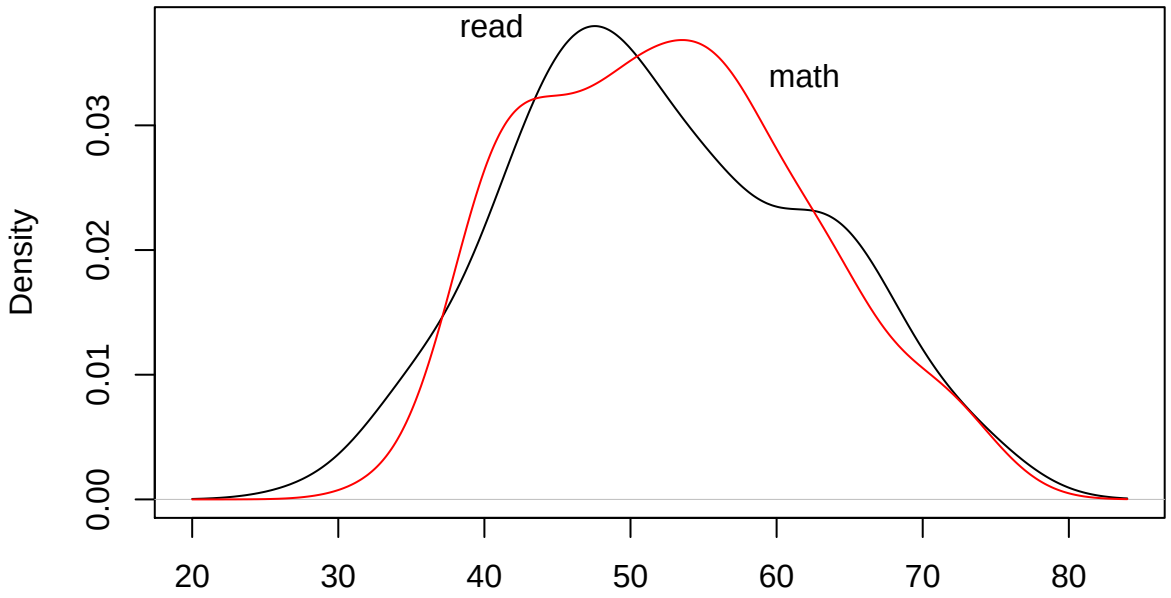
1.2 Adding text to a graph with help of the `locator()` function

a

We are going to add text to the graph created in the previous exercise. Read par. 12.1.8 and 12.1.9 of the book of Matloff, pp. 270 - 271. Instead of the text "Exam 1" and "Exam 2", we use now `read` and `math` from the `tobit` data used in the previous exercise. To determine the coordinates of the text, use the function `locator()`. In your console, type `locator(2)` and use enter. Then, go to the graph you created in Exercise 2 and click on the two points in the graph where you want to put the text. Use these coordinates in the function `text()`.

Answer:

```
#locator(2)
plot(d_read, main = "", xlab = "")
lines(d_math, col = 'red')
text(40.5,  0.038, "read")
text(61.9, 0.034, "math")
```

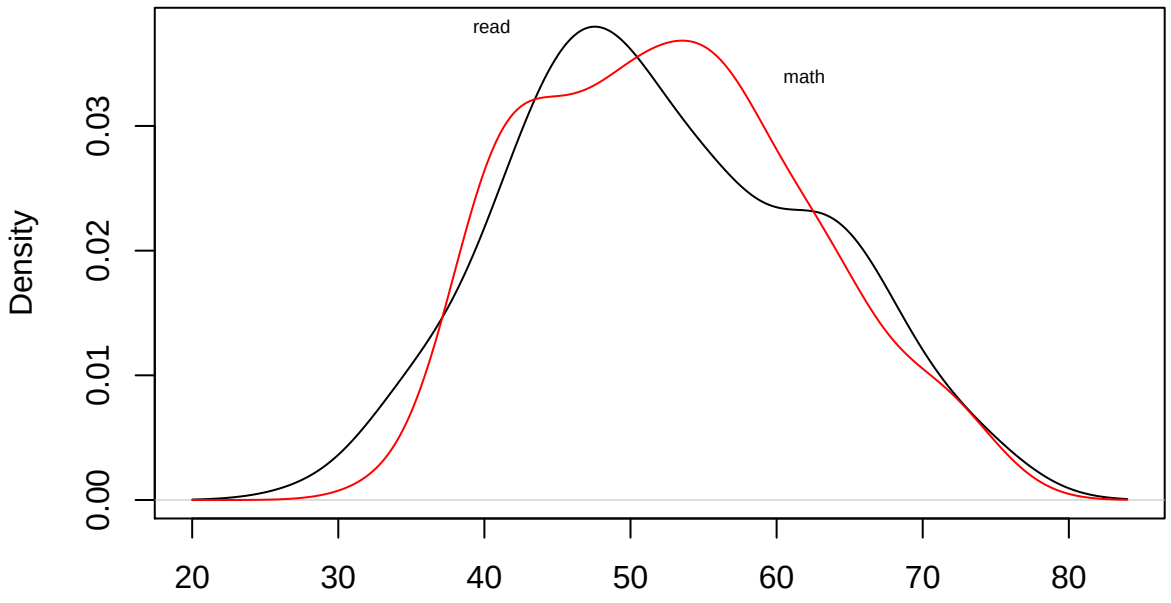


b

Change the size of the text by varying the `cex` parameter (See also par. 12.2.1 on p. 272 of the book of Matloff).

Answer:

```
plot(d_read, main = "", xlab = "")
lines(d_math, col = 'red')
text(40.5,  0.038, "read", cex = 0.6)
text(61.9, 0.034, "math", cex = 0.6)
```



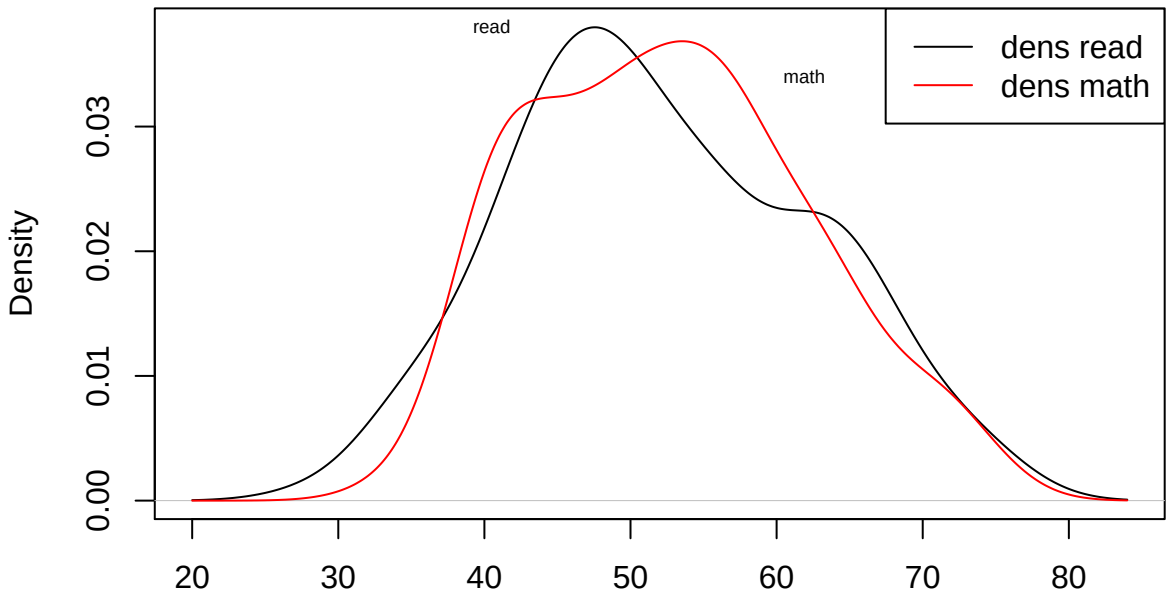
c

Add a legend to the plot using `legend()`. Put it in the topright corner of the plot.

Answer:

The location may also be specified by setting x to a single keyword from the list "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right" and "center".

```
plot(d_read, main = "", xlab = "")
lines(d_math, col = 'red')
text(40.5,  0.038, "read", cex = 0.6)
text(61.9, 0.034, "math", cex = 0.6)
legend(x = "topright",
       legend = c("dens read", "dens math"),
       lty = 1, col =  c('black', 'red')
)
```



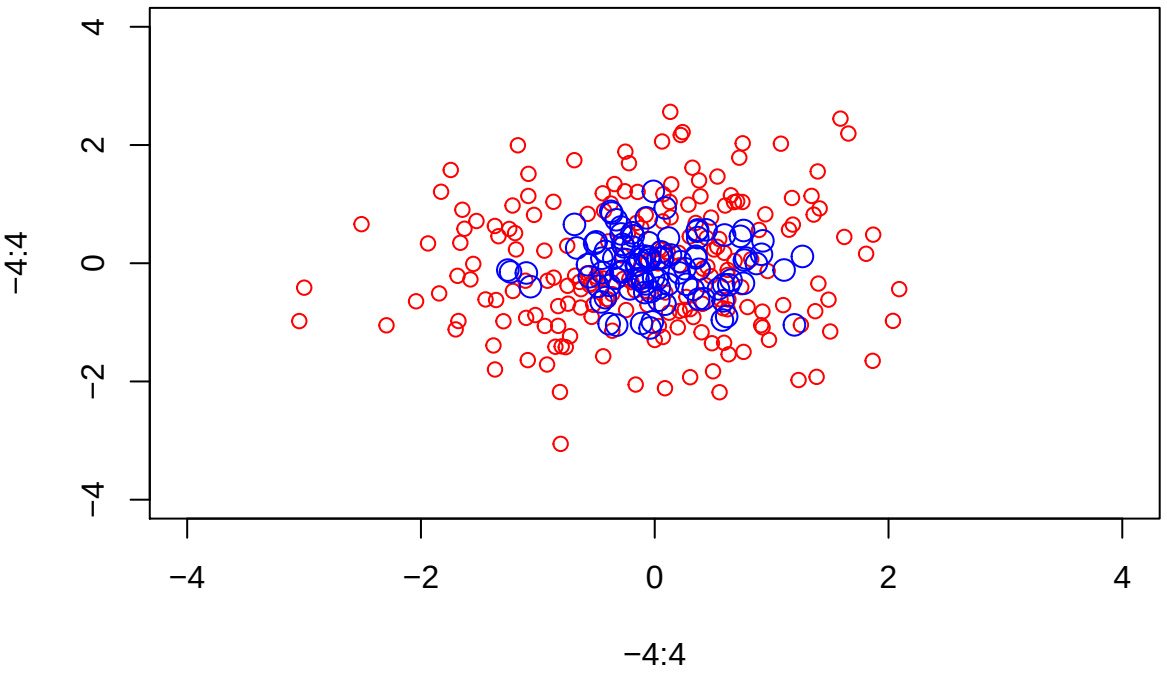
1.3 Adding points to a graph and customizing a graph

Let's start from scratch!

a

We are going to add points to an existing, but empty, graph with the function `points()`. Look at the following code. Take special care to try to see what the second line does. Play around with some of the arguments if you're not sure what's going on.

```
set.seed(99)
plot(-4:4, -4:4, type = "n")  # setting up coord. system
points(rnorm(200), rnorm(200), col = "red")
points(rnorm(100)/2, rnorm(100)/2, col = "blue", cex = 1.5)
```



Answer:

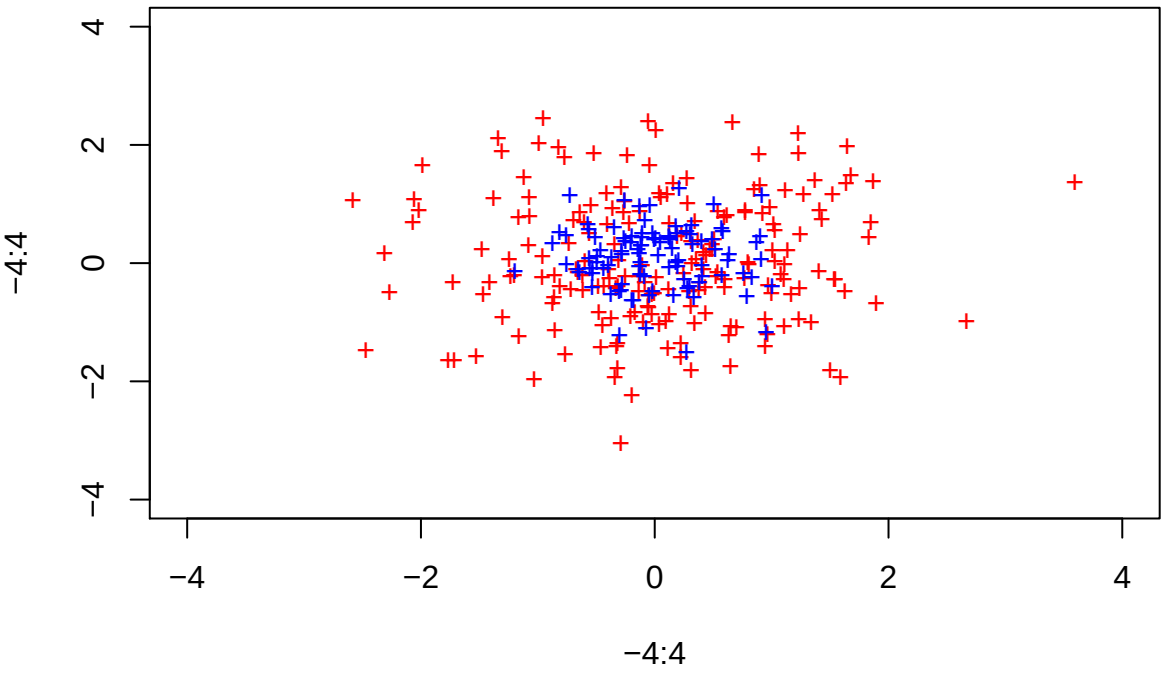
The second line plots 4 points, but we told `R` to not show them (see the `type` argument).

b

Change the symbol of the points used in the graph created in **a**, to a `+`, by adapting the parameter `pch` (see e.g. the helpfile of `points`).

Answer:

```
plot(-4:4, -4:4, type = "n")  # setting up coord. system
points(rnorm(200), rnorm(200), pch = "+", col = "red")
points(rnorm(100)/2, rnorm(100)/2, pch = "+", col = "blue")
```

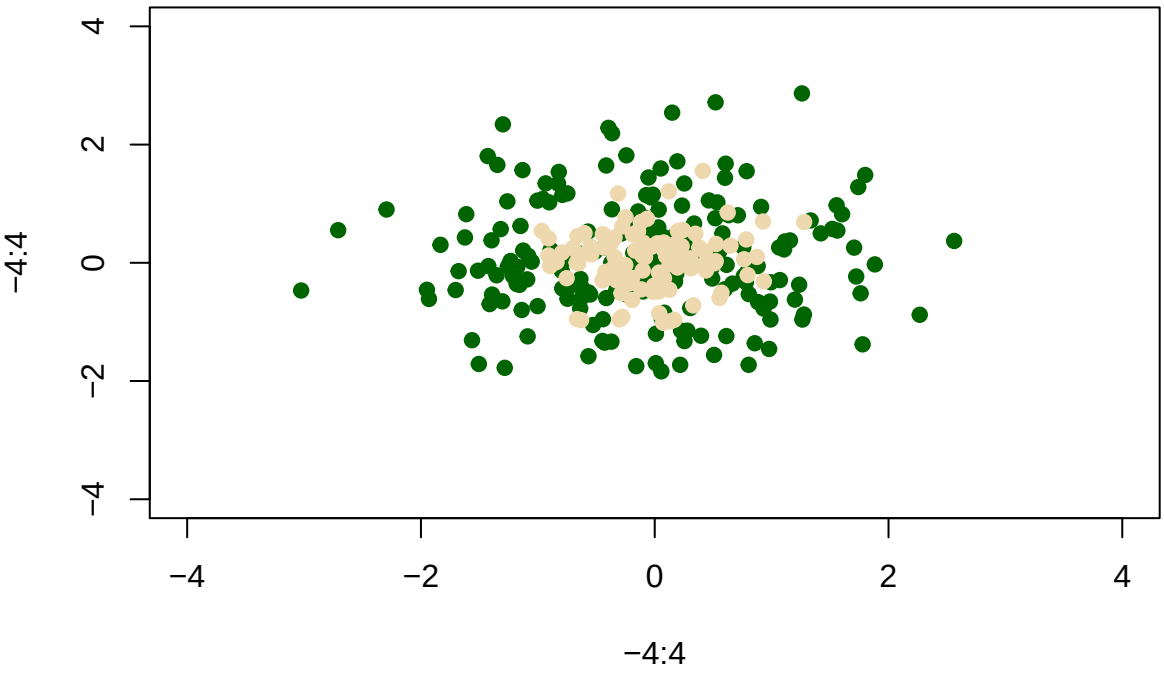


c

Change the color of the points, by adapting the parameter `col`. Use `colors()` to inspect which colors are available in `R`. Go nuts with colours by using e.g. `rainbow()`.

Answer:

```
plot(-4:4, -4:4, type = "n")  # setting up coord. system
points(rnorm(200), rnorm(200), pch = 19, col = "darkgreen")
points(rnorm(100)/2, rnorm(100)/2, pch = 19, col = "wheat2")
```

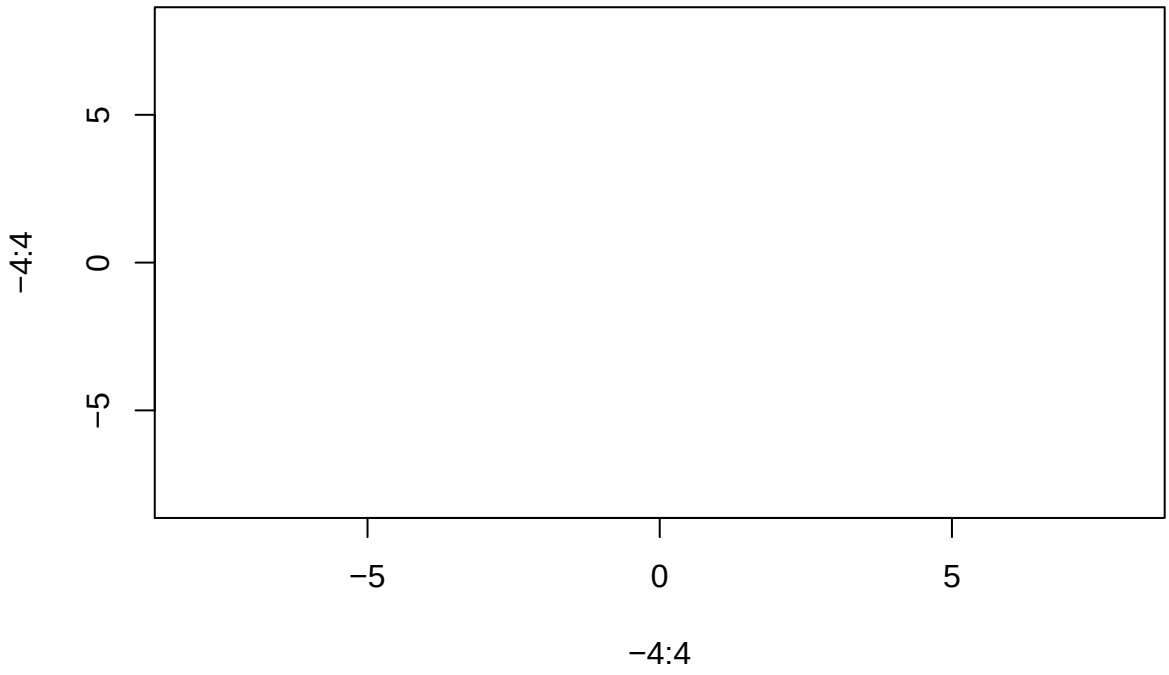


d

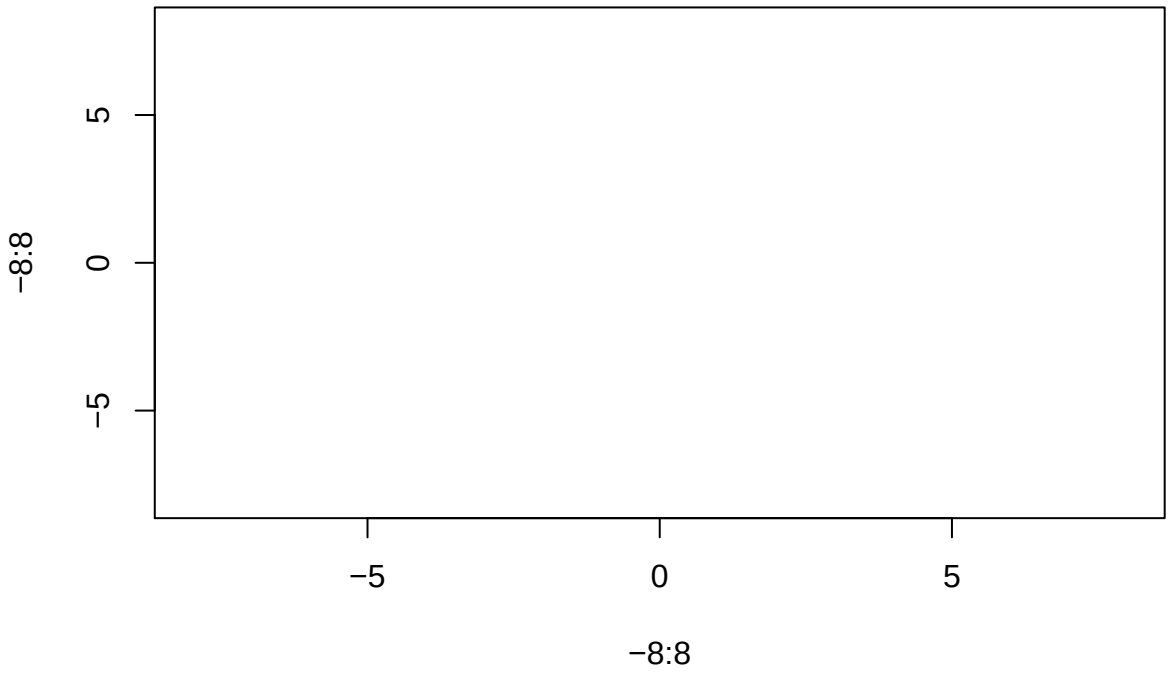
Change the range of the x-axis and y-axis (using `xlim` and `ylim`; see also paragraph 12.2.2, p. 273 of Matloff).

Answer:

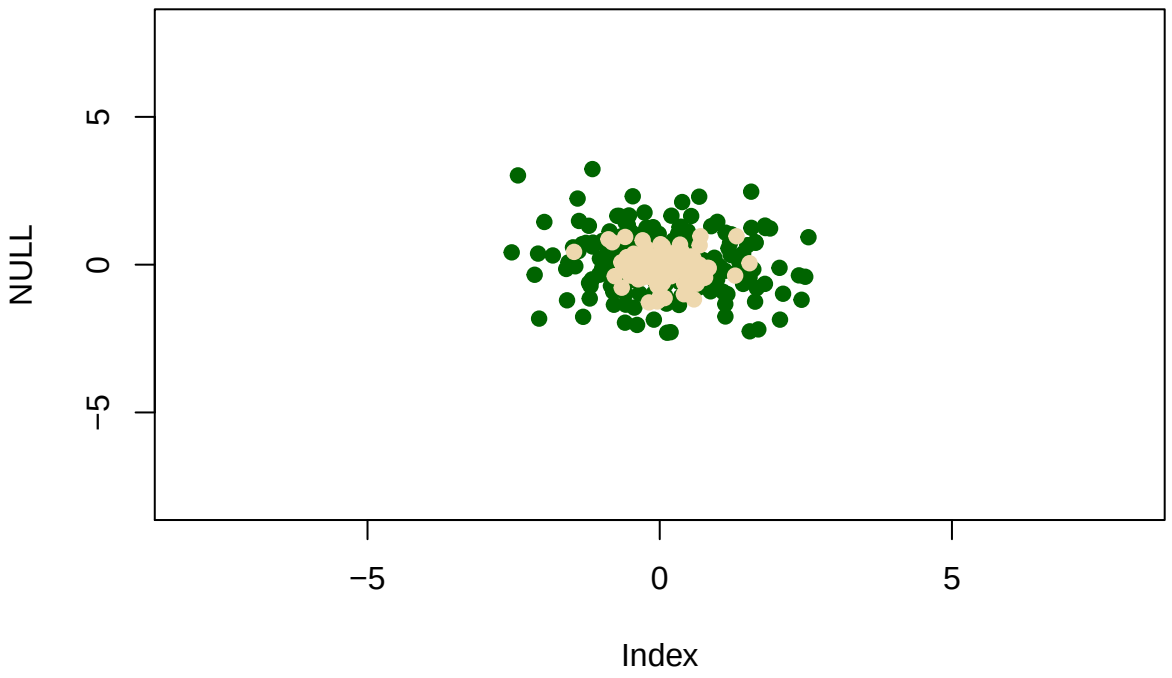
```
# the following three lines all work towards the same end,
# although in slightly different ways, all are correct.
plot(-4:4, -4:4, type = 'n', xlim = c(-8, 8), ylim = c(-8, 8))
```




```
plot(-8:8, -8:8, type = 'n')
```



```
plot(NULL,  xlim = c(-8, 8),  ylim = c(-8, 8))
points(rnorm(200), rnorm(200), pch = 19, col = "darkgreen")
points(rnorm(100)/2, rnorm(100)/2, pch = 19, col = "wheat2")
```

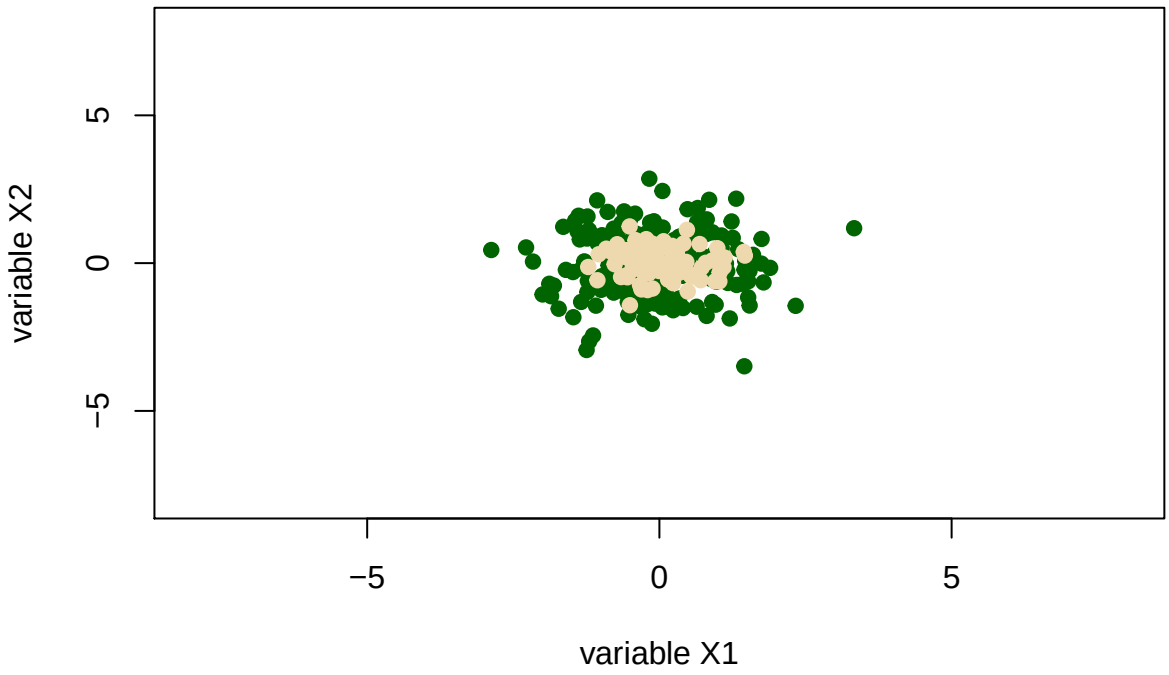


e

Change the labels of the x-axis and y-axis (using `xlab` and `ylab`) and create a title above the plot (using `main`).

Answer:

```
plot(NULL,  xlim = c(-8, 8),  ylim = c(-8, 8), xlab = "variable X1", ylab = "variable X2")
points(rnorm(200), rnorm(200), pch = 19, col = "darkgreen")
points(rnorm(100)/2, rnorm(100)/2, pch = 19, col = "wheat2")
```



1.4 Plotting the chi-square density function

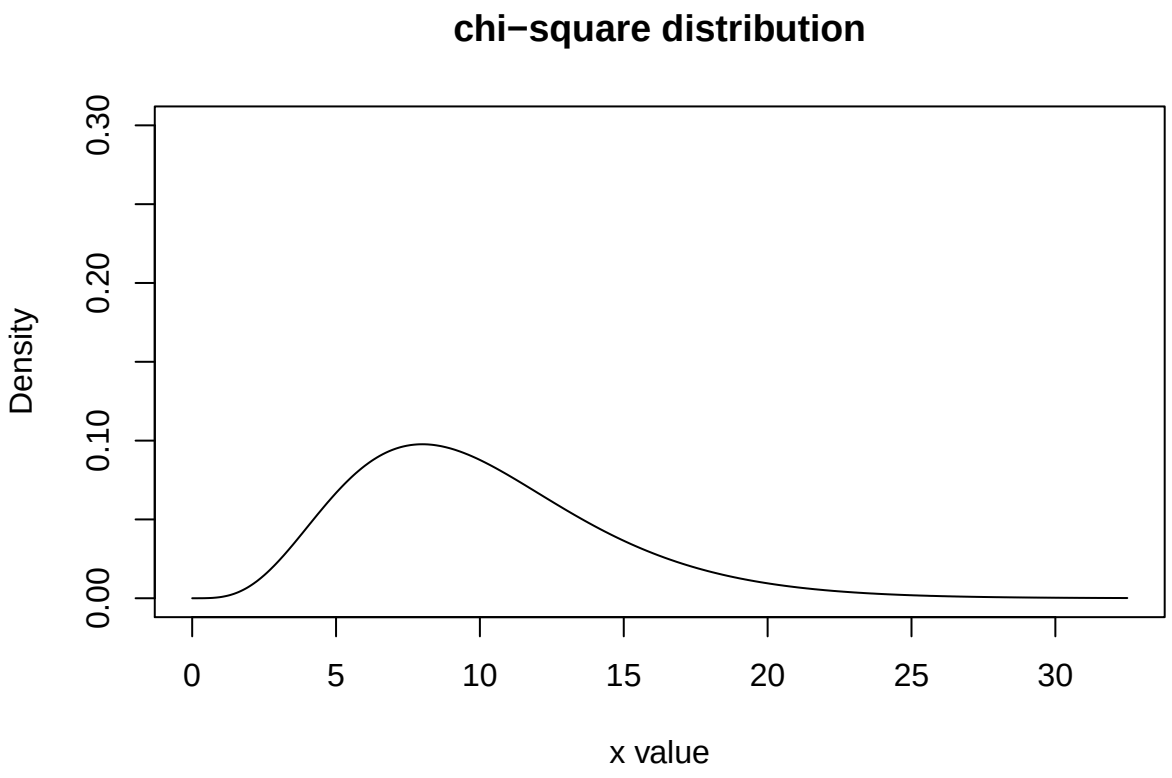
a

Create a graph of the chi-square density function with $df = 10$ (df = degrees of freedom). Make sure that you set the limits of your y-axis to 0 and 0.30 and the limits of the x-axis to 0 and 32.5.

Answer:

```
x_ax <- seq(0, 32.5, length = 1000)
plot(x_ax, dchisq(x_ax, df = 10),
  type = "l", xlab = "x value", ylab = "Density",
  ylim = c(0, 0.30), xlim = c(0, 32.5),
  main = "chi-square distribution"
)
```

d: density
p: prob (cdf)
q: inverse of p



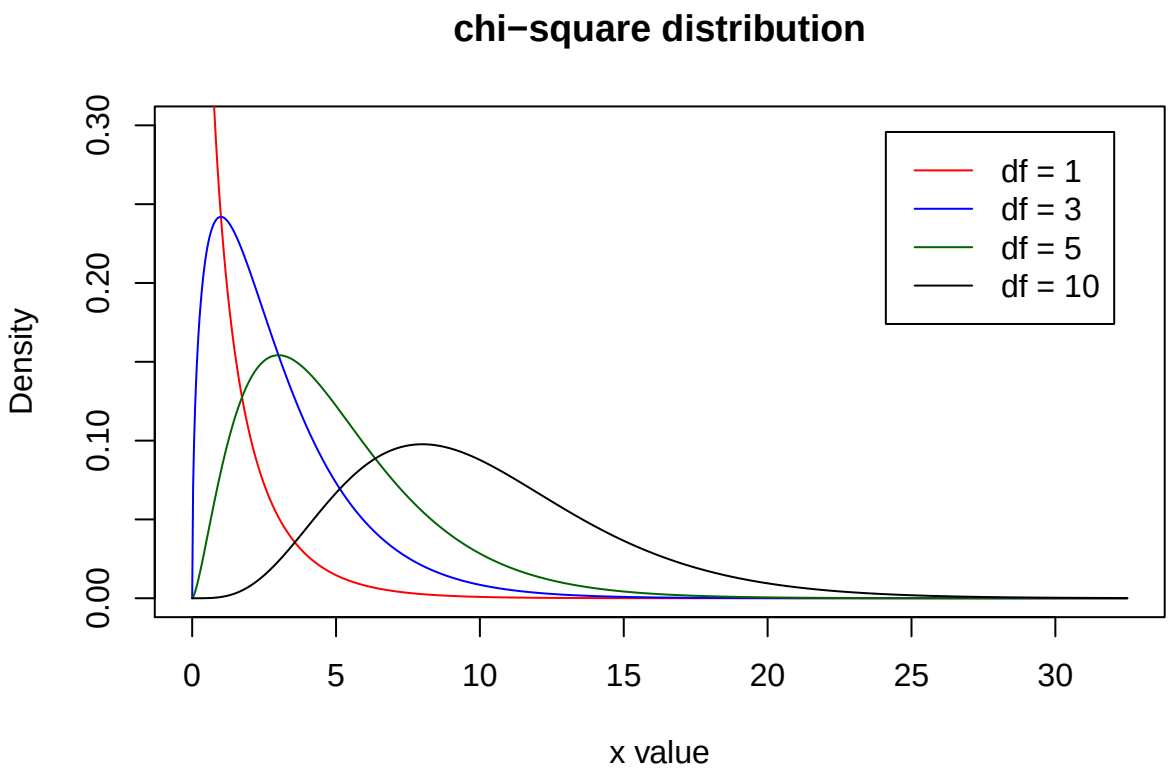
b

Use the function `lines()` to add more chi-square distributions, differing in number of degrees of freedom, to the graph created in a. Use $df = 1$, $df = 3$ and $df = 5$. Add a suitable legend to the graph. Think about the limits we made you set in **a**, why do you think we suggested these limits?

Answer:

```
plot(NULL, type = "l",
  xlab = "x value", ylab = "Density",
  ylim = c(0, 0.30), xlim = c(0, 32.5),
  main = "chi-square distribution"
)
degf <- c(1, 3, 5, 10)
colors <- c("red", "blue", "darkgreen", "black")
labels <- c("df = 1", "df = 3", "df = 5", "df = 10")
for (i in 1:length(degf)) {
  lines(x_ax, dchisq(x_ax, degf[i]), col = colors[i])
}
legend("topright", inset = .05, labels, lty = rep(1, 4), col = colors)
```

循环



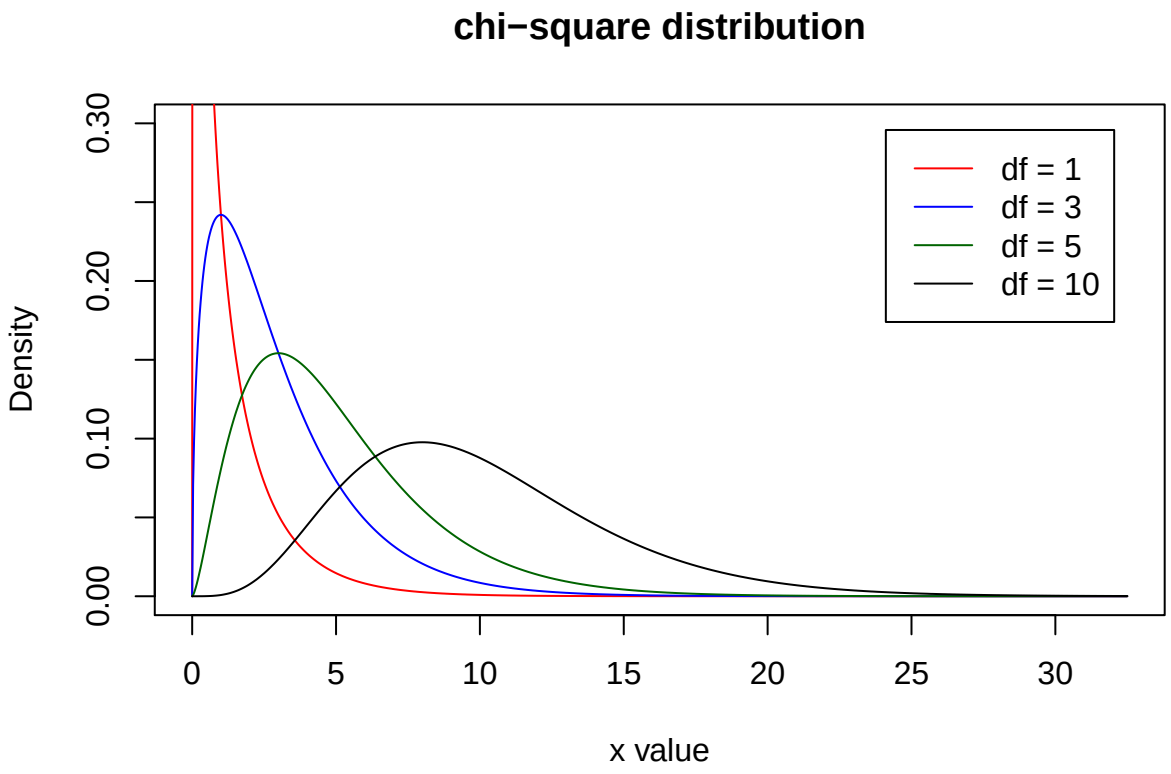
We suggested those limits, as the default limits of the first plot would mean some of the lines would be outside of the plotting window.

c

Look at the helpfile of the function `curve`. See if you can produce the plot in the previous exercise, using the `curve` function.

Answer:

```
#solution to a
plot(NULL, type = "l",
  xlab = "x value", ylab = "Density",
  ylim = c(0, 0.30), xlim = c(0, 32.5),
  main = "chi-square distribution"
)
#solution to b
degf <- c(1, 3, 5, 10)
colors <- c("red", "blue", "darkgreen", "black")
labels <- c("df = 1", "df = 3", "df = 5", "df = 10")

# this is where the answer differs:
mydchisq <- function(x, df) {
  return(dchisq(x, df))
}
for (i in 1:length(degf)) {
  curve(mydchisq(x, degf[i]), col = colors[i], add = TRUE, n = 1000)
}
legend("topright", inset = .05, labels, lty = rep(1, 4), col = colors)
```

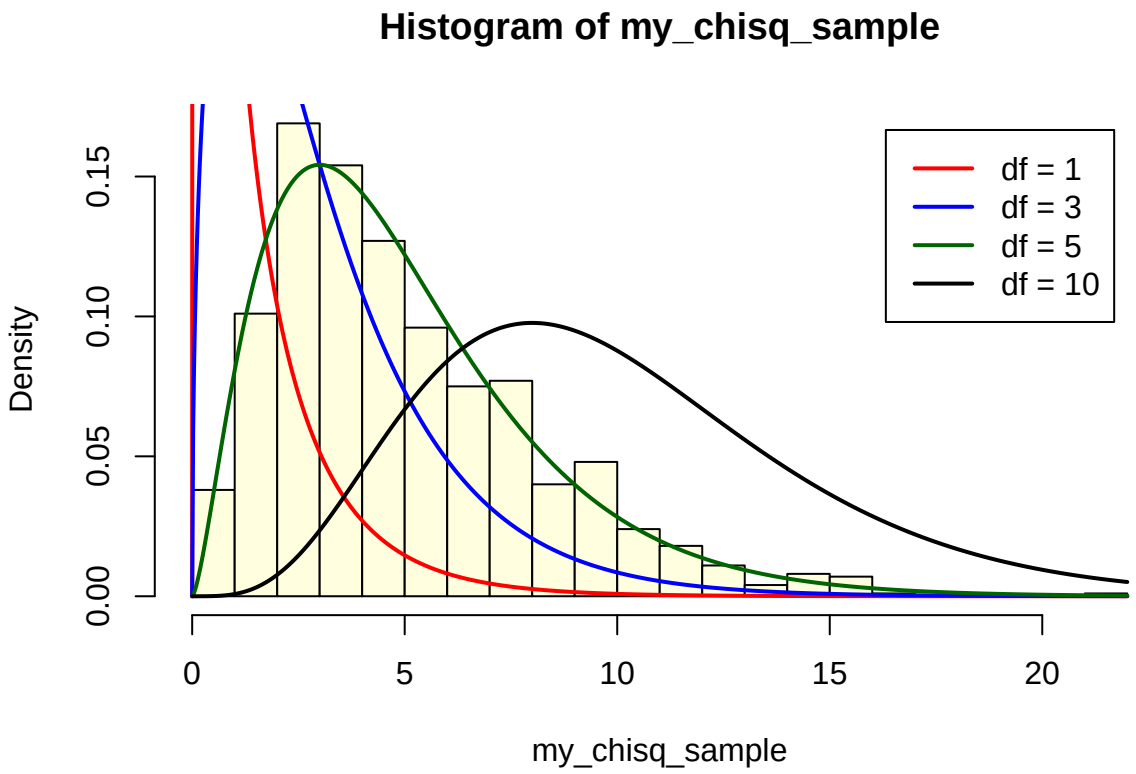


d

Sample a 1000 random values from a chisquare distribution with degrees of freedom equal to 3. Plot the results using a histogram. Use the code you've written in **b** (and/or **c**) to add the theoritical density lines to the histogram. Which theoretical density best matches the empirical distribution given by the histogram?

Answer:

```
my_chisq_sample <- rchisq(1000, df = 5)

hist(my_chisq_sample, breaks = "fd", freq = FALSE, col = 'lightyellow')
degf <- c(1, 3, 5, 10)
colors <- c("red", "blue", "darkgreen", "black")
labels <- c("df = 1", "df = 3", "df = 5", "df = 10")

for (i in 1:length(degf)) {
  curve(mydchisq(x, degf[i]), col = colors[i], add = TRUE, n = 1000, lwd = 2)
}
legend("topright", inset = .05, labels, lty = rep(1, 4), col = colors, lwd = 2)
```



1.5 Plotting some categorical data

Suppose we have the following variable:

```
observed_haircolours <- c("blonde" =  10, "brown" = 14, "black" = 3, "red" = 2)
```

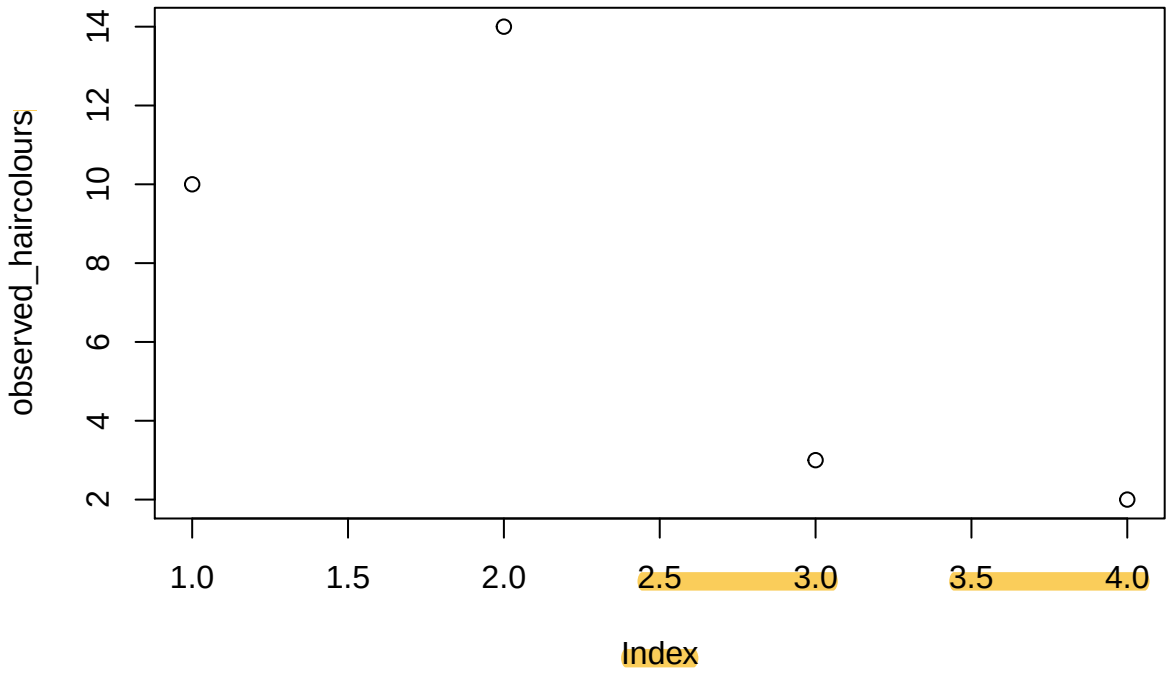
where each entry represents the number of times the specified haircolours were observed in a sample of 29 people.

a

Use `plot` to plot the observed haircolours. Is this plot any good?

Answer:

```
plot(observed_haircolours)
```



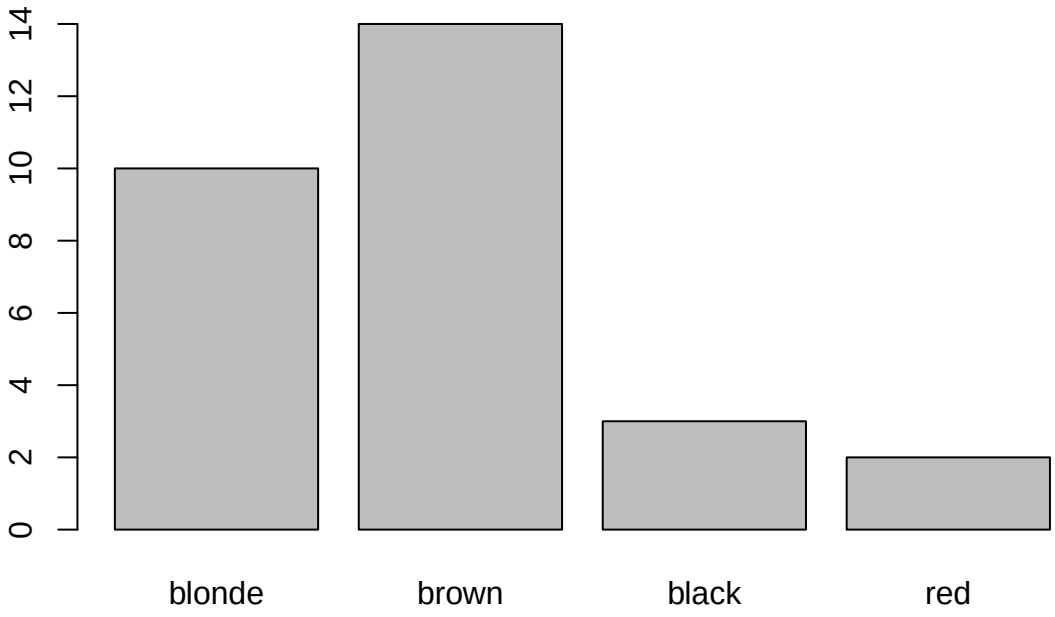
This isn't very useful.

b

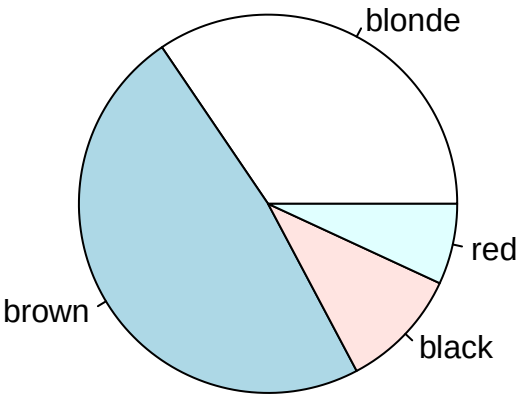
Now use `barplot` and `pie`. Are these any better?

Answer:

```
barplot(observed_haircolours)
```




```
pie(observed_haircolours)
```



Yes much better, these are graphs that tell us something interesting.

c

Convert the variable `observed_haircolours` to a table (recall e.g. the `as.numeric` function).

Answer:

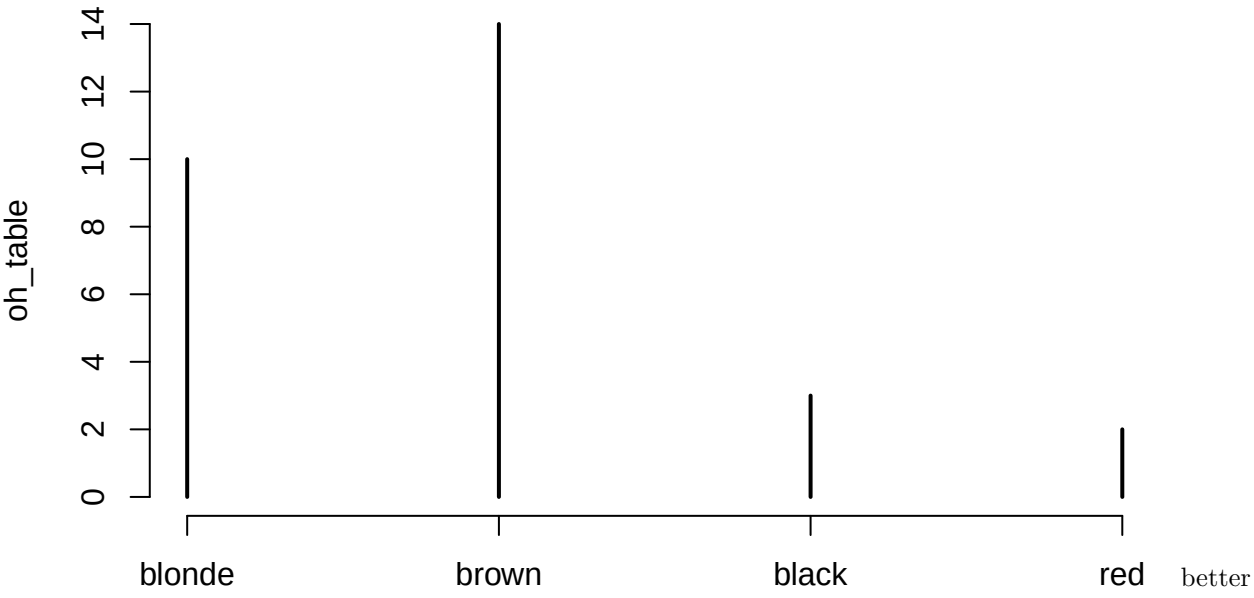
```
oh_table <- as.table(observed_haircolours)
```

d

Use `plot` on this object. What do you think of this plot?

Answer:

```
plot(oh_table)
```



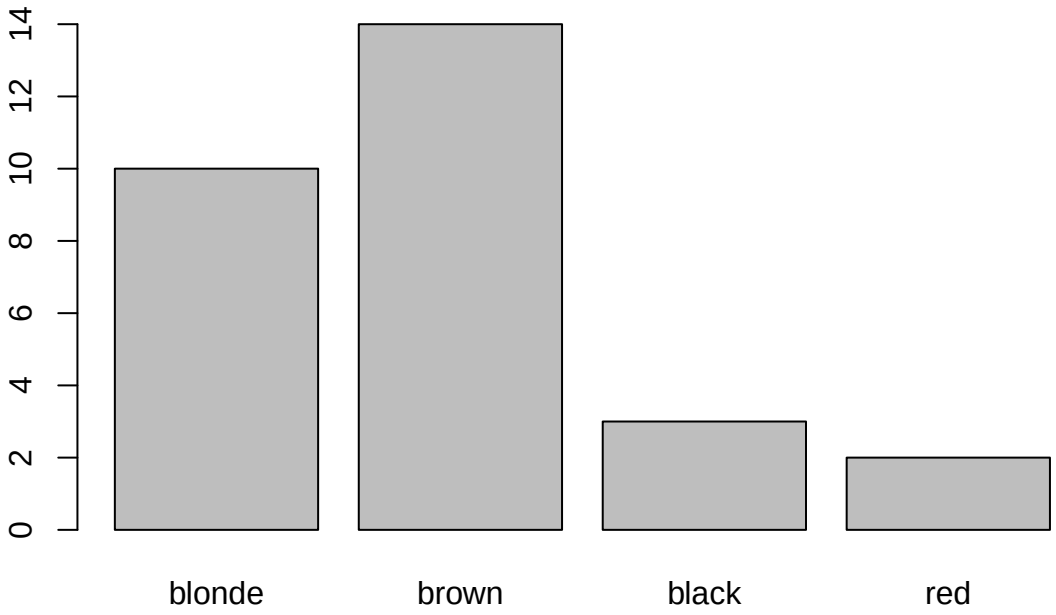
better than `plot` of the original, but still not very nice.

e

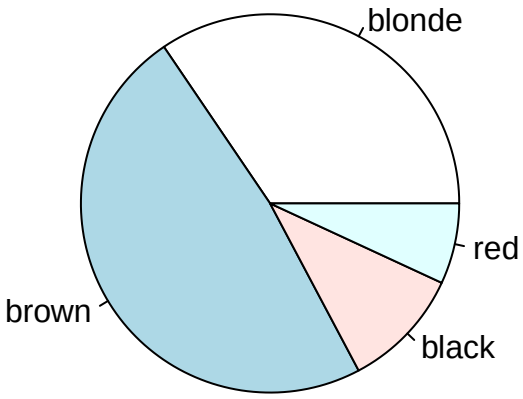
Use `barplot` and `pie` again, this time on your variable that's a table. Are they different from the ones you made in **b**?

Answer:

```
barplot(oh_table)
```




```
pie(oh_table)
```



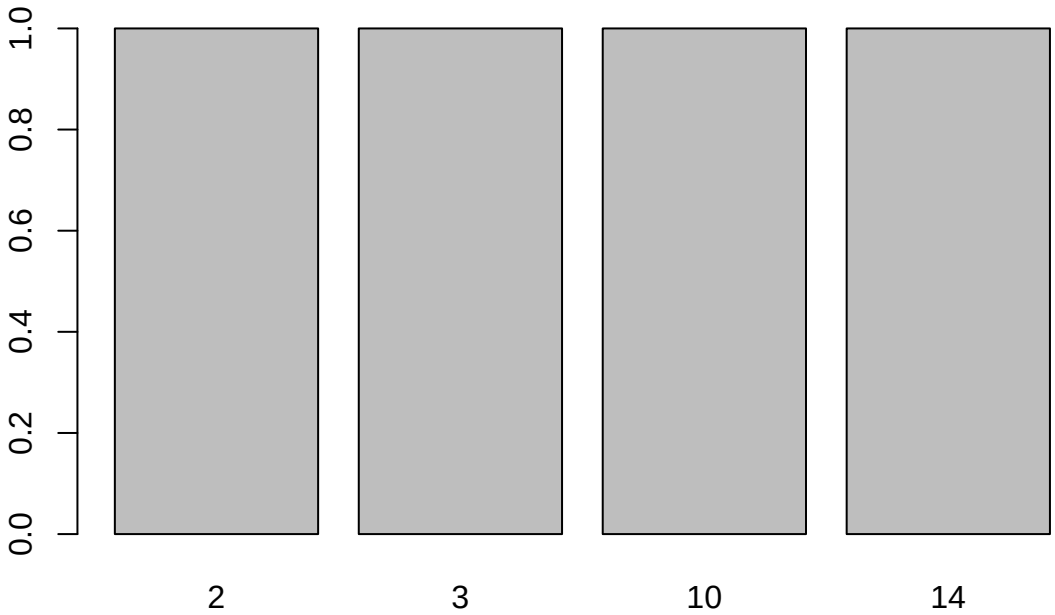
No, not different.

f

Turn the variable `observed_haircolours` into a factor. Make a plot of this new variable. What type of plot is it? Do you think this plot has anything interesting to say?

Answer:

```
plot(factor(observed_haircolours))
```



It's a barplot. The plot itself is not interesting at all. It just takes the values `2`, `3`, `10` and `14`, and acts as if each has been observed a single time.

g

Use `rep` to create a `factor` variable that contains the entries `blonde`, `brown`, `black` and `red`, as many times as given in `observed_haircolours`.

Answer:

rep(names(observed_haircolours),c(1,1,1,2)) #各重复1，1，1，2次

```
oh_factor <- factor(rep(names(observed_haircolours), observed_haircolours))
```

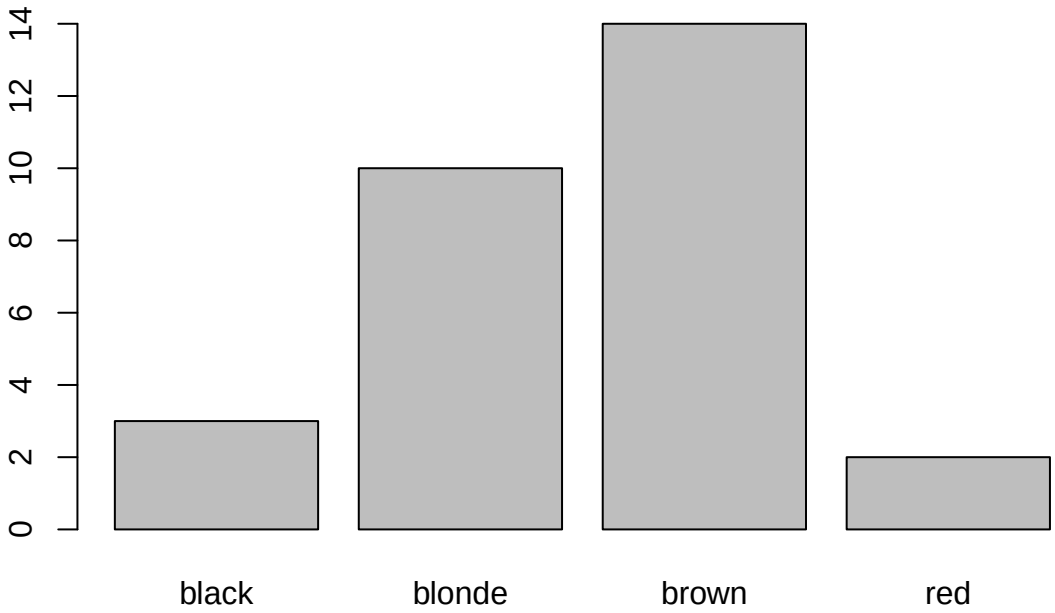
h

Use `plot` on this `factor`. Is this any better than in **f**?

Answer:

```
plot(oh_factor)
```

# Create a factor
factor(directions, levels= c("North", "East", "South", "West"), labels=c("N", "E", "S", "W"))
[1] N E S S
Levels: N E S W



Yes much better, this is the plot we wanted to get.

1.6 Some static plots

A group of 1000 people was taken in for some testing. Half are males, and the other half are females. In a moment read the example observed variables a bit further down this exercise. Convert these examples to data in `R` . Match each of the examples to one of the plot types given, and produce the corresponding plot.

1. They voted on whether they'd prefer tea or coffee. It is not quite clear how many of the 150 people actually voted, but of those who voted, 32% voted in favor of coffee, 45% voted in favour of tea, and 23% voted neutral.
2. We also measured their heights. The results are (randomly) normally distributed values with, for men, a mean of 185 cm's, and a standard deviation of 5 cm's. The heights of females are normally distributed with mean 175 and standard deviation 3.
3. Each was asked to roll a 6 sided die. The aggregated results are: 165 times 1, 158 times 2, 178 times 3, 156 times 4, 173 times 5, 170 times 6.
4. We also measured IQ scores of all the participants. The scores are normally distributed with mean 100, and sd 15. We are looking for outliers!

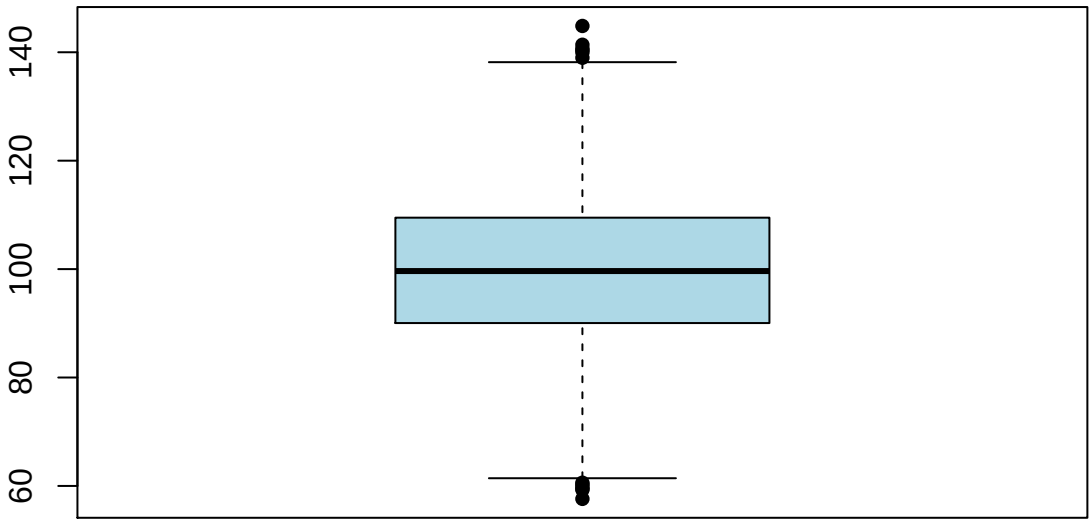
a. histogram
b. pie chart
c. boxplot (with only 1 box)
d. barchart

Create a nice title for each of the plots and make sure you give any categories nice labels. Choose a different colour yourself for each of the plots. Make sure to play with the `breaks` argument of the histogram function to make sure that your visualization is accurate.

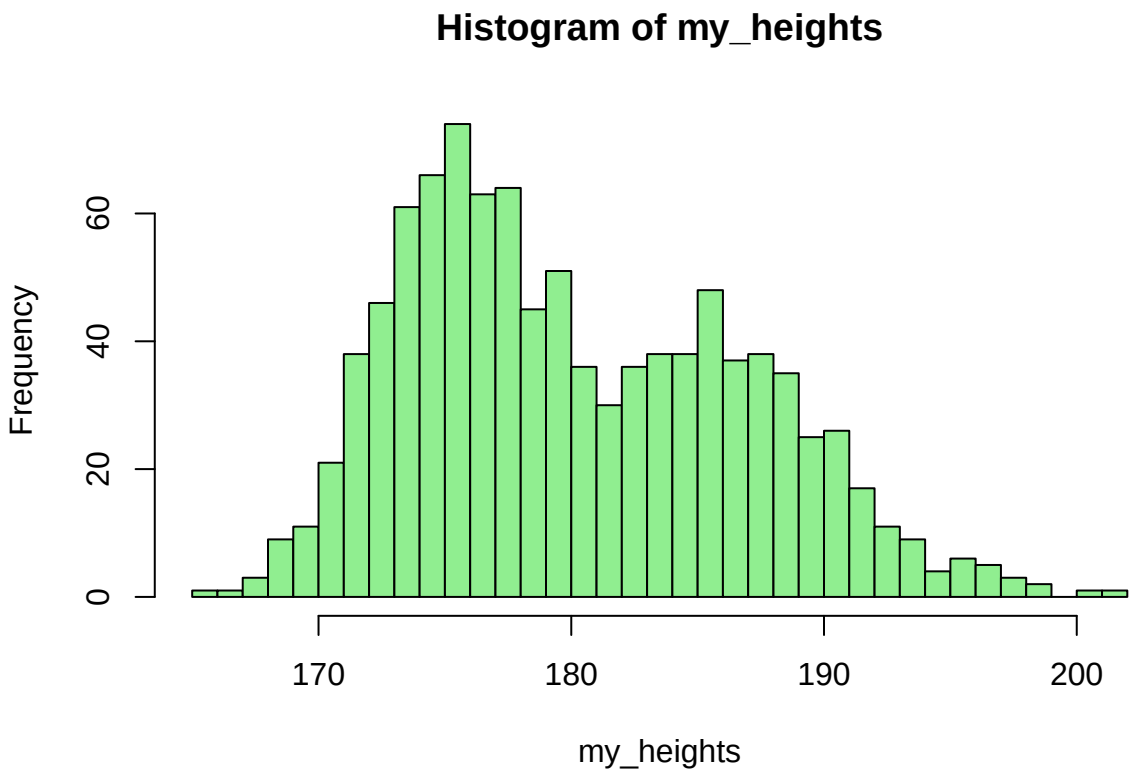
Answer:

```
set.seed(20171003)
N <- 1000
my_heights <- c(rnorm(N/2, mean = 185, sd = 5), rnorm(N/2, 175, sd = 3))
my_vote <- c(32, 45, 23)
```

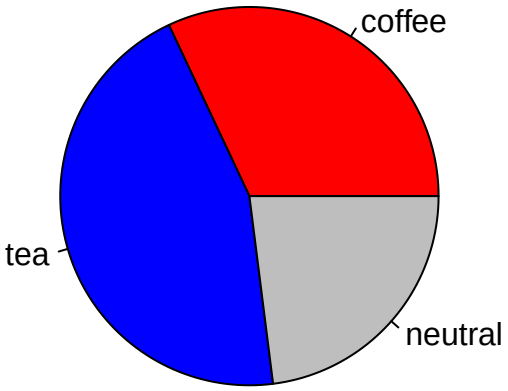
```r
my_rolls <- c(165, 158, 178, 156, 173, 170)
my_iq <- rnorm(N, mean = 100, sd = 15)

# histogram would also be ok, actually I prefer both next to eachother.
# histogram is nice to evaluate the shape of the distribution
# boxplot is nice to evaluate the critical points, such as the median, and the quantiles.
# also outliers are usually more nicely visible than in histogram.
boxplot(my_iq, col = 'lightblue', pch = 16)
```



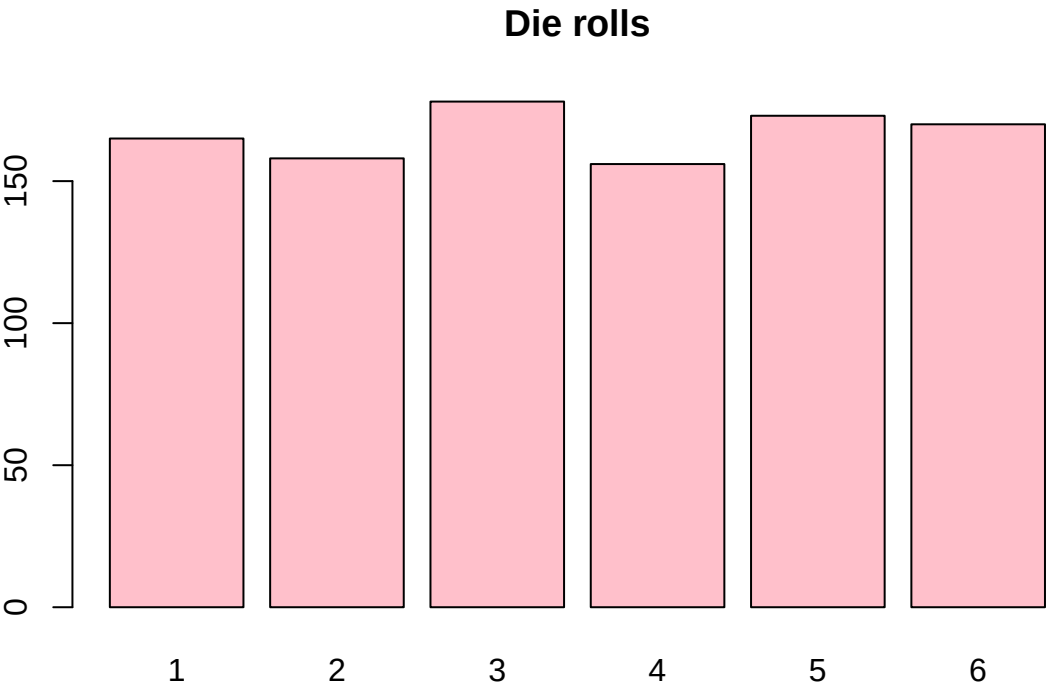

```r
# bimodality would be invisible when using a boxplot
hist(my_heights, col = 'lightgreen', breaks = 30)
```



```r
# percentages better in piechart:
pie(my_vote, labels = c("coffee", "tea", "neutral"), col = c('red', 'blue', 'grey'))
```




```r
# rolls in a barplot
barplot(my_rolls, col = 'pink', names.arg = c(1:6), main = "Die rolls")
```



1.7 Formula Boxplot

We've seen that `plot` has several methods. The same is true for `boxplot`. We'll explore a convenient method in this exercise.

a.

Create a factor called `animal` and sample, with equal probability, 1000 samples from the following 5 categories: `c("ant", "cat", "dog", "giraffe", "elephant")`.

Answer:

```r
set.seed(1234)
n_animals <- 1000
my_animals <- factor(
  sample(1:5, n_animals, replace = T),
  labels =  c("ant", "cat", "dog", "giraffe", "elephant"))
```

b.

Use your favorite search engine to look up the average weights of these animals. Create a variable called `weight`, with as many entries as sampled animals. Use a `for` loop (or an `*apply` function) and some `if` statements to fill `weight` with normally distributed values, with parameters depending on the type of animal. Use as mean for each of the distributions the average weight of the animals, and use as standard deviation 10% of the average body weight.

Answer:

This may not be the most elegant way to do this, but it is what we asked you to do!

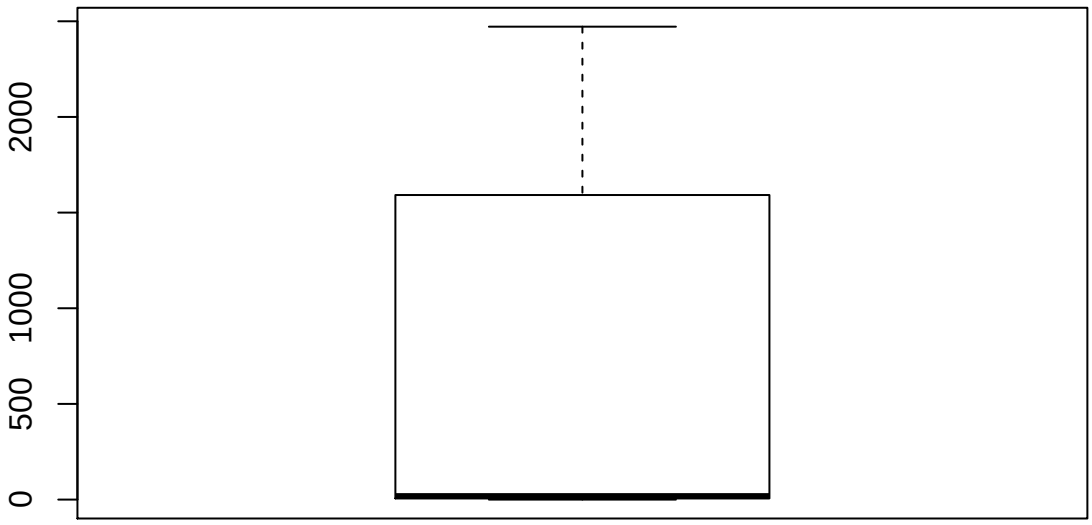
```r
# these are the weights we found:
animal_weight_means <- c(0.0025, 6.5, 18, 1500, 2000)
animal_weight_sd <- animal_weight_means * 0.1
weight <- numeric(n_animals)
for (i in 1:n_animals) {
  if (my_animals[i] == "ant") {
    weight[i] <- rnorm(1, mean = animal_weight_means[1], sd = animal_weight_sd[1])
  } else if (my_animals[i] == "cat") {
    weight[i] <- rnorm(1, mean = animal_weight_means[2], sd = animal_weight_sd[2])
  } else if (my_animals[i] == "dog") {
    weight[i] <- rnorm(1, mean = animal_weight_means[3], sd = animal_weight_sd[3])
  } else if (my_animals[i] == "giraffe") {
    weight[i] <- rnorm(1, mean =  animal_weight_means[4], sd = animal_weight_sd[4])
  } else if (my_animals[i] == "elephant") {
    weight[i] <- rnorm(1, mean = animal_weight_means[5], sd = animal_weight_sd[5])
  }
}
```

c

Make a boxplot of `weights`. Is there anything strange about this boxplot?

Answer:

```r
boxplot(weight)
```

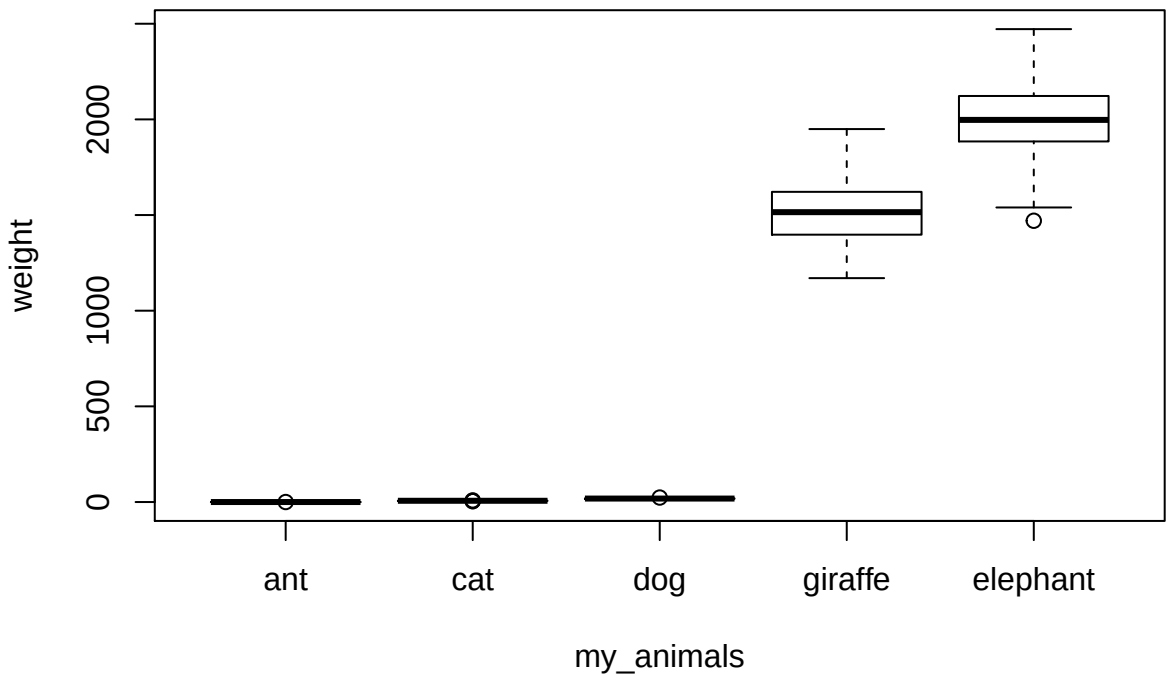


d

Of course this boxplot is not very nice, as we've basically created a boxplot of a multimodal distribution: the average weights depends on the animal. It would therefore be nicer to look at the weights for each of the animals seperately. Use `boxplot` with a `formula`, where you tell `R` that weight depends on the type of animal. Is this better?

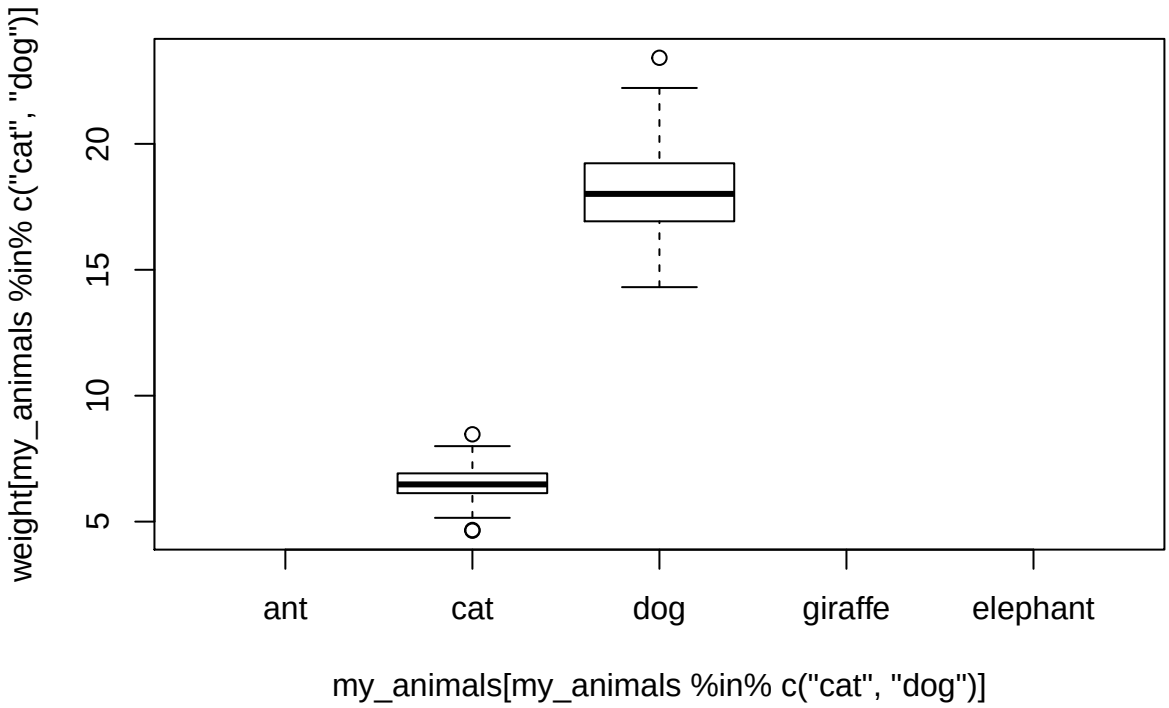
Answer:

```
boxplot(weight ~ my_animals)
```



Better yes, but not perfect, as we don't see the distribution of weights for the smaller animals very nicely. Better would be to also show those seperately. E.g. using:

```r
boxplot(weight[my_animals%in%c("cat", "dog")] ~  my_animals[my_animals%in%c("cat", "dog")])
```



Exercises part 2

2.1 Making our own qqchisq plot

As discussed during the lecture, a QQ-plot compares (estimates of) empericial quantiles with theoretical quantiles. In this exercise we will make our own QQ-plot function.

Take the following random variable `x` as an example.

```
set.seed(20171004)
x <- rchisq(500, df = 3)
```

a

Sort `x`.

Answer:

```
x <- sort(x)
```

b

Produce a vector, called `t_probs`, of length N, where N is the number of observations in X, with a sequence from $0.5/N$ to $(N - 0.5)/N$ with a stepsize of $1/N$.

Answer:

```
N <- length(x)
t_probs <- seq(0.5, (N - 0.5), by = 1)/N
```

c

Turn `t_probs` into values such that the cumulative probabilities correspond with quantiles of a chisquare distribution with degrees of freedom equal to 3.

Answer:

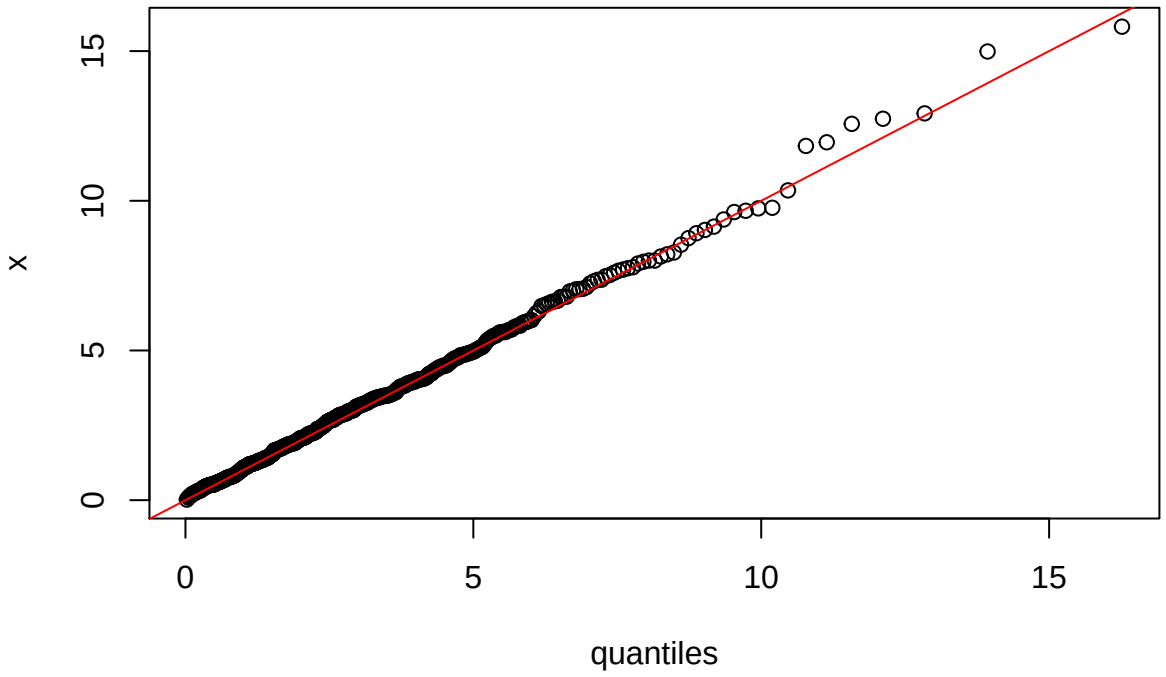
```
quantiles <- qchisq(t_probs, df = 3)
```

d

Plot the observed values `x` against the theoretical quantiles obtained in **c**. Draw a line with unit slope and no intercept. Would you say, from this plot, that the data is distributed similarly to the theoritical distribution we considered?

Answer:

```
plot(quantiles, x)
abline(a = 0, b = 1, col = 'red')
```

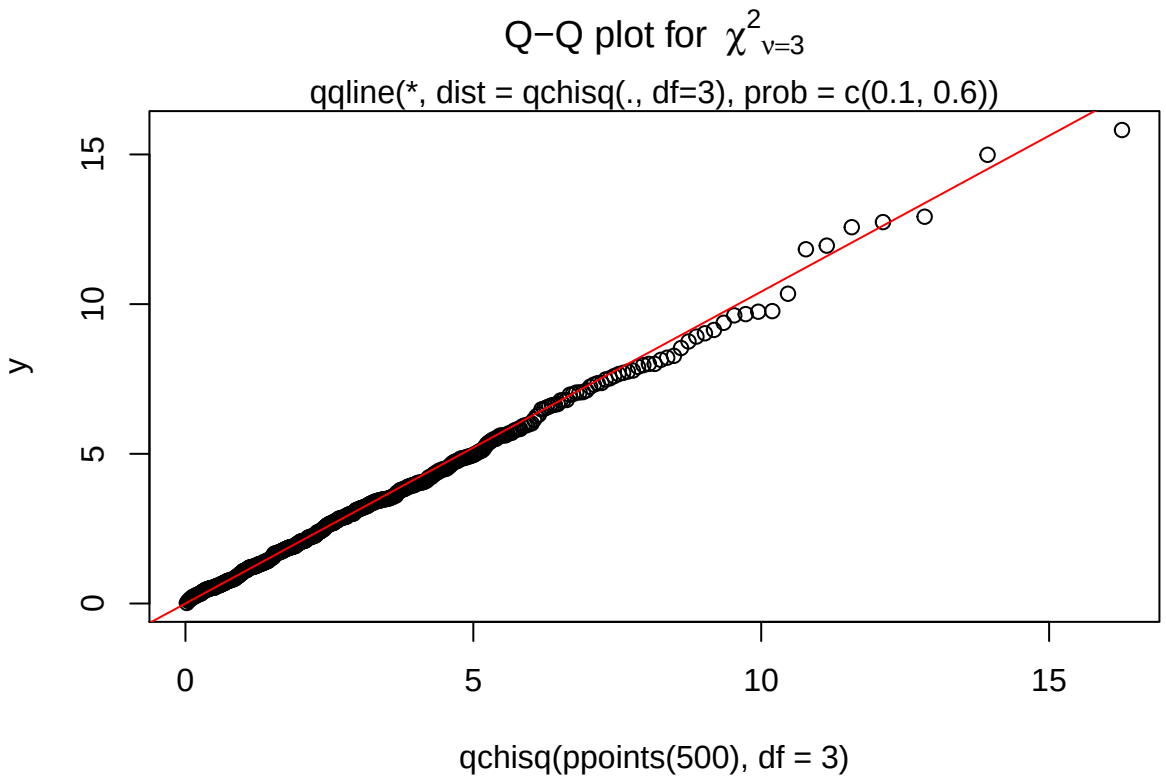


e

Look at the helpfile of `qqplot` and create a QQ-plot for a chisquare distribution using the code from the example. Are there any differences betweem the plot we made, and the plot of the example (besides aesthetical stuff, such as a plot title)?

Answer:

```
## Q-Q plot for Chi^2 data against true theoretical distribution:
y <- x
qqplot(qchisq(ppoints(500), df = 3), y,
       main = expression("Q-Q plot for" ~~ {chi^2}[nu == 3]))
qqline(y, distribution = function(p) qchisq(p, df = 3),
       prob = c(0.1, 0.6), col = 2)
mtext("qqline(*, dist = qchisq(., df=3), prob = c(0.1, 0.6))")
```



`qqline` creates a different line: one that goes through the 0.25th and 0.75th quartiles.

Outro

You may have noticed in **e** that `R` has a different convention when it comes to drawing a `qqline`. It draws a line that passes through the 0.25th, and 0.75th quartiles. This is a more robust way of drawing the line, without need to rescale the line in case of data that looks slightly different. For example redraw values for `x` using:

```
set.seed(20171004)
x <- rchisq(500, df = 3) + 3
```

and go through your plotting procedures again. Do you see that the line is shifted, compared to the data? This is usually 'easily' fixed, but a more simple way is to just simply draw the `qqline` always through the 0.25th and 0.75th quartiles. (In this course, both styles are fine! Just be aware of it!)

2.2 Looking into sampling distributions

In this exercise we will take a closer look at sampling distributions, and the relationship between the variance of a sample, and the variance of a sample mean. The relation is given by: $var(\bar{X}) = \frac{var(X)}{N}$.

For some extra info/recap view, for example, the first part of this video (until 4:12). You may also want to take a look at: http://onlinestatbook.com/2/estimation/mean.html.

a

Generate many samples, e.g 734, of size 100 ($N = 100$) where each sample is i.i.d. according to a normal distribution with $\mu = 2$ and $\sigma = 3$. Repeat this for samples with $N = 10$ and $N = 1000$. Take `set.seed(1212)`.

Show in R, with computations on the generated samples, that the relation between sample variance and sample mean variance holds. Think for a second about the strenght of your evidence upon which you base your conclusions.

Answer:

```
set.seed(1212)
B <- 734

N <- 10
N10 <- replicate(B, mean(rnorm(N, mean = 2, sd = 3)))

N <- 100
N100 <- replicate(B, mean(rnorm(N, mean = 2, sd = 3)))

N <- 1000
N1000 <- replicate(B, mean(rnorm(N, mean = 2, sd = 3)))

sd(N10) * sqrt(10)
```

```
## [1] 3.12486
```

```
sd(N100) * sqrt(100)
```

```
## [1] 3.047378
```

```
sd(N1000) * sqrt(1000)
```

```
## [1] 2.985925
```

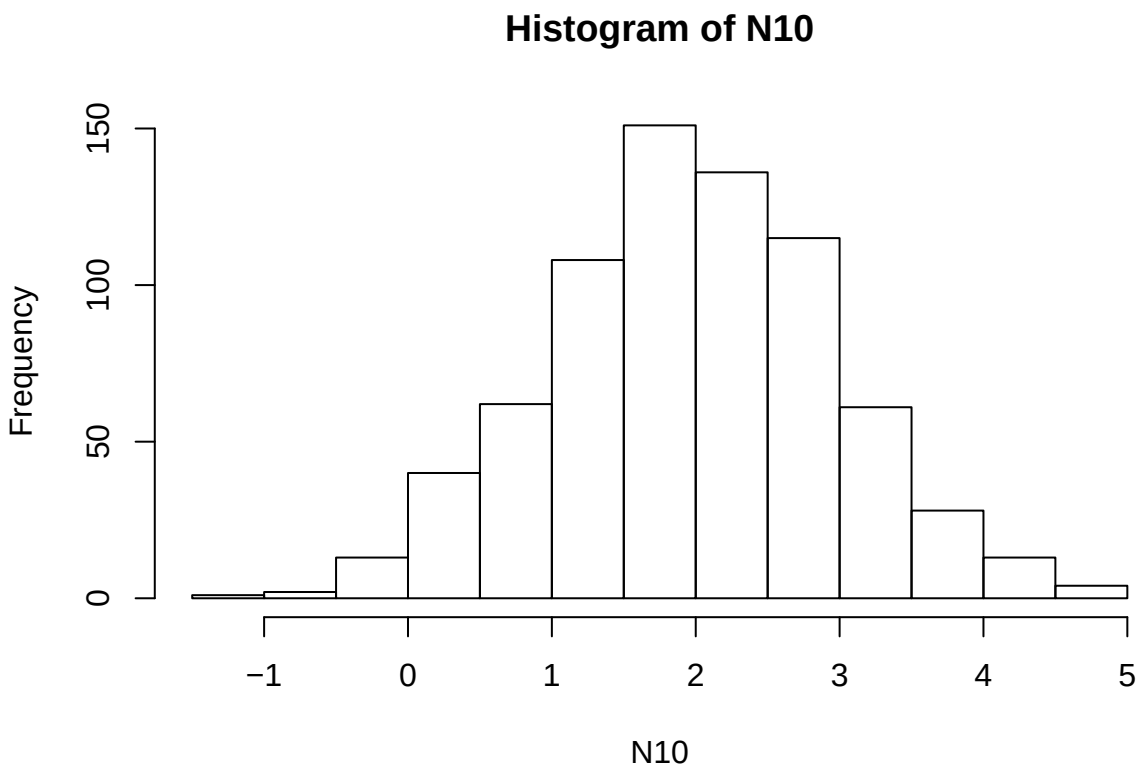
b

Instead of expression the relationship in numbers (as in one of the previous exercises), we now want to express it using some visualization. Do this in the following two ways:

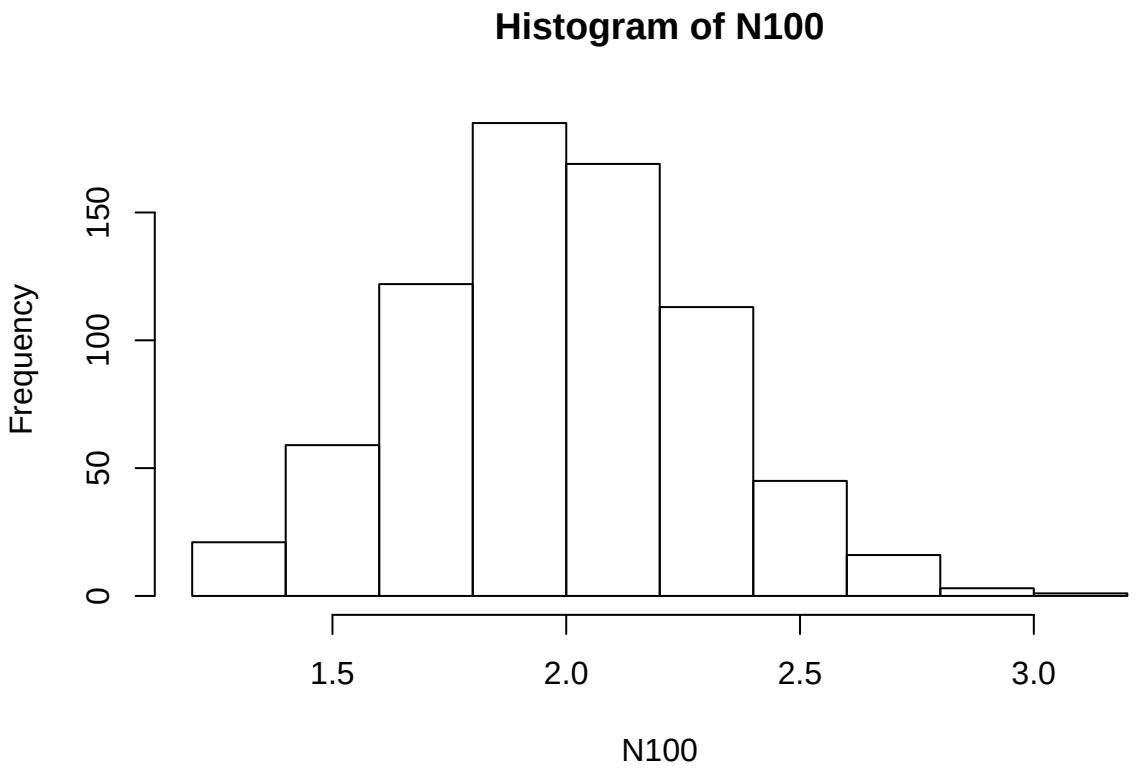
a) by visualizing the sampling distributions of the mean for the samples with the three different sample sizes generated for question 1 (you may show three separate plots)
b) by computing the 95% confidence intervals of the mean for three samples picked from the samples you generated for question 1 (i.e., one sample with $N = 10$, one with $N = 100$ and one with $N = 1000$). Assume in the computation of the confidence interval, that you do not know σ, thus you have to use the sample estimate of it (s). Furthermore, make use of the t-distribution for choosing the values of t to be used in a confidence interval. In R, you can obtain these t-values using the function `qt()`. For example, the values for a sample size of $n = 5$ are obtained by `qt(c(.025, .975), df = 4)`.

Answer:

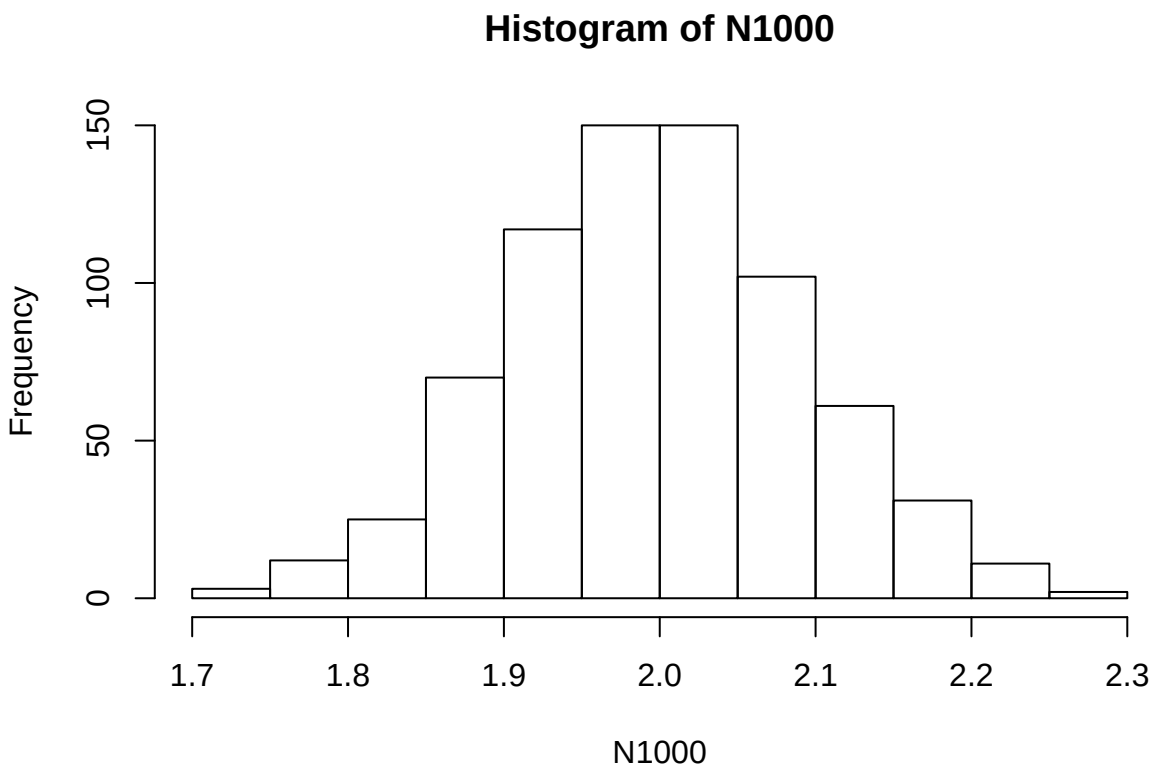
```
hist(N10)
```




```
hist(N100)
```




```
hist(N1000)
```




```
limits_N10 <- qt(c(0.025, 0.975), df = 9)
limits_N100 <- qt(c(0.025, 0.975), df = 99)
limits_N1000 <- qt(c(0.025, 0.975), df = 999)

mean(N10) + limits_N10 * sd(N10)
```

```
## [1] -0.2590674  4.2117133
```

```
mean(N100) + limits_N100 * sd(N100)
```

```
## [1] 1.379963 2.589295
```

```
mean(N1000) + limits_N1000 * sd(N1000)
```

```
## [1] 1.811918 2.182500
```

c

Write a function in R that computes the mean and its 95% confidence interval of a variable. The input argument of the function is a vector that can have different lengths. The output is a list with two components: the mean, and the 95% confidence interval.

Answer:

```
CI <- function(x){
  N <- length(x)
  limits <- qt(c(0.025, 0.975), df = N - 1)
```

```
  mean <- mean(x)

  return(list(
    mean = mean,
    ci = mean + limits * sd(x)
  ))
}
CI(N1000)
```

```
## $mean
## [1] 1.997209
##
## $ci
## [1] 1.811837 2.182581
```

2.3 Contour: visualizing a pyramid

In this exercise we'll visualize a pyramid, using a 2D plot. Of course you all know what a pyramid looks like, but if you're unsure, go to the library and look in some history textbooks. Hopefully, using such a simple shape will give you some insight in how to read a `contour`plot, and how to create one.

a

Create a 15 by 15 matrix. Fill the outer ring of values with a 1, fill the second ring from outside with a 2, etc. These value indicate the 'height' of the pyramid at that point.

Here's some cody to create a 5 by 5 matrix as an example:

```
nrow <- 5
ncol <- 5
pyramid <- matrix(0, nrow=nrow, ncol=ncol)
for (i in 1:nrow){
  for (j in 1:ncol){
    pyramid[i, j] <- min(min(i, nrow-i+1), min(j, ncol-j+1))
  }
}

# look at the contents:
pyramid
```

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

Answer:

```r
nrow <- 15
ncol <- 15
pyramid <- matrix(0, nrow=nrow, ncol=ncol)
for (i in 1:nrow){
  for (j in 1:ncol){
    pyramid[i, j] <- min(min(i, nrow-i+1), min(j, ncol-j+1))
  }
}

# look at the contents:
pyramid
```

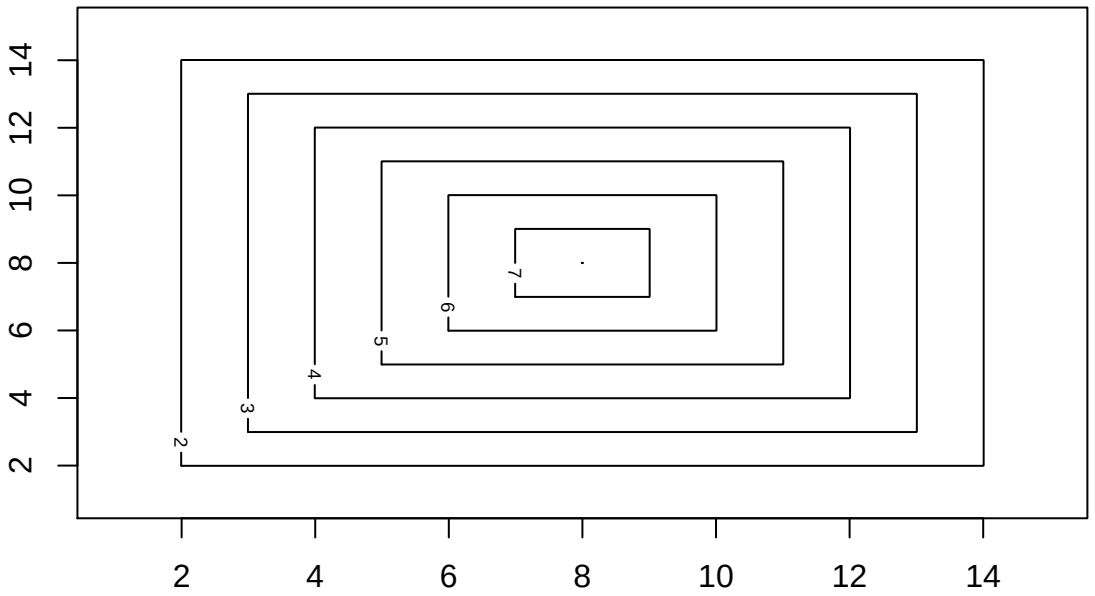
```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
##  [1,]    1    1    1    1    1    1    1    1    1     1     1     1     1
##  [2,]    1    2    2    2    2    2    2    2    2     2     2     2     2
##  [3,]    1    2    3    3    3    3    3    3    3     3     3     3     3
##  [4,]    1    2    3    4    4    4    4    4    4     4     4     4     3
##  [5,]    1    2    3    4    5    5    5    5    5     5     5     4     3
##  [6,]    1    2    3    4    5    6    6    6    6     6     5     4     3
##  [7,]    1    2    3    4    5    6    7    7    7     6     5     4     3
##  [8,]    1    2    3    4    5    6    7    8    7     6     5     4     3
##  [9,]    1    2    3    4    5    6    7    7    7     6     5     4     3
## [10,]    1    2    3    4    5    6    6    6    6     6     5     4     3
## [11,]    1    2    3    4    5    5    5    5    5     5     5     4     3
## [12,]    1    2    3    4    4    4    4    4    4     4     4     4     3
## [13,]    1    2    3    3    3    3    3    3    3     3     3     3     3
## [14,]    1    2    2    2    2    2    2    2    2     2     2     2     2
## [15,]    1    1    1    1    1    1    1    1    1     1     1     1     1
##       [,14] [,15]
##  [1,]     1     1
##  [2,]     2     1
##  [3,]     2     1
##  [4,]     2     1
##  [5,]     2     1
##  [6,]     2     1
##  [7,]     2     1
##  [8,]     2     1
##  [9,]     2     1
## [10,]     2     1
## [11,]     2     1
## [12,]     2     1
## [13,]     2     1
## [14,]     2     1
## [15,]     1     1
```

b

Use `contour` to visualize the pyramid.What is used to represent the height of the pyramid at each point?

Answer:

```
contour(x=1:nrow, y=1:ncol, z=pyramid, nlevels=length(unique(c(pyramid))))
```



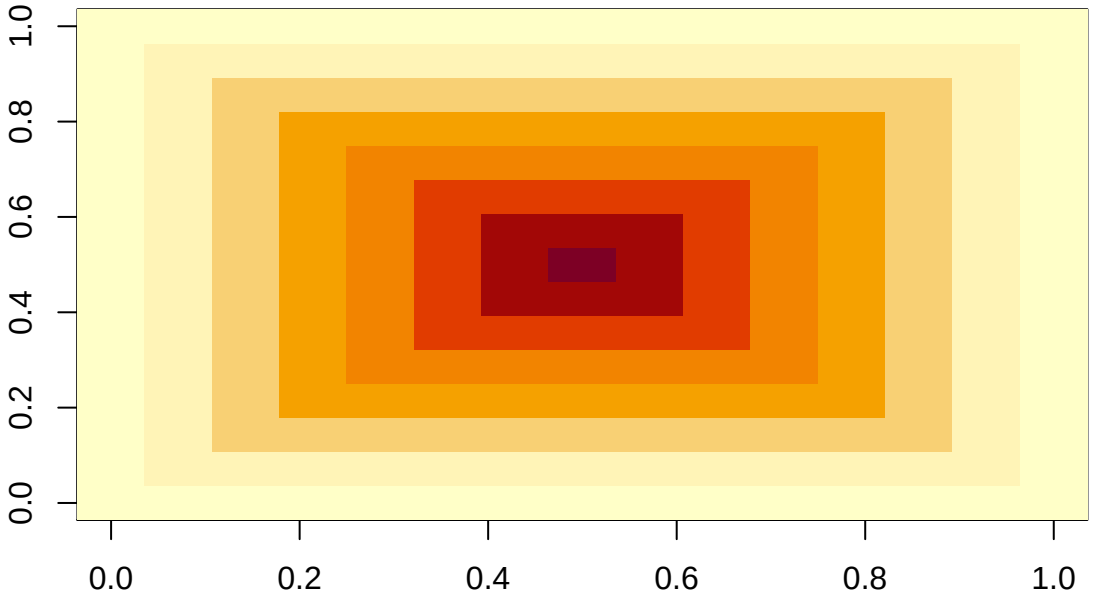
Height lines! Like an old school map!

c

Use `image` to visualize the pyramid. What is used to represent the height of the pyramid at each point?

Answer:

```
image(pyramid)
```



The colour of course!

2.4 Make a heat map and save the plot as `.pdf`

Take a look at the following link: heatmap. Follow the coding instructions on the website. You can find the data (`ppg2008.csv`) on the website, or in the `data` folder. Save the final heatmap as a `.pdf` file.

Answer:

```
nba <- read.csv("http://datasets.flowingdata.com/ppg2008.csv", sep=",")
row.names(nba) <- nba$Name
nba <- nba[,2:20]
nba_matrix <- data.matrix(nba)
pdf("0_images/nba_heatmap.pdf")
nba_heatmap <- heatmap(
  nba_matrix, Rowv=NA, Colv=NA, col = cm.colors(256),
  scale="column", margins=c(5,10))
dev.off()
```

```
## pdf
##   2
```

3 Self-study

3.1 Visualizing a bivariate normal density

Before you start this exercise, install the package `mvtnorm`.

a

Take a look at `dmvnorm` in the `mvtnorm` package. To calculate the density function of a bivariate normal distribution, you will need the covariance matrix of the distribution. Create one, where both marginal distributions are standard normally distributed, and are correlated with a coefficient of 0.7.

Answer:

```
sigma_cor <- matrix(c(1, 0.7, 0.7, 1), ncol=2)
```

$$\boldsymbol{\mu} = \begin{pmatrix} \mu_X \\ \mu_Y \end{pmatrix}, \quad \boldsymbol{\Sigma} = \begin{pmatrix} \sigma_X^2 & \rho\sigma_X\sigma_Y \\ \rho\sigma_X\sigma_Y & \sigma_Y^2 \end{pmatrix}.$$

b

Take a look at `expand.grid`. Use it to create quantiles, so you can evaluate the bivariate normal density at many points. For example at `x1 = -3`, and `x2 = 0`, or `x1 = -3`, and `x2 = 1`, etc. You will need many points (e.g. many combinations of values of `x1` and `x2`) to get a nice picture of the density.

Answer:

```
quantiles_seq <- seq(-3, 3, by = 0.1)
quantiles <- expand.grid(quantiles_seq, quantiles_seq)
```

c.

Put the quantiles you've created into `dmvnorm` twice, once with all the defaults, and once also providing the covariance matrix you created in **a**. Save the results of both operations.

Answer:

```
library(mvtnorm)
densities_uncor <- dmvnorm(quantiles)
densities_cor <- dmvnorm(quantiles, sigma = sigma_cor)
```

d.

If we want to use `contour`, we need to put the density values into a matrix. The entries in this matrix represent a grid, with density evaluations at the intersections of the gridlines. An important issue is that we need to make sure the ordering of the values is correct: e.g. the value in the second row, and third column (where grid lines for row two and column three meet), needs to be the value of evaluating the second quantile for the first normal variable and the third quantile for the second normal variable.

Convert the quantiles you've created into such a matrix now.

Answer:

```
head(quantiles)
```

```
##   Var1 Var2
## 1 -3.0   -3
## 2 -2.9   -3
## 3 -2.8   -3
## 4 -2.7   -3
## 5 -2.6   -3
## 6 -2.5   -3
```

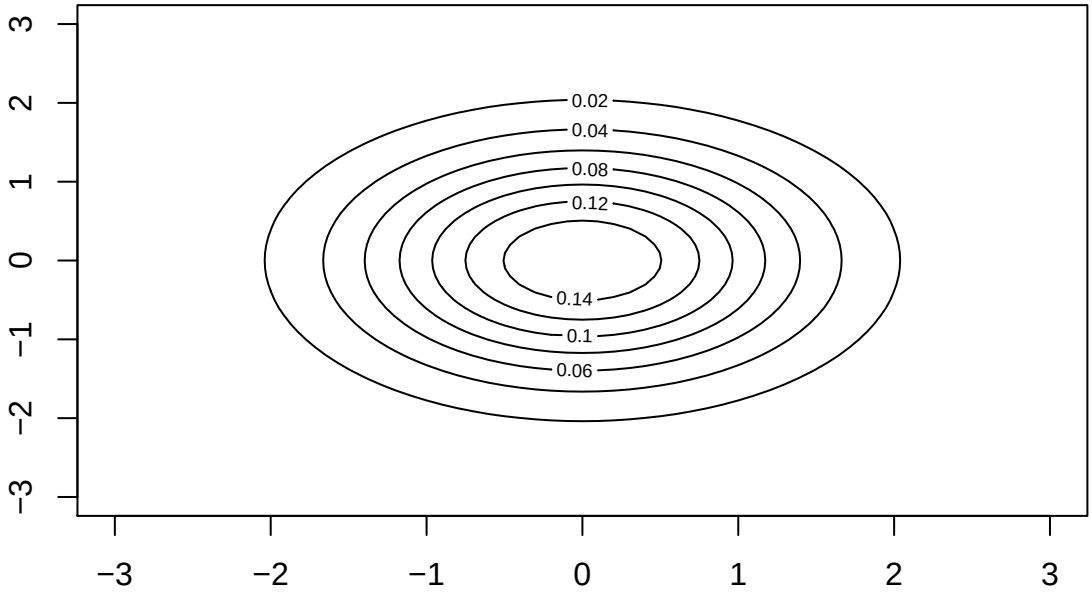
```r
# in my case, for a single var2 quantile, all the quantiles of var1 are calculated
# this is as if we are in a single column of a matrix, and provide the row entries.
# so if we fill or matrix by columns, the various values of var1 will correspond to the rows
# and the various values of var2 will correspond to the columns
densities_uncor_matrix <- matrix(densities_uncor, ncol = sqrt(length(densities_uncor)))
densities_cor_matrix <- matrix(densities_cor, ncol = sqrt(length(densities_cor)))
```

e.

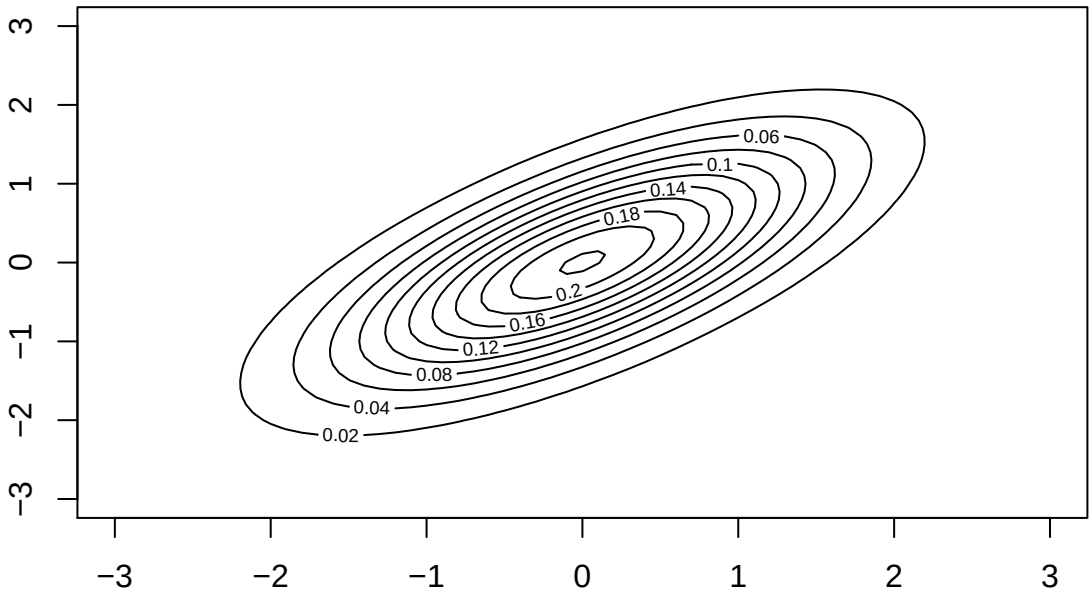
You now have all the ingredients you need for `contour`. Make a contourplot of both bivariate normal distributions. Do they look the same? What's the big difference?

Answer:

```r
contour(x = quantiles_seq, y = quantiles_seq, z = densities_uncor_matrix)
```



```
contour(x = quantiles_seq, y = quantiles_seq, z = densities_cor_matrix)
```



3.2 Hexbin

we commented most of the plotting code as it takes forever to plot and to show the plots in the pdf file, so uncomment or copy the code if you want to check your answers!

a

Read in the file `mystery.txt`. NB: it's a very big file, so this might take a few seconds (the same goes for all the plotting you do during this exercise).

Answer:

```
my_mystery <- read.table("0_data/mystery.txt")
```

b

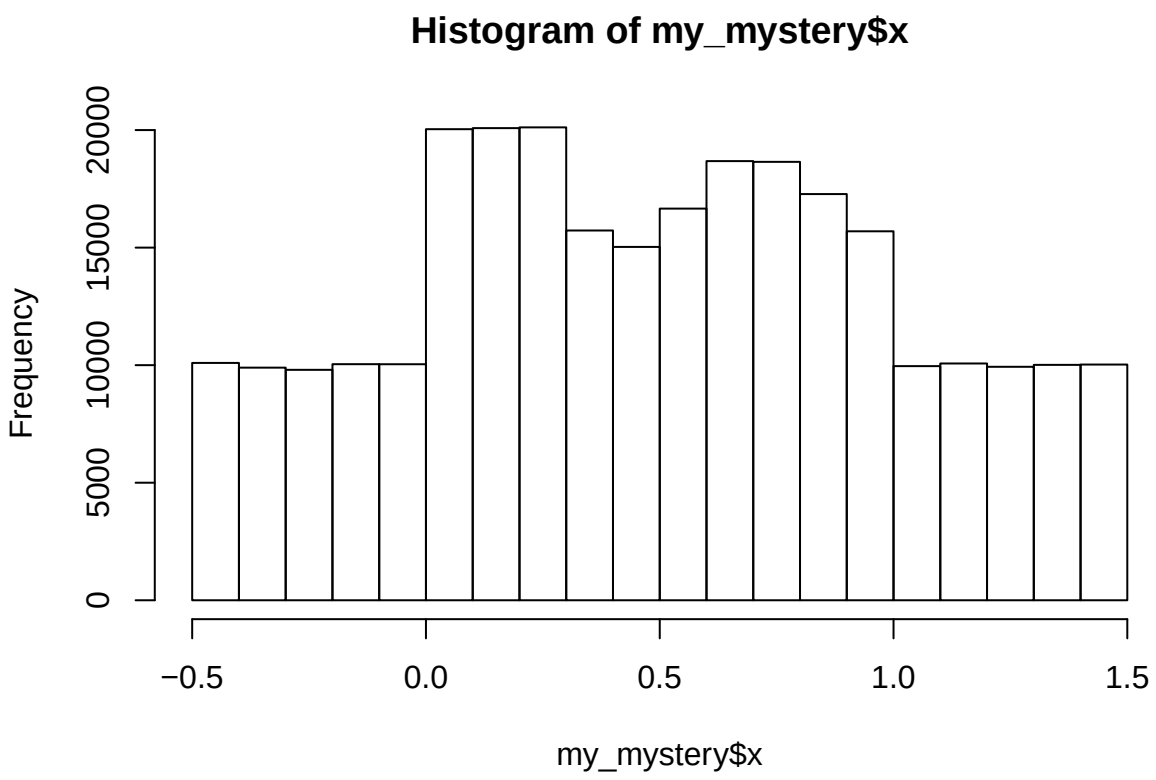
Use some summary statistics functions on the data in the mystery file. Make for example a histogram of `x` and `y`. Do you see anything interesting?

Answer:

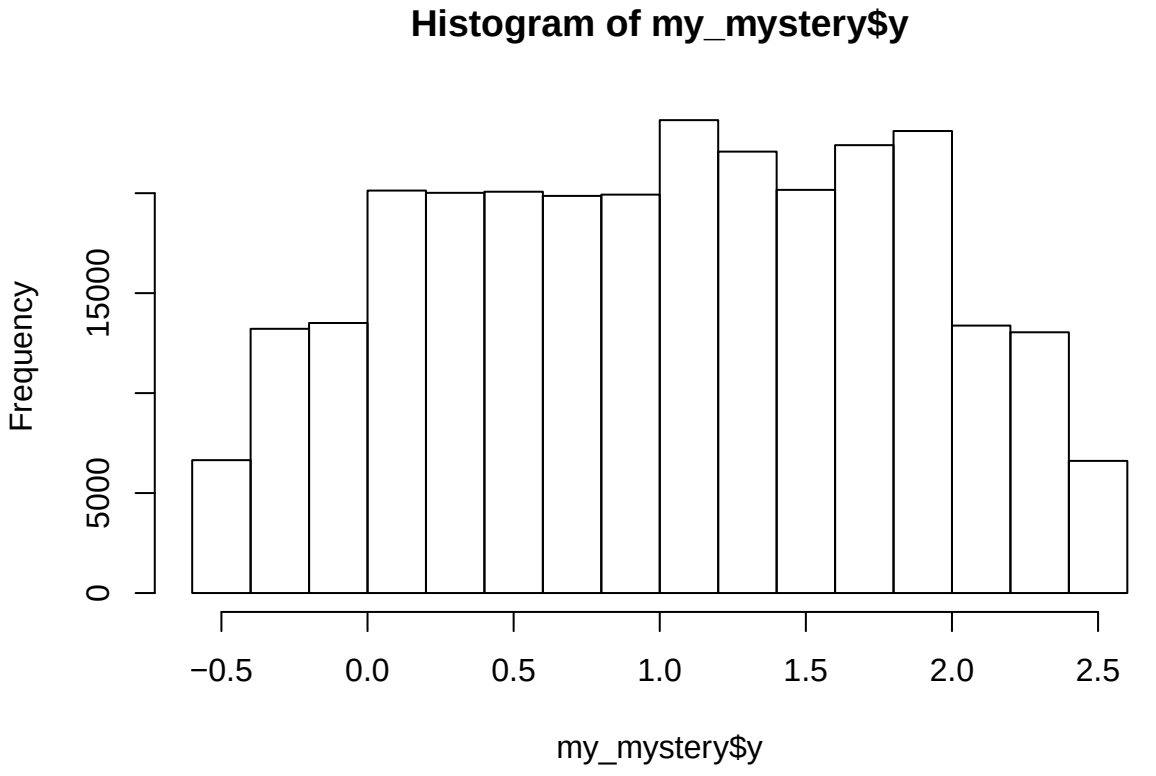
```
summary(my_mystery)
```

```
##        x                   y
##  Min.   :-0.50000   Min.   :-0.5000
##  1st Qu.: 0.09759   1st Qu.: 0.3593
##  Median : 0.48726   Median : 1.0465
##  Mean   : 0.49020   Mean   : 1.0180
##  3rd Qu.: 0.87762   3rd Qu.: 1.6866
##  Max.   : 1.49998   Max.   : 2.5000
```

```
hist(my_mystery$x)
```




```
hist(my_mystery$y)
```



Not really. . .

c

The univariate stuff was boring. Let's now look at some bivariate stuff. Are `x` and `y` correlated? Does a simple `plot` of the entries show anything interesting?

Answer:

```
cor(my_mystery)
```

```
##             x           y
## x 1.000000000 0.008873986
## y 0.008873986 1.000000000
```

```
# plot(my_mystery)
```

Hmm.... not really but that's probably due to the fact that we can't see anything simply due to the sheer amount of observations.

d

Instead of `plot`, use `plot(hexbin())` from the `hexbin` library. Play around with the argument `xbins`. Do you see something interesting?

Answer:

```
library(hexbin)
# plot(hexbin(my_mystery, xbins=100))
```

Clearly there's an R hidden in there!

1. the xy plane over the set (range(x), range(y)) is tessellated by a regular grid of hexagons.
2. the number of points falling in each hexagon are counted and stored in a data structure
3. the hexagons with count > 0 are plotted using a color ramp or varying the radius of the hexagon in proportion to the counts.

e

Also try `smoothScatter`, you may want to play around with the `bandwith` and the `nbin` arguments. Do you prefer `hexbin` or `smoothScatter`?

```
# smoothScatter(my_mystery, bandwidth = 0.001, nbin=1000)
```

`smoothScatter` requires an extra argument we need to 'tune', it does provide more 'smooth' plots though: the `hexbin` plot is, by definition, a collection of hexagons, while `smoothScatter` can take on more 'fluid' shapes.

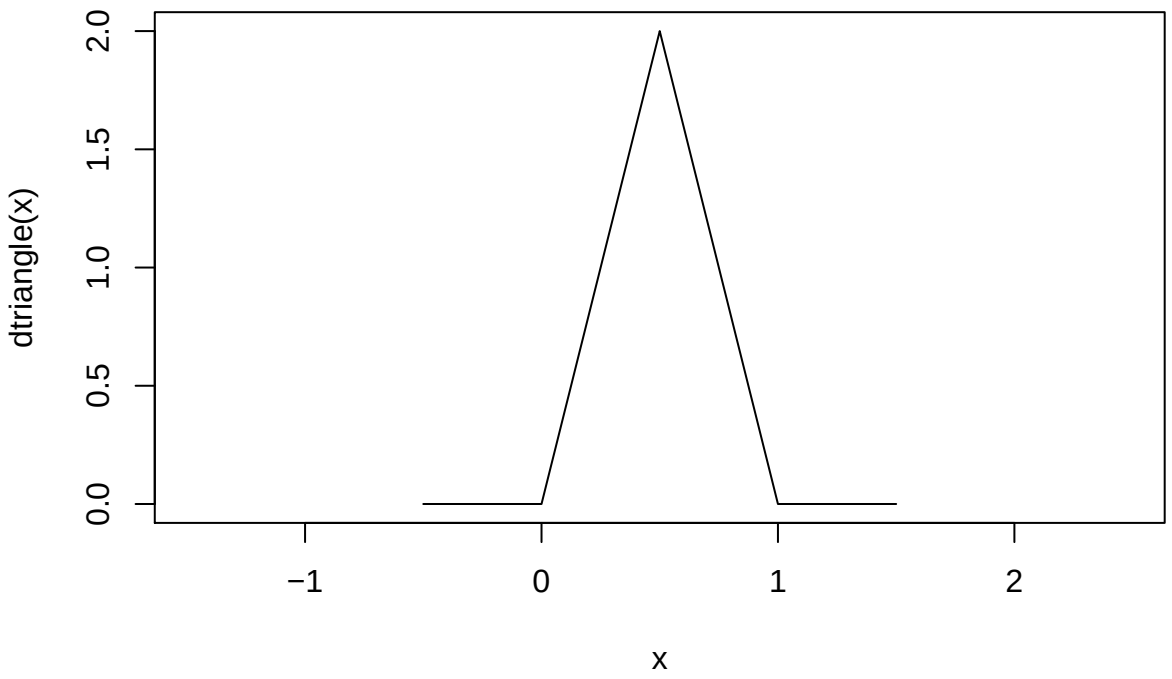
3.4 Sampling from a triangle

During the lecture we've talked about how the cumulative distribution function can be employed to produce random numbers from that distribution. Look at the following density function:

- $4x$, for $x \in [0, 0.5)$
- $4(1 - x)$ for $x \in [0.5, 1]$

See an implementation below.

```r
dtriangle <- function(x){
  (x >= 0 & x < 0.5) * 4 * x + (x >= 0.5 & x <= 1) * 4 * (1 - x)
}
curve(dtriangle, asp = 1, from = -0.5, to = 1.5)
```



a

Integrate the density of the triangle over its domain with `integrate()`, is it a proper density function?

Answer:

```r
integrate(dtriangle, lower = -1, upper = 2)$value
```

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

Up to numerical error, yes.

b

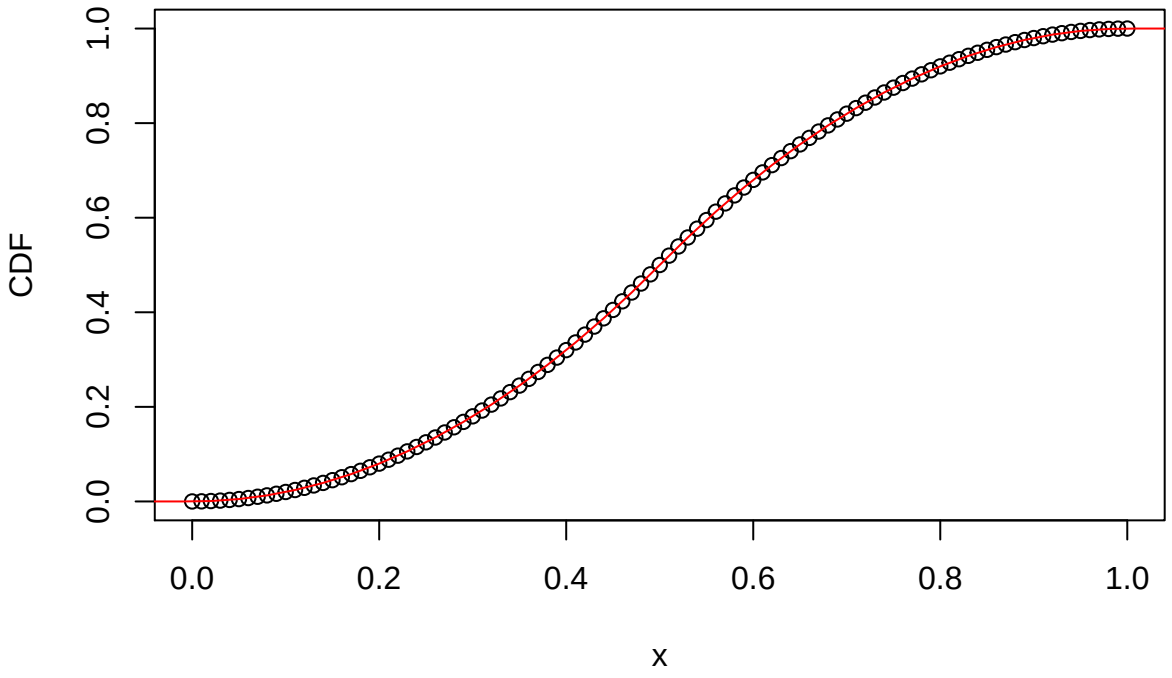
Formulate a cumulative distribution function and write a function called `ptriangle` that implements this. Note that besides integrating the density function, you may also need to choose a proper constant, such that the minimum of the function is 0, and the maximum is 1.

Answer:

Integral (with properly chosen constant): * $2x^2$, for $x \in [0, 0.5)$ * $-2 * (x+1)^2$ for $x \in [0.5, 1]$

```r
y_var <- sapply(seq(0, 1, by=0.01), function(i) integrate(dtriangle, lower=-1, upper=i)$value)
plot(
  x = seq(0, 1, by=0.01),
```

```
  y = y_var,
  ylab = "CDF", xlab = "x"
)
ptriangle <- function(x){
  (x >= 0 & x < 0.5) * 2 * x^2 + (x >= 0.5 & x <= 1) * (1 + -2 * (x-1)^2) + (x > 1)
}
curve(ptriangle, from=-1, to=2, add=T, col='red')
```



c

Formulate the inverse cumulative distribution function and write a function called `qtriangle` that implements this.

The inverse of the CDF (i.e. the Inverse Function) tells you what value x (in this example, the z-score) would make F(x)—

Answer:

Inverse cumulative distribution: * $4x$, for $x \in [0, 0.5)$ * $4(1-x)$ for $x \in [0.5, 1]$

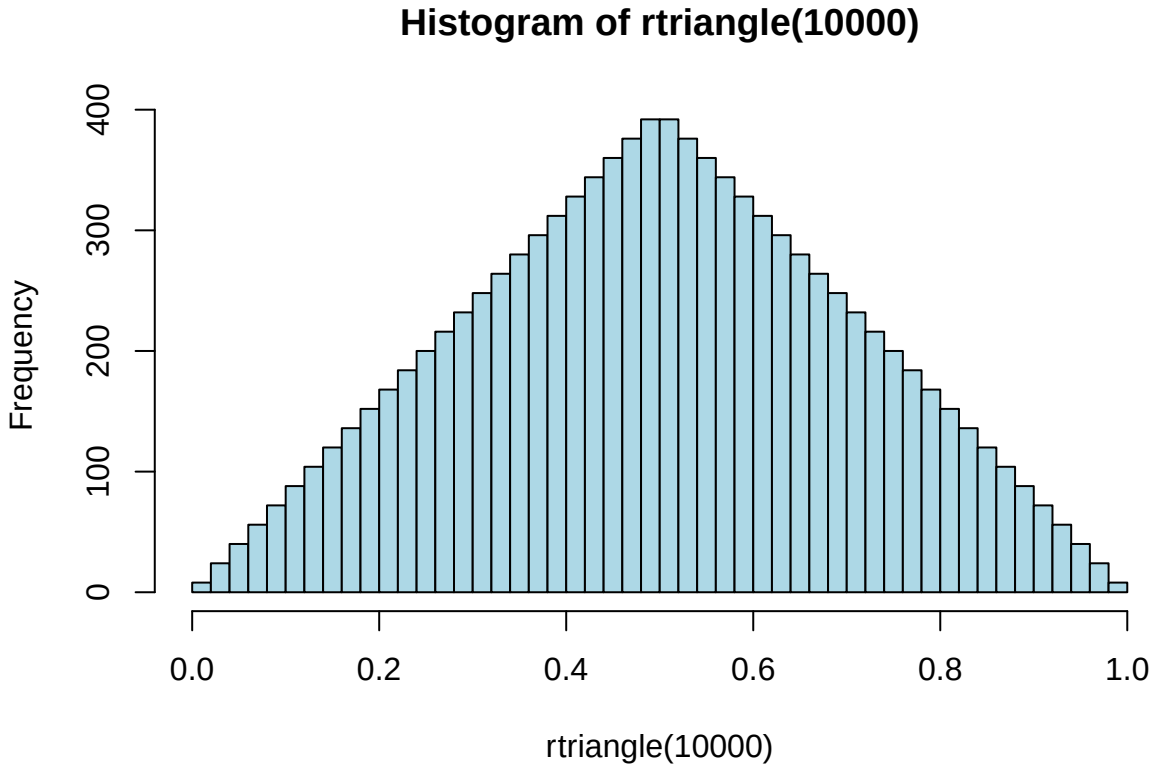
```
qtriangle <- function(x){
  (x >= 0 & x < 0.5) * sqrt(x / 2) + (x >= 0.5 & x <= 1) * (1 - sqrt((1 - x) / 2)) + (x > 1)
}
```

d

Write a function called `rtriangle` that samples `n` random values from a triangle distribution.

Answer:

```
rtriangle <- function(n){
  x <- seq(0, 1, length = n)
  return(qtriangle(x))
}
hist(rtriangle(1e4), breaks = "FD", col = 'lightblue')
```



3.5 Coordinate descent (For advanced students only!)

We've previously seen the Gradient descent algorithm (Week 4). A short recap: there are various methods of estimating the parameters of a statistical model e.g. least squares or maximum likelihood estimation. In maximum likelihood estimation a maximum of a real function (the likelihood function) needs to be found by means of optimization. Various optimization methods exist for finding the minimum or maximum of a real function.

Like Gradient descent, Coordinate descent is an iterative algorithm, that can be used to find a (local) minimum of a function that involves more than 1 parameter! The idea is that one conditions on all parameters except 1, update the parameter we did not condition on, and repeat this for all parameters. We repeat this cycle many times until convergence.

Take for example the function $f(x, y) = x^2 + y^2 + x \cdot y)$. The minimum can be found by condition on some x (keep it fixed) and find the minimum for y given this value of x. We then condition on this new value of y, and find a new x (the one that minimizes f, w.r.t. x, given a value for y). We then repeat this until the parameters x and y do not change in between updates.

a

Write an `R` function that uses the idea of coordinate descent to find the minimum of a function of the form:

$f(x, y) = ax^2 + by^2 + cx \cdot y$, with a, b and c some real numbers.

You will need:

- A function to evaluate the partial derivative w.r.t. x, and solve for x s.t. $f'_x(x, y|y) = 0$
- A function to evaluate the partial derivative w.r.t. y, and solve for y s.t. $f'_y(x, y|x) = 0$
- A while loop that repeats the update cycle until the sum of the absolute differences $|x_{n+1} - x_n|$ and $|y_{n+1} - y_n|$ is smaller than `1e-4`.

The input of the function should be a starting value for x, a starting value for y, a, b, and c. The output should be a dataframe with an entry for the values of x and y at each step of the algorithm. So the first entry of this dataframe is the starting values, provided by you, of x and y. The second entry is the first value of y, and the update of x, given the current value of y. The third entry is the updated value of x and the update value of y. And so on.

Make sure that the algorithm performs no more than a 100 cycles of updating both x and y

Answer:

```
#partial derivatives, w.r.t. x
SolveFx <- function(y, a, c){
  return(-c*y/(2*a))
}

#partial derivative w.r.t. y
SolveFy <- function(x, b, c){
  return(-c*x/(2*b))
}

CoordinateDescent <- function(x0, y0, a, b, c){

  #start value for eps:
  eps <- Inf
  max.iter <- 100

  x.vec <- numeric(max.iter*2)
  y.vec <- numeric(max.iter*2)

  counter <- 1
  cycle <- 1
  x.vec[counter] <- x0
  y.vec[counter] <- y0

  continue <- TRUE
  while (continue){

    x.vec[counter+1] <- SolveFx(y.vec[counter], a, c)
    y.vec[counter+1] <- y.vec[counter]

    counter <- counter + 1

    x.vec[counter+1] <- x.vec[counter]
    y.vec[counter+1] <- SolveFy(x.vec[counter], b, c)

    counter <- counter + 1

    eps <- abs(x.vec[counter-1]-x.vec[counter]) + abs(y.vec[counter-1] - y.vec[counter])

    if (eps < 1e-4 | cycle == max.iter){
      continue <- FALSE
    }
    cycle <- cycle + 1
  }
```

```
  update.history <- data.frame(x = x.vec[1:counter], y=y.vec[1:counter])

  return(update.history)
}
```

b

Demonstrate the algorithm for the function $g(x, y) = x^2 + 2y^2 - x \cdot y$ for start values $x_0 = -3$ and $y_0 = -2$.

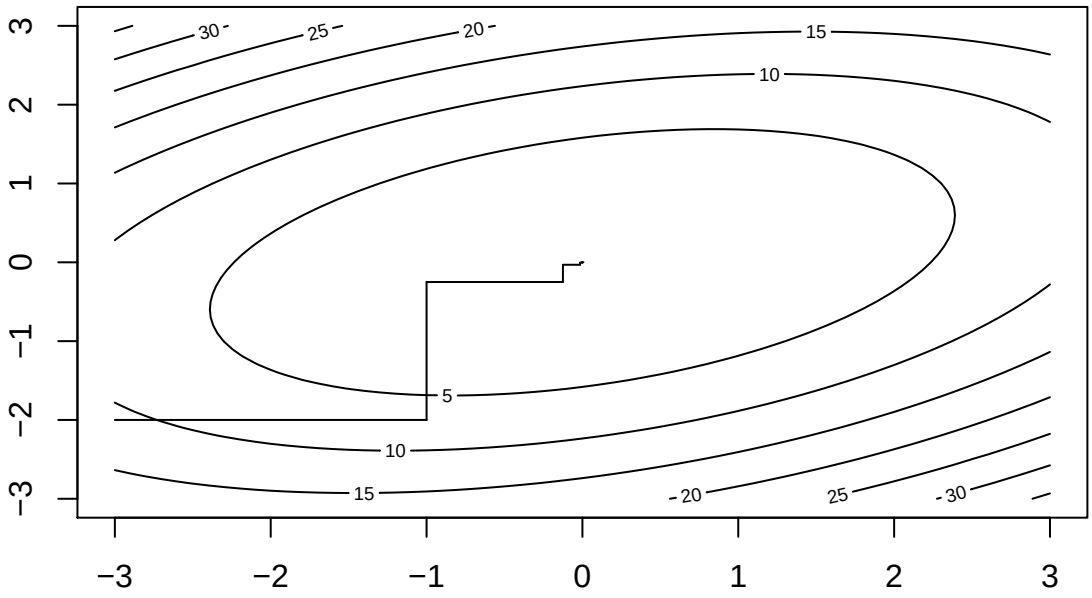
Make a contourplot of the function g. Add to the contourplot, using the function `segments`, a line between each subsequent update of the parameters.

Answer:

```
f <- function(x, y, a, b, c){
  a*x^2+b*y^2+c*x*y
}

values <- seq(-3, 3, by=0.1)
```

```
a <- 1
b <- 2
c <- -1

f.result <- outer(X = values, Y = values, FUN = f, a=a, b=b, c=c)
contour(values, values, f.result)

#result <- CoordinateDescent(-3, -2, 1, 2, -1)
result <- CoordinateDescent(x0=-3, y0=-2, a=a, b=b, c=c)
for (i in 1:(nrow(result)-1)){
  segments(x0 = result[i, 1], y0 = result[i, 2], x1 = result[i+1, 1], y1 = result[i+1, 2])
}
```

画线段



c

Also demonstrate the algorithm for the function $h(x, y) = 1.5x^2 + 2y^2 - 3x \cdot y$, for start values $x_0 = -2$ and $y_0 = -3$. Again make a contourplot with the steps.

What are the differences?

Answer:

```
a <- 1.5
b <- 2
c <- -3

f.result <- outer(X = values, Y = values, FUN = f, a=a, b=b, c=c)
contour(values, values, f.result)

#result <- CoordinateDescent(-3, -2, 1.5, 2, -1.5)
result <- CoordinateDescent(x0=-3, y0=-2, a=a, b=b, c=c)
for (i in 1:(nrow(result)-1)){
  segments(x0 = result[i, 1], y0 = result[i, 2], x1 = result[i+1, 1], y1 = result[i+1, 2])
}
```