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.

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.

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.

d

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

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()`.

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).

c

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

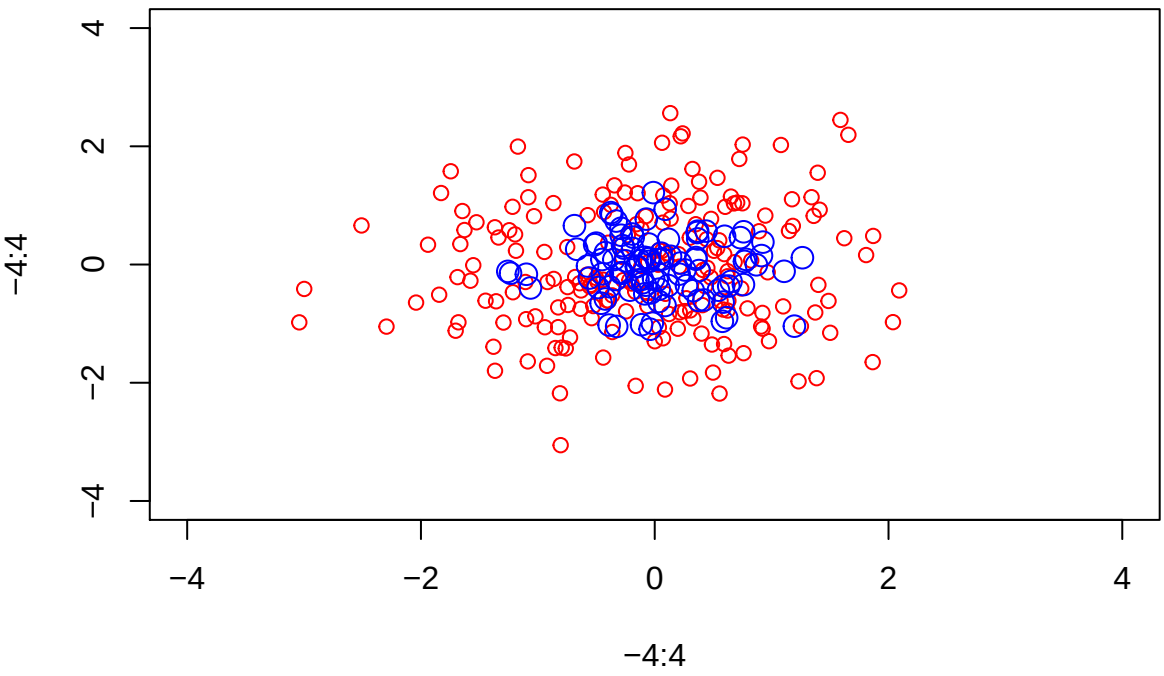
1.3 Adding points to a graph and customizing a graph

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)
```



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`).

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()`.

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).

e

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

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.

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?

c

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

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?

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?

b

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

c

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

d

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

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**?

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?

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`.

h

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

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.

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")`.

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.

c

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

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?

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` and assign it to `x` again.

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 $1/N$.

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.

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?

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)?

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.

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)`.

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.

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
```

b

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

c

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

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.

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.

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.

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.

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.

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?

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).

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?

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?

d

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

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.

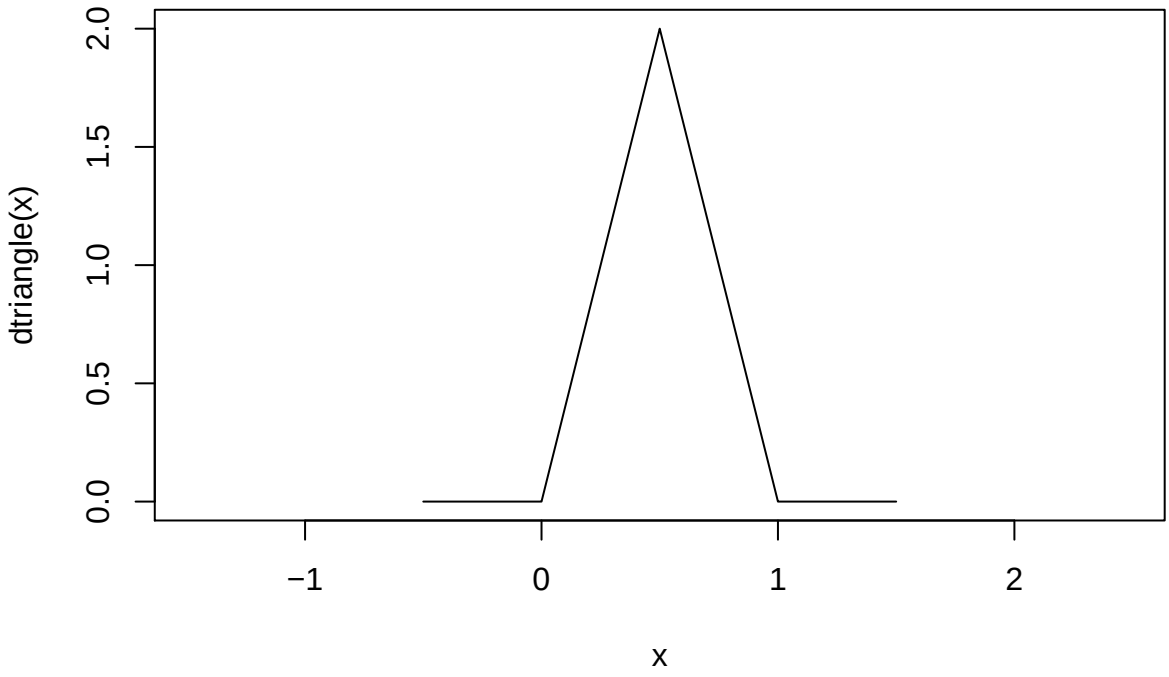
3.4 Sampling from a triangle

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.

```
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?

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.

c

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

d

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

3.5 Coordinate descent (For advanced students!)

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

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.

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?