

# Non-life — Assignment NL2

Niels Keizer\* and Robert Jan Sopers†

September 22, 2016

## 1 Simulating an insurance portfolio-App. A3

### Q1

How many bytes does it take to store  $1, \dots, 10, 1000, 100000$  logical values `TRUE/FALSE`?

We assume that  $1, \dots, 10$  means all the integers from 1 to 10. To how many bytes are needed in R, we use the function `object.size()`.

```
> for (n_values in c(1,2,3,4,5,6,7,8,9,10,1000,100000)){
+   hh <- rep(TRUE,n_values)
+   rr <- sample(c(TRUE,FALSE),n_values,repl=TRUE,prob=c(1,1))
+   af <- as.factor(rr)
+   print(c(n_values, object.size(hh), object.size(rr), object.size(af)))
+ }
```

[1]	1	48	48	464
[1]	2	48	48	464
[1]	3	56	56	528
[1]	4	56	56	528
[1]	5	72	72	544
[1]	6	72	72	488
[1]	7	72	72	544
[1]	8	72	72	544
[1]	9	88	88	560
[1]	10	88	88	560
[1]	1000	4040	4040	4512
[1]	100000	400040	400040	400512

The first column of the output is the length of the vector. The second column indicates the size in bytes of a vector filled with only `TRUE` values. The third with a random selection of `TRUE` and `FALSE`. The final column represents the size of the randomized vector, after it has been turned into a factor object.

---

\*Student number: 10910492

†Student number: 0629049

## Q2

To obtain the  $y$  vector, we first need to run the following code:

```
> n.obs <- 10000; set.seed(4)
> # n.obs <- 10000; set.seed(4) # Gebruik deze regel voor een grotere sample size.
> sx <- as.factor(sample(1:2, n.obs, repl=TRUE, prob=c(6,4)))
> jb <- as.factor(sample(1:3, n.obs, repl=TRUE, prob=c(3,2,1)))
> re.tp <- sample(1:9, n.obs, repl=TRUE, prob=c(.1,.05,.15,.15,.1,.05,.1,.1,.2))
> tp <- as.factor(c(1,2,3,1,2,3,1,2,3)[re.tp])
> re <- as.factor(c(1,1,1,2,2,2,3,3,3)[re.tp])
> mo <- 3 * sample(1:4, n.obs, repl=TRUE, prob=c(1,1,0,8))
> mu <- 0.05 * c(1,1.2)[sx] *
+           c(1,1,1)[jb] *
+           c(1,1.2,1.44)[re] *
+           1.2^(0:2)[tp] * mo/12
> y <- rpois(n.obs, mu)
> table(y)
y
  0    1    2    3
9276 702  20    2
```

Which is then inspected by calculating  $\text{mean}(y)$ ,  $\text{var}(y)$  and the overdispersion factor  $\text{var}(y)/\text{mean}(y)$ .

```
> cbind(mean=mean(y),variance=var(y),phi=var(y)/mean(y))
      mean variance      phi
[1,] 0.0748 0.0744124 0.9948182
```

The overdispersion factor is smaller than 1. This is possible because we are looking at a relatively small sample, with low probabilities. If we would take a much larger sample, the value would be larger than 1. We check this by running the same code, but with a sample 100 times larger. This gives a result with an overdispersion factor larger than 1.

```
> table(y)
y
  0    1    2    3    4
931128 66053 2734  82    3
> cbind(mean=mean(y),variance=var(y),phi=var(y)/mean(y))
      mean variance      phi
[1,] 0.071779 0.07262285 1.011756
```

## Q3

We create a dataframe by using the function `aggregate()`.

```
> aggr <- aggregate(list(Expo=mo/12,nCl=y,nPol=1), list(Jb=jb,Tp=tp,Re=re,Sx=sx), sum)
```

Then we compare the sizes.

```

> object.size(aggr)
5336 bytes
> object.size(mo)
80040 bytes
> object.size(y)
40040 bytes
> object.size(jb) + object.size(tp) + object.size(re) + object.size(sx)
162240 bytes

```

The amount of memory gained is equal to  $80040 + 40040 + 162240 - 5336 = 276984$  bytes.

## Q4

According to MART Sec. 3.9.3, the maximum likelihood estimate  $\hat{\lambda}_{3,3,3,2}$  is equal to the number of claims divided by the exposure.

```

> aggr[54,]
   Jb Tp Re Sx   Expo nCl nPol
54  3  3  3  2 115.75  13  130
> lambda3332 <- aggr$nCl[54]/aggr$Expo[54]
> lambda3332
[1] 0.112311

```

In the first command, we show that observation 54 contains the desired aggregated values to calculate the estimate, which is then determined at 0.112.

## 2 Exploring the automobile portfolio of Sec. 9.5

First we execute the following code in R to generate the portfolio.

```

> rm(list=ls(all=TRUE))
> n <- scan(n=54) ## read 54 numbers into vector n
1:  1  8 10  8  5 11 14 12 11 10  5 12 13 12 15 13 12 24
19: 12 11  6  8 16 19 28 11 14  4 12  8 18  3 17  6 11 18
37: 12  3 10 18 10 13 12 31 16 16 13 14  8 19 20  9 23 27
Read 54 items
> expo <- scan(n=54) ## the number of policies
1:  10 22 30 11 15 20 25 25 23 28 19 22 19 21 19 16 18 29
19: 25 18 20 13 26 21 27 14 16 11 23 26 29 13 26 13 17 27
37: 20 18 20 29 27 24 23 26 18 25 17 29 11 24 16 11 22 29
Read 54 items
> expo <- 7 * expo ## each policy is in force during a 7-year period
> sex <- gl(2,27); region <- gl(3, 9, 54); type <- gl(3, 3, 54); job <- gl(3, 1, 54)

```

## Q5

We are asked to comment on the difference between to lines of R code.

```
> str(type)
Factor w/ 3 levels "1","2","3": 1 1 1 2 2 2 3 3 3 1 ...
> str(rep(1:3, each=3, len=54))
int [1:54] 1 1 1 2 2 2 3 3 3 1 ...
```

The `str` function compactly displays the structure of an arbitrary R object. `type` contains a **Factor** object, with 3 ordered levels (or categories), and a list of integers which indicate which element is at that position. `rep(1:3, each=3, len=54)` creates a vector of integers of three ones, three twos and three threes, repeated to a length of 54.