Examination Advanced R Programming

Linköpings Universitet, IDA, Statistik

Course code and name:	732A94 Advanced R Programming
Date:	2018/10/23, 8–12
Teacher:	Krzysztof Bartoszek
Allowed aids:	The extra material is included in the zip file **exam_material.zip**
Grades:	A= $[18-20]$ points B= $[16-18)$ points C= $[12-16)$ points D= $[9-12)$ points E= $[8-9)$ points F= $[0-8)$ points
Instructions:	Write your answers in an R script file named **[your exam account].R** The R code should be complete and readable code, possible to run by copying directly into a script. Comment directly in the code whenever something needs to be explained or discussed. Follow the instructions carefully. There are **THREE** problems (with sub–questions) to solve.

Problem 1 (5p)

a) (2p) What is the difference between `<-` and `<<-`?
b) (1p) If you don't supply an explicit environment, where do `ls()` and `rm()` look?
c) (2p) What is the difference between the fields `Depends` and `Imports` in the `DESCRIPTION` file in an R package? What needs to be provided in the `importFrom()` statements in the `NAMESPACE` file?

Problem 2 (10p)

READ THE WHOLE QUESTION BEFORE STARTING TO IMPLEMENT! Remember that your functions should **ALWAYS** check for correctness of user input!

a) (3p) In this task you should use object oriented programming in S3 or RC to write code that simulates what should be taken off a ship so that more people can come aboard.

On 1 October 1939 without any hope for relief and running out of supplies the defenders of the Hel peninsula in Poland decided to surrender. However, a group of diehard sailors found that the pursuit cutter ORP Batory was still functional and decided to make a run to neutral Sweden. As fuel is very limited they need to remove as much as they can from the vessel.

The first task is to implement a function called `build_Batory()` that returns an object corresponding to the ship. The `build_Batory()` function should take two numeric arguments: **fuel** and **max_cargo_weight**. The object should contain a data structure that contains what is present on the ship (an entity present on the ship is described by its name (character) and weight (numeric)). Furthermore, the object should have a **cargo_weight** field, that is the total weight of all the present objects on it and the fuel. At creation the value of the **cargo_weight** field should be equal to **fuel**. Of course the weight of fuel cannot be greater, than **max_cargo_weight**.

```
## S3 and RC call to build_Batory() function
ORP_Batory <- build_Batory(fuel=1.5,max_cargo_weight=10)
```

b) (3p) Now implement a function called `equip_Batory()` to simulate what is stored on the ship. The function should take two arguments, the name of the object and the weight. This user provided data should then be remembered in the data structure that stores the ship's cargo. Run the function a number of times (say 50 times) to equip the ship. Remember to update the **cargo_weight** field. You decide yourself how to react when a object to store will make **cargo_weight** exceed **max_cargo_weight**. The names of the objects to store can be anything, including the same name for all the objects. The weights of the object should be drawn from the exponential distribution, function `rexp()` with rate equalling 5.

```
## S3 and RC call
ORP_Batory <-equip_Batory(ORP_Batory,"food",0.5)

## if using RC you may also call in this way
ORP_Batory$equip_Batory("food",0.5)
```

c) (3p) Now implement a function called `embark_sailor()`. The function should take as an argument the weight of the sailor. The function should check if the sailor can be embarked, i.e. the weight does not make **cargo_weight** exceed **max_cargo_weight**. If it does not, then use the `equip_Batory()` to add the sailor as "cargo". As name use `"sailor"`. If it does exceed, then remove objects from the cargo until the sailor can be embarked. An already embarked sailor cannot be removed to make space for another. Remember that there is also fuel on board that cannot be removed. If after removing all the cargo it is not possible to embark the sailor the function should return a comment and not embark. Run the function a number of times (say 10 times) to embark sailors on the ship. The weight should be taken from the uniform distribution `runif()` with `min=0.08` and `max=0.2`.

Provide some example calls to your code.

```
## S3 and RC call
ORP_Batory <-embark_sailor(ORP_Batory,0.15)

## if using RC you may also call in this way
ORP_Batory$embark_sailor(0.15)
```

d) (1p) Implement a plot **OR (NO NEED TO DO BOTH!)** print function for your ship's cargo. You are free to choose yourself how to report the content!

```
# Plotting and printing calls
plot(ORP_Batory); print(ORP_Batory)
```

Problem 3 (5p)

a) (3p) Multiplication of two matrices $\mathbf{A} \in \mathbb{R}^{n \times p}$ and $\mathbf{B} \in \mathbb{R}^{p \times m}$ is defined so that entry i, j of the product equals

$$\mathbf{V}[i,j] = \sum_{k=1}^{p} \mathbf{A}[i,k] \cdot \mathbf{B}[k,j],$$

where $\mathbf{V} = \mathbf{AB}$. Write your own function that takes as its input two matrices and returns the product. Remember that with matrices the order of multiplication matters. Do not forget that your function should check for correctness of input and react appropriately.
b) (1p) What is the complexity of your solution in terms of the number of required multiplication operations?
c) (1p) The same can be achieved using the `%*%` operator in R code, i.e. the product of two matrices is `A%*%B`. Implement a unit test that compares your implementation with direct R matrix multiplication.

Figure 1: Left: ORP Batory today as a museum ship at Polish Naval Museum in Gdynia, source: Maciek Hypś, Konflikty.pl . Right: map of the Baltic, source: Google maps. Klintehamn in Gotland was the end port of ORP Batory's escape.