Djillali Liabes University of Sidi Bel Abbes
Faculty of Exact Sciences
Department of Computer Sciences

Module : Modélisation et Simulation
1st year of Master's degeree in
Networks,Information Systems & Security(RSSI)
2021/2022

TP:01 : Probabilité avec R

Students:
Hadjazi M.Hisham
Amuer Wassim
Group: 02 / RSSI

Instructor:
Dr. S.Benbekriti

A paper submitted in fulfilment of the requirements for the
Modélisation et Simulation TP-01

November 12, 2021

Contents

List of Figures

Chapter 1

Solutions of Fiche TP-01

Notes regarding this solution :

This solution and the executions of the code in it was done in the following machine :

- *PC* : Lenovo IdeaPad S210 8GB
- *OS* : Linux Mint 20.2 Cinnamon Kernel v.5.4.0-88
- *IDE* : RStudio 2021.09.0 Build 351
- *R Version* : 3.6.1 (2019-07-05)

During This TP we were solving the questions according to our understanding of them, we had some difficulties in understanding some of them for example while we ere solving question #3 , it asks to represent results as a histogram to the proportion of each color while a histogram is used to represent the distribution of the population or results which didn't make scene to us so we used a barchart as we saw it was more fit, and we used a histogram in question #6 to showcase where we think it is more appropriate to use it (of course to our limited understandings).

Also during the solutions we tried to get the data from experiments and not as facts from the question as it is much faster to calculate probability with formulas and simple calculations, but as the objective of this TP we thought that for our data we had to calculate the probability from experiments and simulations of the problems and we get results close to the calculations as the number of experiments increase.

We also included an R script file that has our code to ease the testing. The code we wrote has comments to explain what each piece of code does, we also included .tex in the document.

1.1 Exercise 1

Que font ces lignes de commande ? Indiquez ce que R vous retourne.

```
x<-c(0,1,2,3)
probax<-c(3/15,4/15,5/15,3/15)
frex<-cumsum(probax)
d<-stepfun(x,c(0,frex))
plot(d,vertical=FALSE)
```

The execution of the code gives the following.

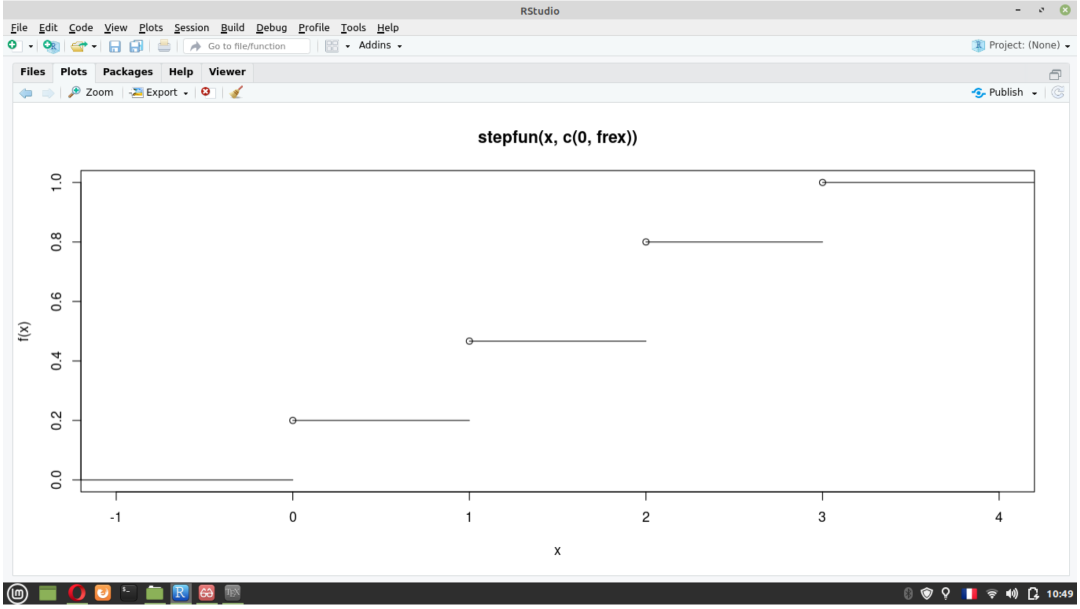


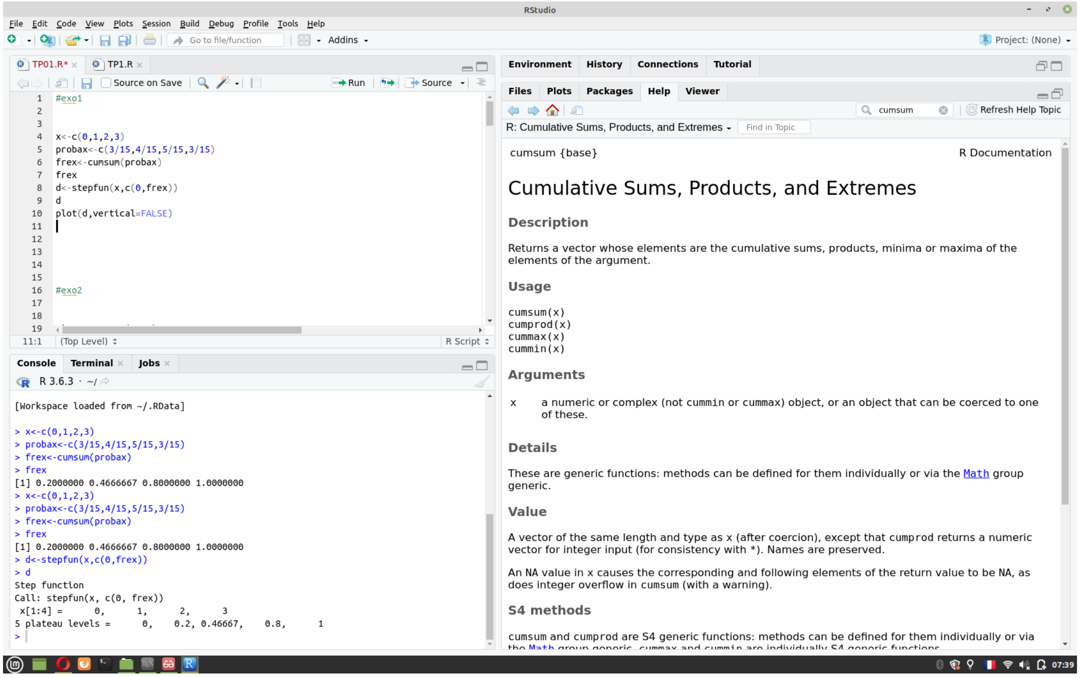
FIGURE 1.1: Plot 1

```
x<-c(0,1,2,3)
probax<-c(3/15,4/15,5/15,3/15)
```

here we are just creating two vectors the firs **x** has the vales from 0 to 3. and the second vector **probax** has the values of probability of each of the numbers of vector **x**.

```
frex<-cumsum(probax)
```

Now we are creating a vector with each element has the sum of the previous elements.



```
#exo1


x<-c(0,1,2,3)
probax<-c(3/15,4/15,5/15,3/15)
frex<-cumsum(probax)
frex
d<-stepfun(x,c(0,frex))
d
plot(d,vertical=FALSE)





#exo2
```

```
[Workspace loaded from ~/.RData]

> x<-c(0,1,2,3)
> probax<-c(3/15,4/15,5/15,3/15)
> frex<-cumsum(probax)
> frex
[1] 0.2000000 0.4666667 0.8000000 1.0000000
> x<-c(0,1,2,3)
> probax<-c(3/15,4/15,5/15,3/15)
> frex<-cumsum(probax)
> frex
[1] 0.2000000 0.4666667 0.8000000 1.0000000
> d<-stepfun(x,c(0,frex))
> d
Step function
Call: stepfun(x, c(0, frex))
 x[1:4] =      0,      1,      2,      3
5 plateau levels =      0,    0.2, 0.46667,    0.8,      1
>
```

R: Cumulative Sums, Products, and Extremes

cumsum {base} R Documentation

Cumulative Sums, Products, and Extremes

Description

Returns a vector whose elements are the cumulative sums, products, minima or maxima of the elements of the argument.

Usage

```
cumsum(x)
cumprod(x)
cummax(x)
cummin(x)
```

Arguments

x a numeric or complex (not cummin or cummax) object, or an object that can be coerced to one of these.

Details

These are generic functions: methods can be defined for them individually or via the Math group generic.

Value

A vector of the same length and type as x (after coercion), except that cumprod returns a numeric vector for integer input (for consistency with *). Names are preserved.

An NA value in x causes the corresponding and following elements of the return value to be NA, as does integer overflow in cumsum (with a warning).

S4 methods

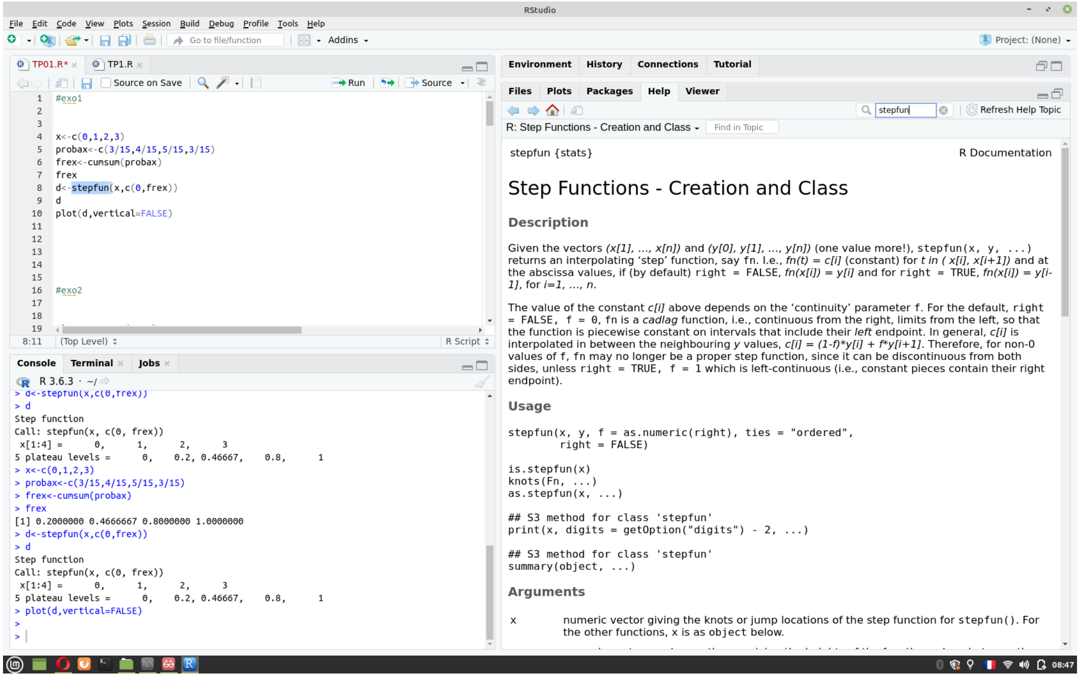
cumsum and cumprod are S4 generic functions: methods can be defined for them individually or via the Math group generic. cummax and cummin are individually S4 generic functions.

FIGURE 1.2: cumsum function

```
d<−stepfun(x,c(0,frex))
```

The function **stepfun** is to Compute a left- or right-continuous step function and a step function is sometimes called greatest integer function or a floor function and what it does is simply round the members to the greatest small or big integer before it or after it as in left or right to the value.

However here we are giving the plateaus or the floors as a parameter from our **frex** vector. as we are creating our own step function.



```
#exo1


x<-c(0,1,2,3)
probax<-c(3/15,4/15,5/15,3/15)
frex<-cumsum(probax)
frex
d<-stepfun(x,c(0,frex))
d
plot(d,vertical=FALSE)





#exo2
```

```
> d<-stepfun(x,c(0,frex))
> d
Step function
Call: stepfun(x, c(0, frex))
 x[1:4] =      0,      1,      2,      3
5 plateau levels =      0,    0.2, 0.46667,    0.8,      1
> x<-c(0,1,2,3)
> probax<-c(3/15,4/15,5/15,3/15)
> frex<-cumsum(probax)
> frex
[1] 0.2000000 0.4666667 0.8000000 1.0000000
> d<-stepfun(x,c(0,frex))
> d
Step function
Call: stepfun(x, c(0, frex))
 x[1:4] =      0,      1,      2,      3
5 plateau levels =      0,    0.2, 0.46667,    0.8,      1
> plot(d,vertical=FALSE)
>
> 
```

stepfun {stats} R Documentation

Step Functions - Creation and Class

Description

Given the vectors *(x[1], ..., x[n])* and *(y[0], y[1], ..., y[n])* (one value more!), `stepfun(x, y, ...)` returns an interpolating 'step' function, say `fn`. I.e., *fn(t) = c[i]* (constant) for *t in (x[i], x[i+1])* and at the abscissa values, if (by default) `right = FALSE`, *fn(x[i]) = y[i]* and for `right = TRUE`, *fn(x[i]) = y[i-1]*, for *i=1, ..., n*.

The value of the constant *c[i]* above depends on the 'continuity' parameter `f`. For the default, `right = FALSE, f = 0`, `fn` is a *cadlag* function, i.e., continuous from the right, limits from the left, so that the function is piecewise constant on intervals that include their *left* endpoint. In general, *c[i]* is interpolated in between the neighbouring *y* values, *c[i] = (1-f)*y[i] + f*y[i+1]*. Therefore, for non-0 values of `f`, `fn` may no longer be a proper step function, since it can be discontinuous from both sides, unless `right = TRUE, f = 1` which is left-continuous (i.e., constant pieces contain their right endpoint).

Usage

```
stepfun(x, y, f = as.numeric(right), ties = "ordered",
        right = FALSE)

is.stepfun(x)
knots(Fn, ...)
as.stepfun(x, ...)

## S3 method for class 'stepfun'
print(x, digits = getOption("digits") - 2, ...)

## S3 method for class 'stepfun'
summary(object, ...)
```

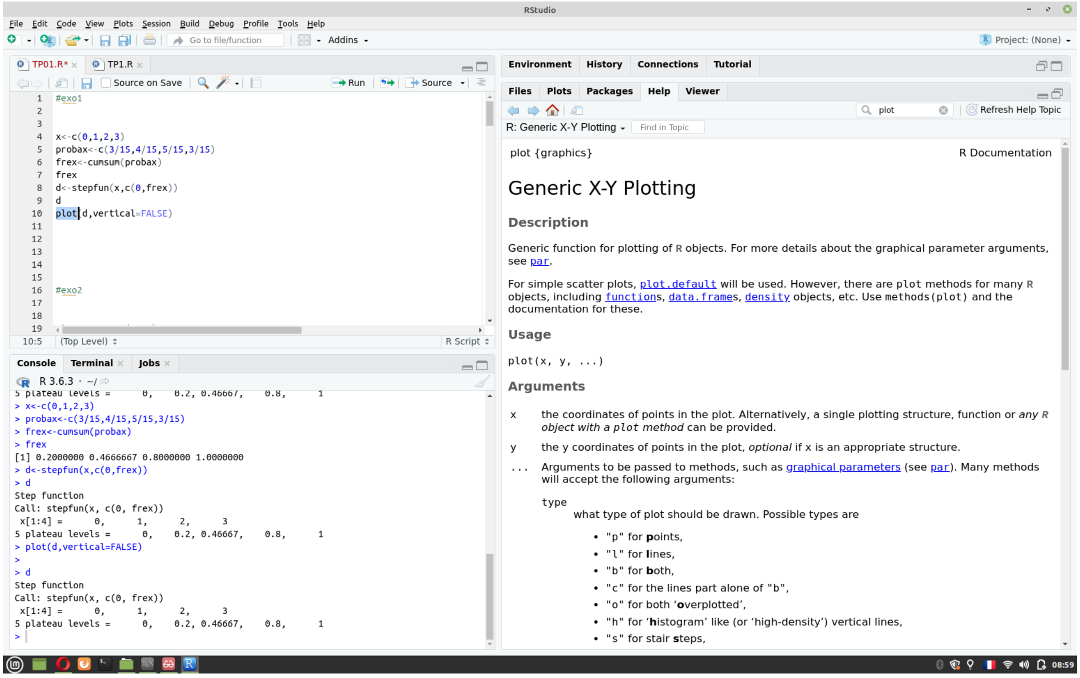
Arguments

x numeric vector giving the knots or jump locations of the step function for `stepfun()`. For the other functions, `x` is as `object` below.

FIGURE 1.3: stepfun function

```
plot(d, vertical=FALSE)
```

Finally we are plotting our steps that we created in **d** in the plot function.



```
#exo1


x<-c(0,1,2,3)
probax<-c(3/15,4/15,5/15,3/15)
frex<-cumsum(probax)
frex
d<-stepfun(x,c(0,frex))
d
plot(d,vertical=FALSE)





#exo2
```

```
5 plateau levels =      0,    0.2, 0.46667,    0.8,      1
> x<-c(0,1,2,3)
> probax<-c(3/15,4/15,5/15,3/15)
> frex<-cumsum(probax)
> frex
[1] 0.2000000 0.4666667 0.8000000 1.0000000
> d<-stepfun(x,c(0,frex))
> d
Step function
Call: stepfun(x, c(0, frex))
 x[1:4] =      0,      1,      2,      3
5 plateau levels =      0,    0.2, 0.46667,    0.8,      1
> plot(d,vertical=FALSE)
>
> d
Step function
Call: stepfun(x, c(0, frex))
 x[1:4] =      0,      1,      2,      3
5 plateau levels =      0,    0.2, 0.46667,    0.8,      1
> 
```

plot {graphics} R Documentation

Generic X-Y Plotting

Description

Generic function for plotting of R objects. For more details about the graphical parameter arguments, see par.

For simple scatter plots, `plot.default` will be used. However, there are `plot` methods for many R objects, including `function`s, `data.frame`s, `density` objects, etc. Use `methods(plot)` and the documentation for these.

Usage

```
plot(x, y, ...)
```

Arguments

x the coordinates of points in the plot. Alternatively, a single plotting structure, function or *any R object with a `plot` method* can be provided.

y the y coordinates of points in the plot, *optional* if `x` is an appropriate structure.

`...` Arguments to be passed to methods, such as graphical parameters (see par). Many methods will accept the following arguments:

`type`
what type of plot should be drawn. Possible types are

- `"p"` for **p**oints,
- `"l"` for **l**ines,
- `"b"` for **b**oth,
- `"c"` for the lines part alone of `"b"`,
- `"o"` for both '**o**verplotted',
- `"h"` for '**h**istogram' like (or 'high-density') vertical lines,
- `"s"` for stair **s**teps,

FIGURE 1.4: stepfun function

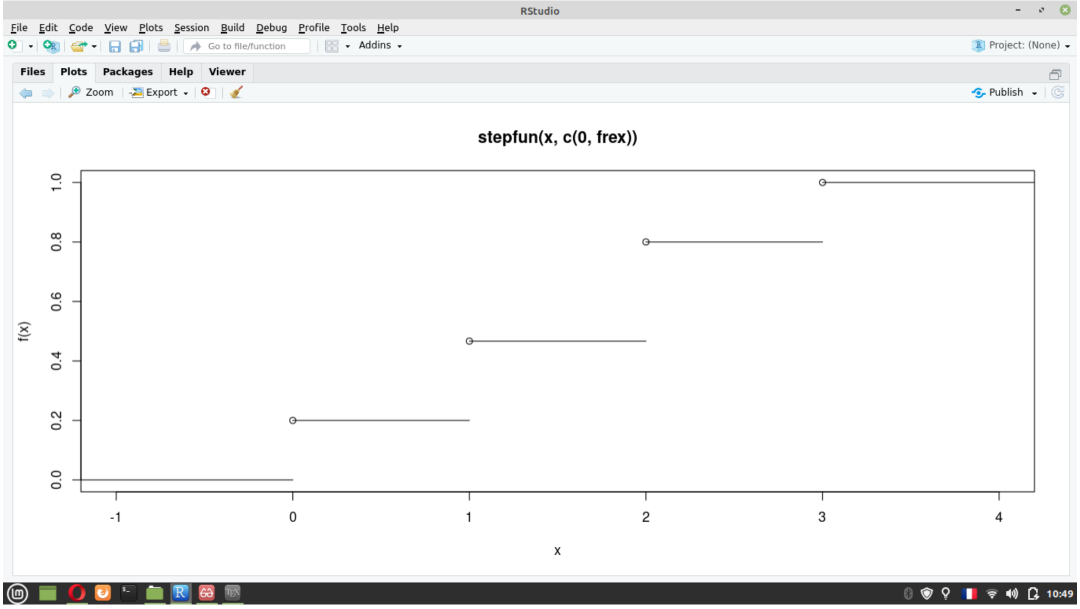


FIGURE 1.5: Plot 1

1.2 Exercise 2

Que font ces lignes de commande ? Indiquez ce que R vous retourne.

```
simu <- rnorm(8000)
{hist(simu, prob=T,
main="Histogramme␣de␣8000␣tirages")}
curve(dnorm(x), add=T)
```

The execution of the entire code gives the following.

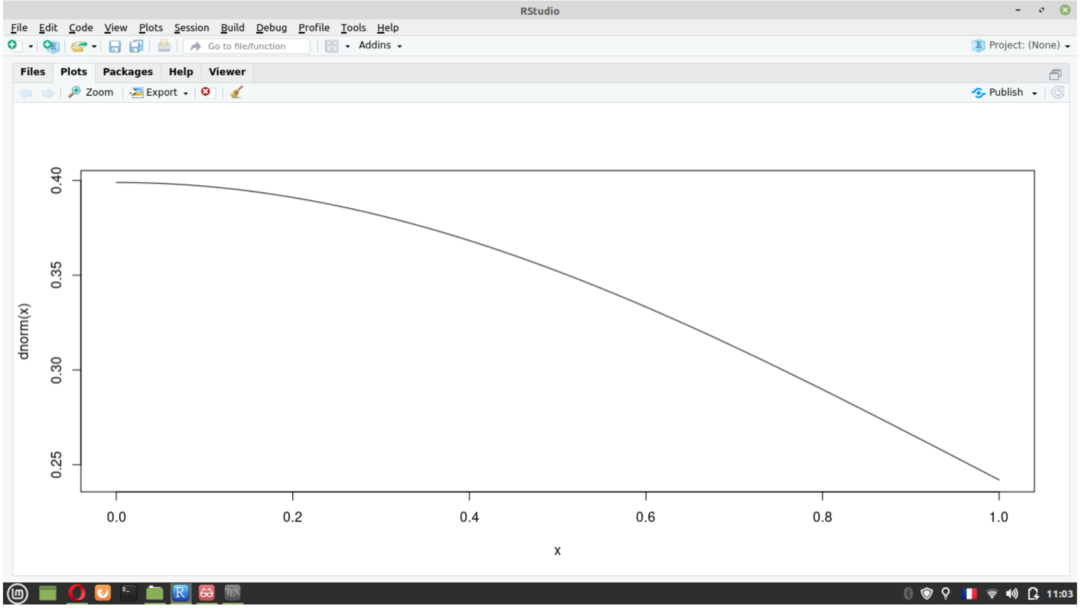
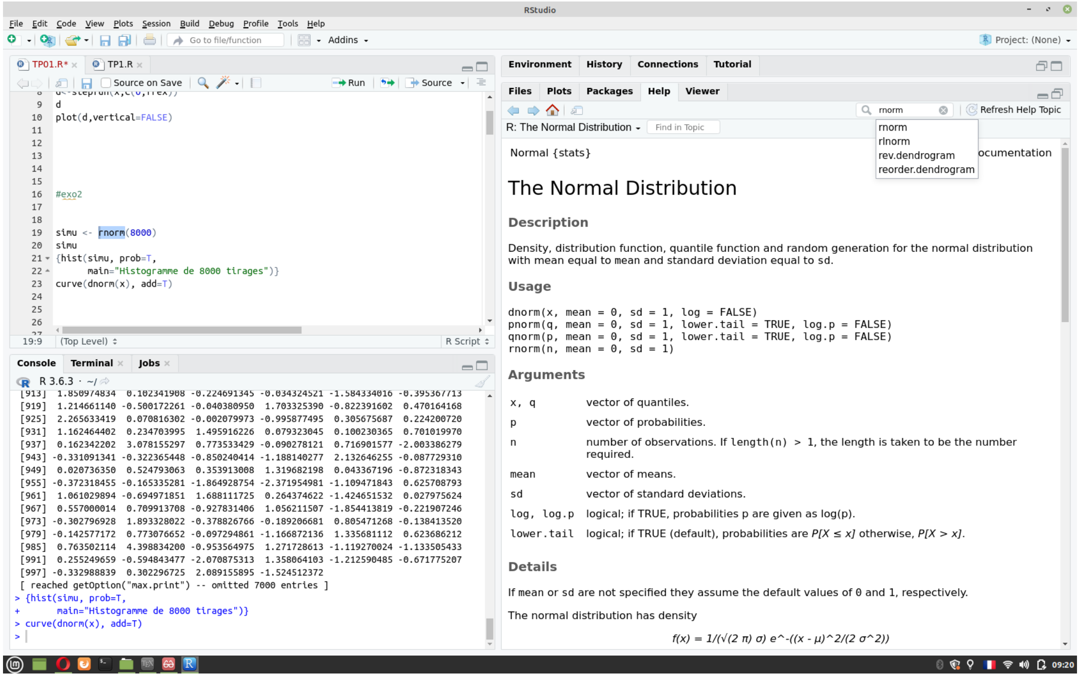


FIGURE 1.6: Plot 2

```
simu <- rnorm(8000)
```

We are using the **rnorm** function to generate random numbers according to the normal distribution, we are giving the size of **8000** as a parameter.



```
d<-stepfun(x,c(0,Frex))
d
plot(d,vertical=FALSE)





#exo2


simu <- rnorm(8000)
simu
{hist(simu, prob=T,
      main="Histogramme de 8000 tirages")}
curve(dnorm(x), add=T)
```

```
 [913]  1.850974834  0.102341908 -0.224691345 -0.034324521 -1.584334016 -0.395367713
 [919]  1.214661140 -0.500172261 -0.040380950  1.703325390 -0.822391602  0.470164168
 [925]  2.265633419  0.070816302 -0.002079973 -0.995877495  0.305675687  0.224200720
 [931]  1.162464402  0.234703995  1.495916226  0.079323045  0.100230365  0.701019970
 [937]  0.162342202  3.078155297  0.773533429 -0.090278121  0.716901577 -2.003386279
 [943] -0.331091341 -0.322365448 -0.850240414 -1.188140277  2.132646255 -0.087729310
 [949]  0.020736350  0.524793063  0.353913008  1.319682198  0.043367196 -0.872318343
 [955] -0.372318455 -0.165335281 -1.864928754 -2.371954981 -1.109471843  0.625708793
 [961]  1.061029894 -0.694971851  1.688111725  0.264374622 -1.424651532  0.027975624
 [967]  0.557000014  0.709913708 -0.927831406  1.056211507 -1.854413819 -0.221907246
 [973] -0.302796928  1.893328022 -0.378826766 -0.189206681  0.805471268 -0.138413520
 [979] -0.142577172  0.773076652 -0.097294861 -1.166872136  1.335681112  0.623686212
 [985]  0.763502114  4.398834200 -0.953564975  1.271728613 -1.119270024 -1.133505433
 [991]  0.255249659 -0.594843477 -2.070875313  1.358064103 -1.212590485 -0.671775207
 [997] -0.332988839  0.302296725  2.089155895 -1.524512372
 [ reached getOption("max.print") -- omitted 7000 entries ]
> {hist(simu, prob=T,
+       main="Histogramme de 8000 tirages")}
> curve(dnorm(x), add=T)
> 
```

Normal {stats} ocumentation

The Normal Distribution

Description

Density, distribution function, quantile function and random generation for the normal distribution with mean equal to mean and standard deviation equal to sd.

Usage

```
dnorm(x, mean = 0, sd = 1, log = FALSE)
pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
rnorm(n, mean = 0, sd = 1)
```

Arguments

x, q	vector of quantiles.
p	vector of probabilities.
n	number of observations. If length(n) > 1, the length is taken to be the number required.
mean	vector of means.
sd	vector of standard deviations.
log, log.p	logical; if TRUE, probabilities p are given as log(p).
lower.tail	logical; if TRUE (default), probabilities are *P[X ≤ x]* otherwise, *P[X > x]*.

Details

If mean or sd are not specified they assume the default values of 0 and 1, respectively.

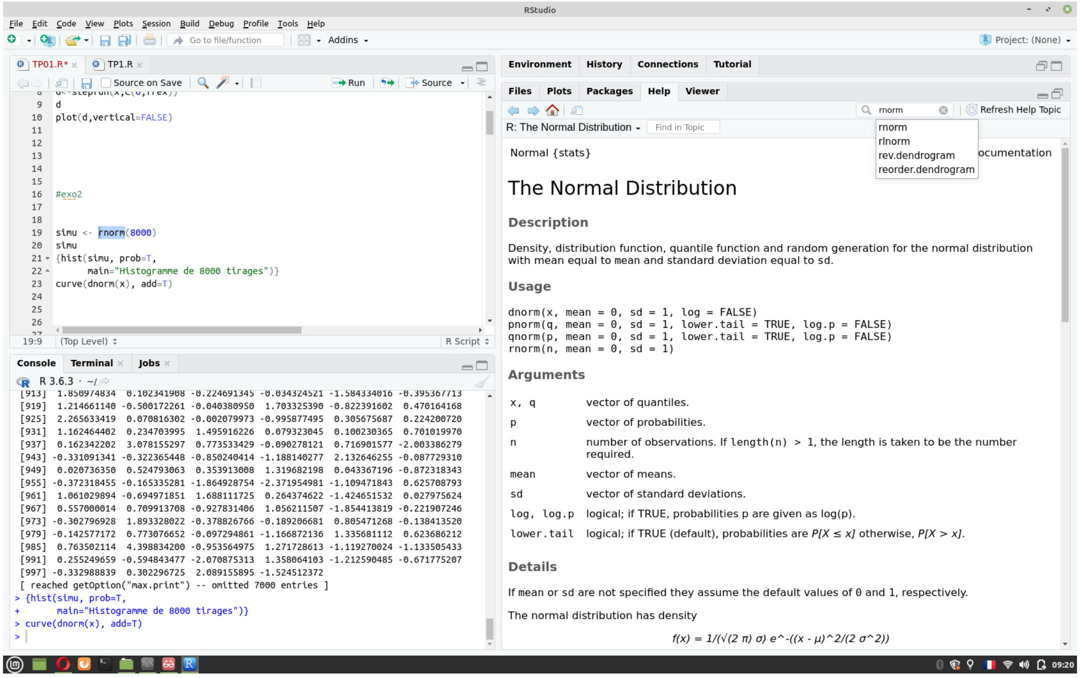
The normal distribution has density

f(x) = 1/(√(2 π) σ) e^-((x - μ)^2/(2 σ^2))

FIGURE 1.7: Plot 2

```
{hist(simu, prob=T,
main="Histogramme␣de␣8000␣tirages")}
curve(dnorm(x), add=T)
```

Here we are drawing a histogram to showcase the density in our sample of **simu**, The parameter **prob=T** is used to give the true histogram so it has a relative frequency density scale. main is just to give a title to our histogram and the with the help of the parameters **curve** and **dnorm** we can draw a curve over our histogram.



```
d
plot(d,vertical=FALSE)




#exo2


simu <- rnorm(8000)
simu
{hist(simu, prob=T,
      main="Histogramme de 8000 tirages")}
curve(dnorm(x), add=T)
```

```
[913]  1.850974834  0.102341908 -0.224691345 -0.034324521 -1.584334016 -0.395367713
[919]  1.214661140 -0.500172261 -0.040380950  1.703325390 -0.822391602  0.470164168
[925]  2.265633419  0.070816302 -0.002079973 -0.995877495  0.305675687  0.224200720
[931]  1.162464402  0.234703995  1.495916226  0.079323045  0.100230365  0.701019970
[937]  0.162342202  3.078155297  0.773533429 -0.090278121  0.716901577 -2.003386279
[943] -0.331091341 -0.322365448 -0.850240414 -1.188140277  2.132646255 -0.087729310
[949]  0.020736350  0.524793063  0.353913008  1.319682198  0.043367196 -0.872318343
[955] -0.372318455 -0.165335281 -1.864928754 -2.371954981 -1.109471843  0.625708793
[961]  1.061029894 -0.694971851  1.688111725  0.264374622 -1.424651532  0.027975624
[967]  0.557000014  0.709913708 -0.927831406  1.056211507 -1.854413819 -0.221907246
[973] -0.302796928  1.893328022 -0.378826766 -0.189206681  0.805471268 -0.138413520
[979] -0.142577172  0.773076652 -0.097294861 -1.166872136  1.335681112  0.623686212
[985]  0.763502114  4.398834200 -0.953564975  1.271728613 -1.119270024 -1.133505433
[991]  0.255249659 -0.594843477 -2.070875313  1.358064103 -1.212590485 -0.671775207
[997] -0.332988839  0.302296725  2.089155895 -1.524512372
 [ reached getOption("max.print") -- omitted 7000 entries ]
> {hist(simu, prob=T,
+       main="Histogramme de 8000 tirages")}
> curve(dnorm(x), add=T)
> 
```

Normal {stats} ocumentation

The Normal Distribution

Description

Density, distribution function, quantile function and random generation for the normal distribution with mean equal to mean and standard deviation equal to sd.

Usage

```
dnorm(x, mean = 0, sd = 1, log = FALSE)
pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
rnorm(n, mean = 0, sd = 1)
```

Arguments

x, q	vector of quantiles.
p	vector of probabilities.
n	number of observations. If length(n) > 1, the length is taken to be the number required.
mean	vector of means.
sd	vector of standard deviations.
log, log.p	logical; if TRUE, probabilities p are given as log(p).
lower.tail	logical; if TRUE (default), probabilities are *P[X ≤ x]* otherwise, *P[X > x]*.

Details

If mean or sd are not specified they assume the default values of 0 and 1, respectively.

The normal distribution has density

f(x) = 1/(√(2 π) σ) e^-((x - μ)^2/(2 σ^2))

FIGURE 1.8: Plot 2

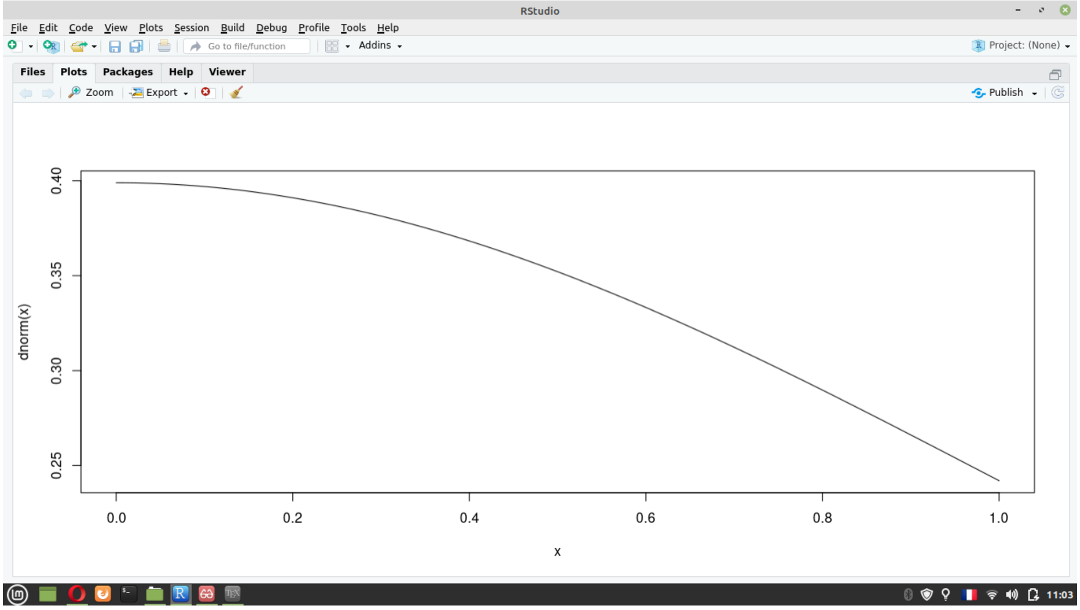
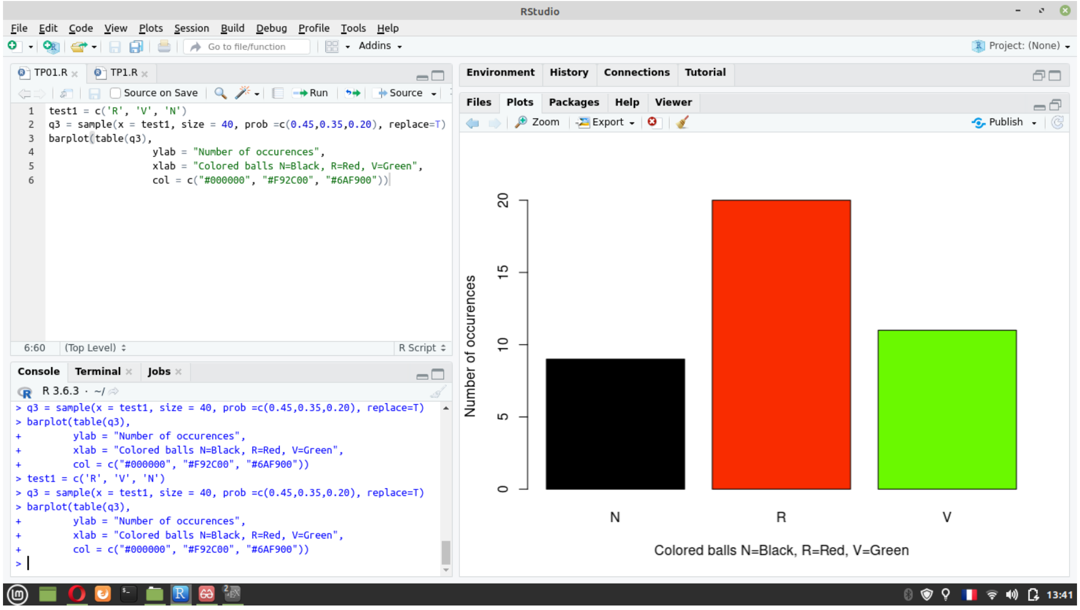


FIGURE 1.9: Histogram

1.3 Exercise 3

Simuler sous R, 40 tirages avec remise dans une urne dont 45% de boules sont rouges (R), 35% de boules sont vertes (V) et 20% de boules sont noires (N). Afficher la proportion de chaque couleur de façon graphique (histogramme).

```
# Creating a vector congaing the 3 colors R,V,N

test1 = c('R', 'V', 'N')

# Running a sample function with size 40
# and probability of each color

q3 = sample(x = test1,
            size = 40,
            prob =c(0.45,0.35,0.20),
            replace=T)

# Plotting the result of each color in a barplot
barplot(table(q3),
        ylab = "Number␣of␣occurences",
        xlab = "Colored␣balls␣N=Black,␣R=Red,␣V=Green",
        col = c("#000000", "#F92C00", "#6AF900"))
```



```
test1 = c('R', 'V', 'N')
q3 = sample(x = test1, size = 40, prob =c(0.45,0.35,0.20), replace=T)
barplot(table(q3),
                    ylab = "Number of occurences",
                    xlab = "Colored balls N=Black, R=Red, V=Green",
                    col = c("#000000", "#F92C00", "#6AF900"))
```

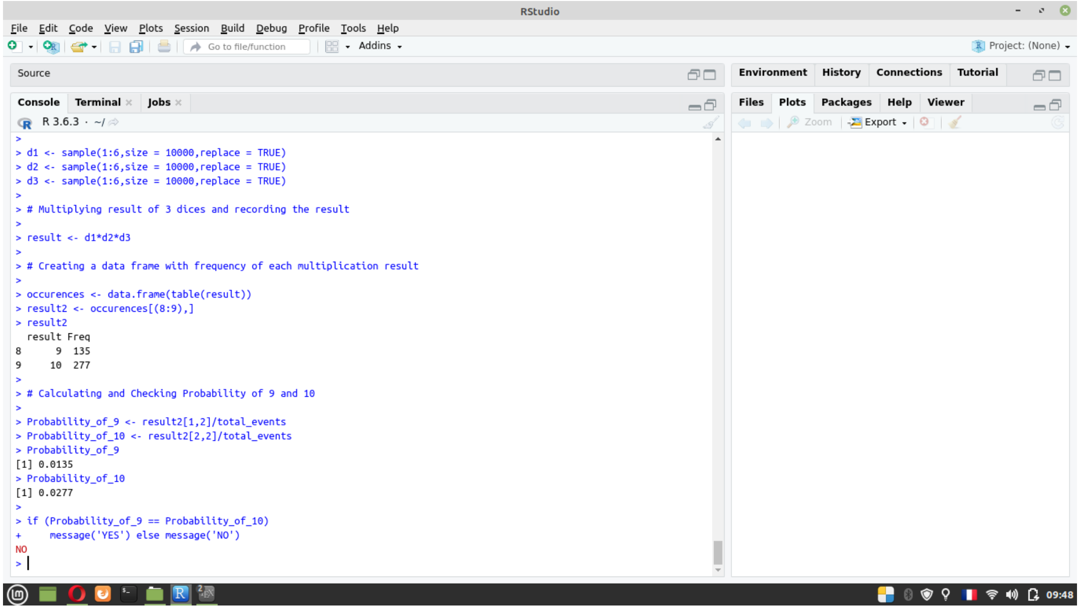
```
> q3 = sample(x = test1, size = 40, prob =c(0.45,0.35,0.20), replace=T)
> barplot(table(q3),
+         ylab = "Number of occurences",
+         xlab = "Colored balls N=Black, R=Red, V=Green",
+         col = c("#000000", "#F92C00", "#6AF900"))
> test1 = c('R', 'V', 'N')
> q3 = sample(x = test1, size = 40, prob =c(0.45,0.35,0.20), replace=T)
> barplot(table(q3),
+         ylab = "Number of occurences",
+         xlab = "Colored balls N=Black, R=Red, V=Green",
+         col = c("#000000", "#F92C00", "#6AF900"))
> 
```

FIGURE 1.10: Plot 3

1.4 Exercise 4

Écrire sous R le programme qui permet de lancer trois dés équilibrés à 6 faces et noter la multiplication des trois faces obtenues. A-t-on autant de chances d'obtenir 9 que 10 (Si oui afficher un message "OUI") ?

```r
# Number of events
total_events=10000

# d1,d2,d3 are three dices that are rolled 10000 times each

d1 <- sample(1:6,size = 10000,replace = TRUE)
d2 <- sample(1:6,size = 10000,replace = TRUE)
d3 <- sample(1:6,size = 10000,replace = TRUE)

# Multiplying result of 3 dices and recording the result

result <- d1*d2*d3

# Creating a data frame with frequency of
# each multiplication result

occurences <- data.frame(table(result))
result2 <- occurences[(8:9),]
result2

# Calculating and Checking Probability of 9 and 10

Probability_of_9 <- result2[1,2]/total_events
Probability_of_10 <- result2[2,2]/total_events
Probability_of_9
Probability_of_10

if (Probability_of_9 == Probability_of_10)
  message('YES') else message('NO')
```



```
>
> d1 <- sample(1:6,size = 10000,replace = TRUE)
> d2 <- sample(1:6,size = 10000,replace = TRUE)
> d3 <- sample(1:6,size = 10000,replace = TRUE)
>
> # Multiplying result of 3 dices and recording the result
>
> result <- d1*d2*d3
>
> # Creating a data frame with frequency of each multiplication result
>
> occurences <- data.frame(table(result))
> result2 <- occurences[(8:9),]
> result2
  result Freq
8      9  135
9     10  277
>
> # Calculating and Checking Probability of 9 and 10
>
> Probability_of_9 <- result2[1,2]/total_events
> Probability_of_10 <- result2[2,2]/total_events
> Probability_of_9
[1] 0.0135
> Probability_of_10
[1] 0.0277
>
> if (Probability_of_9 == Probability_of_10)
+     message('YES') else message('NO')
NO
> |
```

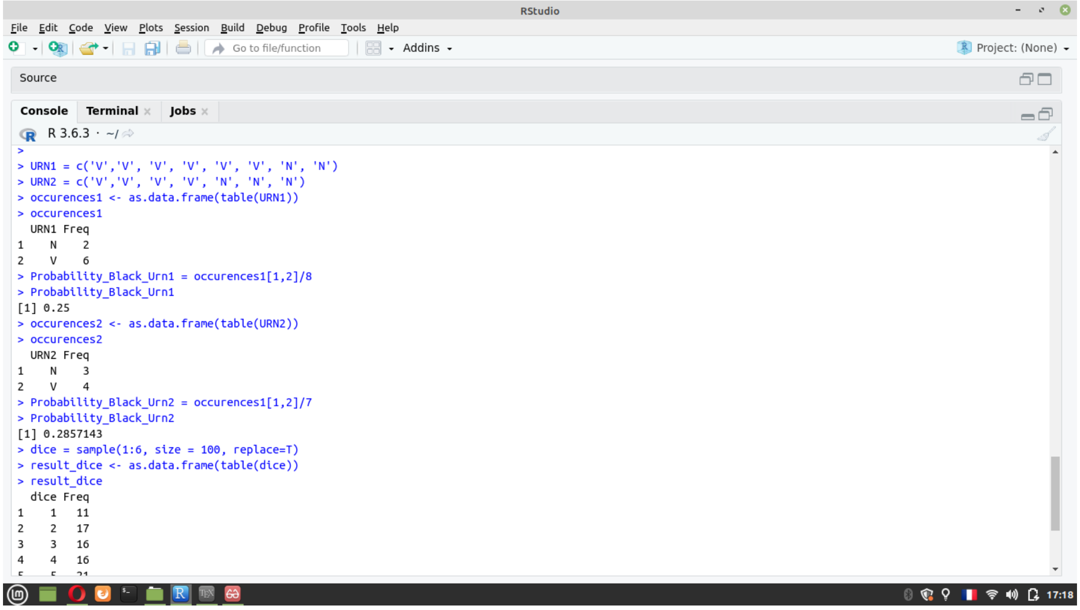
FIGURE 1.11: dice roll function

1.5 Exercise 5

Une urne1 contient 6 boules vertes et 2 noires. Une urne2 contient 4 vertes et 3 noires. On jette un dé équilibré à 6 faces, si le résultat est 1, on tire une boule dans l'urne1 sinon on tire une boule dans l'urne2. On répète l'expérience 100 fois, écrire sous R le programme qui calcule la probabilité d'obtenir une boule noire.

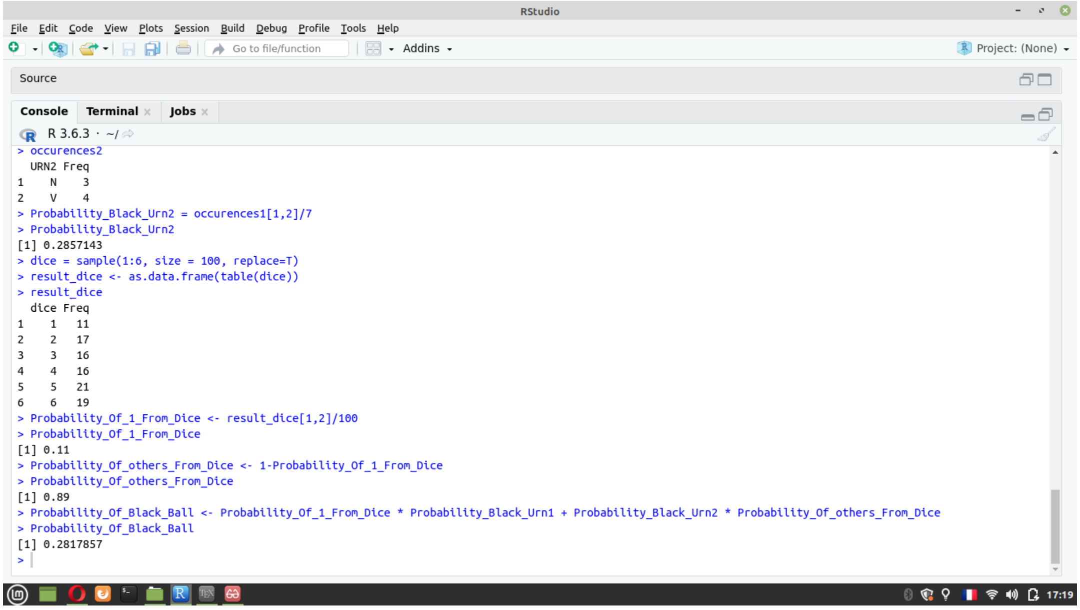
```
# We create our Urns with the colored
# balls inside each one of them
URN1 = c('V','V', 'V', 'V', 'V', 'V', 'N', 'N')
URN2 = c('V','V', 'V', 'V', 'N', 'N', 'N')

# We calculate the frequency of each color inside the Urn1
occurences1 <- as.data.frame(table(URN1))
occurences1
Probability_Black_Urn1 = occurences1[1,2]/8
Probability_Black_Urn1

# We calculate the frequency of each color inside the Urn1
occurences2 <- as.data.frame(table(URN2))
occurences2
Probability_Black_Urn2 = occurences1[1,2]/7
Probability_Black_Urn2

# We roll our dice 100 times and record the results
dice = sample(1:6, size = 100, replace=T)
result_dice <- as.data.frame(table(dice))
result_dice

# From the Frequency table we calculate the probability of
# getting 1 from the dice and probability of the others too
```

```
Probability_Of_1_From_Dice <- result_dice[1,2]/100
Probability_Of_1_From_Dice
Probability_Of_others_From_Dice <- 1-Probability_Of_1_From_Dice
Probability_Of_others_From_Dice

# Finally we calculate the probability of a black ball by
# multiplying probability of each black ball with its
# corresponding probability of the dice and we add the
#2 probabilities for the final answer
Probability_Of_Black_Ball <- Probability_Of_1_From_Dice
                             * Probability_Black_Urn1
                             + Probability_Black_Urn2
                             * Probability_Of_others_From_Dice
Probability_Of_Black_Ball
```



```
>
> URN1 = c('V','V', 'V', 'V', 'V', 'V', 'N', 'N')
> URN2 = c('V','V', 'V', 'V', 'N', 'N', 'N')
> occurences1 <- as.data.frame(table(URN1))
> occurences1
  URN1 Freq
1    N    2
2    V    6
> Probability_Black_Urn1 = occurences1[1,2]/8
> Probability_Black_Urn1
[1] 0.25
> occurences2 <- as.data.frame(table(URN2))
> occurences2
  URN2 Freq
1    N    3
2    V    4
> Probability_Black_Urn2 = occurences1[1,2]/7
> Probability_Black_Urn2
[1] 0.2857143
> dice = sample(1:6, size = 100, replace=T)
> result_dice <- as.data.frame(table(dice))
> result_dice
  dice Freq
1    1   11
2    2   17
3    3   16
4    4   16
5    5   21
```

FIGURE 1.12: Jar problem part 1



```
> occurences2
  URN2 Freq
1    N    3
2    V    4
> Probability_Black_Urn2 = occurences1[1,2]/7
> Probability_Black_Urn2
[1] 0.2857143
> dice = sample(1:6, size = 100, replace=T)
> result_dice <- as.data.frame(table(dice))
> result_dice
  dice Freq
1    1   11
2    2   17
3    3   16
4    4   16
5    5   21
6    6   19
> Probability_Of_1_From_Dice <- result_dice[1,2]/100
> Probability_Of_1_From_Dice
[1] 0.11
> Probability_Of_others_From_Dice <- 1-Probability_Of_1_From_Dice
> Probability_Of_others_From_Dice
[1] 0.89
> Probability_Of_Black_Ball <- Probability_Of_1_From_Dice * Probability_Black_Urn1 + Probability_Black_Urn2 * Probability_Of_others_From_Dice
> Probability_Of_Black_Ball
[1] 0.2817857
> 
```

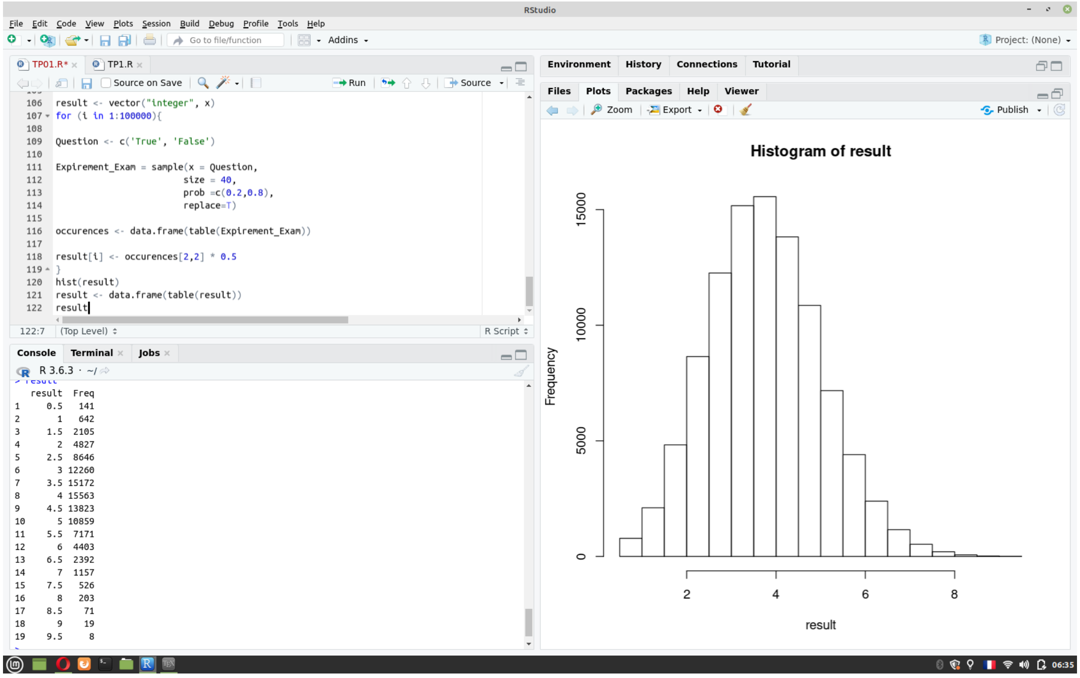
FIGURE 1.13: Jar problem part 2

1.6 Exercise 6

Un examen comprend 40 questions. Pour chacune de ces questions on propose cinq réponses dont une est juste. La réponse juste vaut 0.5, la réponse fausse vaut 0. Quelle est la probabilité qu'un étudiant ait la note de 12?

```
# create a vector to store the results of 100000 exam results
result <- vector("integer", x)

# start a loop of the desired test size
for (i in 1:100000){

# An answer to a question can be either true with
  #probability 1/5 or false with probability 4/5
Question <- c('True', 'False')

# we start the a single exam experiment with 40 questions
Expriment_Exam = sample(x = Question,
                        size = 40,
                        prob =c(0.2,0.8),
                        replace=T)

# we calculate true and false answers
occurences <- data.frame(table(Expriment_Exam))

# we multiply all true answers with the note of 0.5
#and store the total result of an exam
result[i] <- occurences[2,2] * 0.5
}
# display a histogram of all results
hist(result)
```

```
# check the total number of each score we observed
result <- data.frame(table(result))
result
```



```
result <- vector("integer", x)
for (i in 1:100000){

Question <- c('True', 'False')

Expirement_Exam = sample(x = Question,
                         size = 40,
                         prob =c(0.2,0.8),
                         replace=T)

occurences <- data.frame(table(Expirement_Exam))

result[i] <- occurences[2,2] * 0.5
}
hist(result)
result <- data.frame(table(result))
result
```

```
> result
   result  Freq
1     0.5   141
2       1   642
3     1.5  2105
4       2  4827
5     2.5  8646
6       3 12260
7     3.5 15172
8       4 15563
9     4.5 13823
10      5 10859
11    5.5  7171
12      6  4403
13    6.5  2392
14      7  1157
15    7.5   526
16      8   203
17    8.5    71
18      9    19
19    9.5     8
>
```

FIGURE 1.14: Exam problem + histogram