

# Documentație Proiect 1

Probabilități și Statistică

January, 2021

Contents

<b>1</b>	<b>Echipă</b>	<b>2</b>
<b>2</b>	<b>Introducere</b>	<b>2</b>
<b>3</b>	<b>Cerințe</b>	<b>2</b>
3.1	Cerința 1 . . . . .	2
3.2	Cerința 2 . . . . .	3
3.3	Cerința 3 . . . . .	4
3.4	Cerința 4 . . . . .	5
3.5	Cerința 5 . . . . .	10
3.6	Cerința 6 . . . . .	10
3.7	Cerința 7 . . . . .	11
3.8	Cerința 8 . . . . .	14
3.9	Cerința 9 . . . . .	16
3.10	Cerința 10 . . . . .	17
3.11	Cerința 11 . . . . .	18
3.12	Cerința 12 . . . . .	20
<b>4</b>	<b>Concluzie</b>	<b>22</b>

# 1 Echipă

**Membri:** Ștefan Radu, Comorașu Ana-Maria, Buhai Darius și Cîiașu Nicoleta.

**Grupă:** 234

## 2 Introducere

Construirea unui pachet R pentru lucru cu variabile aleatoare continue.

Folosind documentul suport și orice alte surse de documentare considerați potrivite construiți un pachet R care să permită lucru cu variabile aleatoare continue. Pentru a primi punctaj maxim, pachetul trebuie să implementeze cel puțin 8 din următoarele cerințe (oricare 8):

## 3 Cerințe

### 3.1 Cerința 1

#### Enunț

Fiind dată o funcție  $f$ , introdusă de utilizator, să se determine constanta de normalizare. Pentru cazul în care o asemenea constantă nu există, afișarea unui mesaj corespunzător către utilizator.

#### Rezolvare

Pentru calcularea constantei de normalizare, vom folosi regula:  $k \int_{-\infty}^{\infty} f(x) dx = 1$ .

Astfel, funcția *compute\_normalizing\_constant* primește 3 parametri, printre care ultimii doi pot avea valori implicite. Se va încerca să se calculeze integrala funcției pe domeniul său - sau pe intervalul  $(-\infty, \infty)$ .

Dacă aceasta a fost calculată cu succes, se va calcula constanta de normalizare drept  $i^{-1}$ , pentru  $i$  integrala rezultată. Dacă încercarea calculării constantei de normalizare eșuează, se va prinde eroarea și se va afișa mesajul corespunzător, iar valoarea returnată va fi 0.

#### Cod

```
1  # compute the normalizing constant of a pdf
2
3  compute_normalizing_constant <- function(f, lowerBound=-Inf, upperBound=Inf){
4    tryCatch(
5      expr = {
6        # compute the integral
7        integral <- integrate(f, lower = lowerBound, upper = upperBound)$value
8        # the constant is 1 / computed integral
9        normalizing_constant <- 1 / integral
10       return(normalizing_constant)
11     },
12     error = function(e){
13       message('Caught an error!')
14       print(e)
15       return(0)
16     },
17     warning = function(w){
18       message('Caught a warning!')
19       print(w)
20     },
21     finally = {
22       message('')
23     }
24   )
25 }
```

## Teste

```
1 f <- function(x){
2   return (exp((-x^2)/2))
3 }
4 compute_normalizing_constant(f, 0, 1)
5 # [1] 1.168737
6 compute_normalizing_constant(f)
7 # [1] 0.3989423
8
9 g <- function(x){
10   x^2
11 }
12 # Caught an error!
13 # <simpleError in integrate(f, lower = lowerBound, upper = upperBound): the integral is
14 # [1] 0
# probably divergent>
```

## 3.2 Cerința 2

### Enunț

Verificarea dacă o funcție introdusă de utilizator este densitate de probabilitate.

### Rezolvare

Pentru a verifica dacă funcția  $f$  este densitate de probabilitate, am testat următoarele 2 proprietăți:

1.  $f(x) \geq 0$  ( $f$  este nenegativă)
2.  $\int_{-\infty}^{\infty} f(x) dx = 1$

În primă parte, pentru a verifica dacă funcția dată este pozitivă, i-am calculat toate valorile din intervalul  $[-10000000, 10000000]$  și am verificat dacă sunt pozitive.

Pentru a evita erorile generate în urma integrării funcțiilor divergente, am creat o funcție `safe_integrate` ce returnează `FALSE` în cazul în care funcția dată nu poate fi integrată.

### Cod

```
1
2 # Integrate a function and return False if any errors are generated
3 safe_integrate <- function(f, d = c(-Inf, Inf)){
4   tryCatch(integrate(Vectorize(f), d[1], d[2]),
5     error = function(e){
6       print(e)
7       FALSE
8     })
9 }
10
11 # Check if a function is positive in a given interval
12 is_positive <- function(f, d = c(-10000000, 10000000), step=10000000){
13   # Set maximum lower and upper bounds
14   if(d[2] == Inf)
15     d[2] = 10000000;
16   if(d[1] == -Inf)
17     d[1] = -10000000;
18   # Calculate values in that intervals
19   vals <- seq(d[1], d[2], len=step)
20   # Check if all values are positive
21   all(f(vals)>=0)
22 }
23
24 check_pdf <- function(f, d = c(-Inf, Inf)){
25   # Check if the function is positive
26   if(!is_positive(f, d)){
```

```

27     print("Function is negative")
28     return(FALSE)
29   }
30   # Safe integrate, handle errors
31   i = safe_integrate(f, d)
32   # Cannot integrate
33   if(typeof(i)=='logical' && i==FALSE)
34     return(FALSE)
35   v = i $ value
36   round(v) == 1
37 }

```

## Teste

```

1  f1 <- function(x){
2    if (x > 0 && x < 2){
3      3/8 * (4*x-2*x^2)
4    }else{
5      0
6    }
7  }
8
9  check_pdf(f1)
10 # TRUE
11
12 f2 <- function(x){
13   x * (4*x-2*x^2)
14 }
15
16 check_pdf(f2)
17 # FALSE

```

## Probleme întâlnite

Verificarea funcțiilor pozitive nu este 100% precisă atunci când domeniul nu este definit, deoarece pe un interval infinit, valorile funcției sunt limitate la valori între -10000000 și 10000000.

## 3.3 Cerința 3

### Enunț

Crearea unui obiect de tip variabilă aleatoare continuă pornind de la o densitate de probabilitate introdusă de utilizator.

### Rezolvare

Se va crea o clasă cu mai multe proprietăți, printre care se numara pdf, constanta de normalizare, funcția normalizată.

### Cod

```

1  # Continuous RV class
2
3  Continuous_RV <- setClass(
4    "continuous_RV",
5    slots = list(
6      pdf = "function",
7      normalized = "function",
8      cdf = "function",
9      norm_constant = "numeric",
10     mean = "numeric"
11   )
12 )
13

```

```

14 init <- function(.Object, pdf)
15   # initialization
16   setMethod("init", "Continuous_RV", function(.Object, pdf){
17     tryCatch({
18       rand_var <- .Object
19       rand_var@pdf = pdf
20       rand_var@norm_constant = compute_normalizing_constant(pdf)
21       rand_var@normalized <- rand_var@pdf * rand_var@norm_constant
22       rand_var@cdf <- rand_var@pdf
23       rand_var@deviation <- rand_var@pdf
24       rand_var@mean <- E(rand_var@pdf)
25     })
26   })
27
28
29 show <- function(.Object)
30   setMethod("show", "Continuous_RV", function(.Object){
31     print(rand_var@norm_constant)
32     print(rand_var@pdf)
33     print(rand_var@mean)
34     print(rand_var@normalized)
35   })
36
37 # compute the mean
38 E <- function (probability_density_function, normalization_constant = 1){
39   # int (x * pdf(x)dx)
40   forIntegration <- function(x){
41     return (x * pdf(x) / normalization_constant)
42   }
43   # try to compute the integral
44   tryCatch(
45     expr = {
46       integral <- integrate(forIntegration, lower = -Inf, upper = Inf)$value
47       return(integral)
48     },
49     error = function(e){
50       message('Caught an error when calculating the mean!')
51       print(e)
52       return(0)
53     },
54     warning = function(w){
55       message('Caught a warning!')
56       print(w)
57     },
58     finally = {
59       message('')
60     })
61 }

```

## Teste

```

1 f <- function(x){
2   return (exp((-x^2)/2))
3 }
4 v <- Continuous_RV(pdf = f)

```

## 3.4 Cerința 4

### Enunț

Reprezentarea grafică a densității și a funcției de repartiție pentru diferite valori ale parametrilor repartiției. n cazul n care funcția de repartiție nu este dată într-o formă explicită (ex. repartiția normală) se acceptă reprezentarea grafică a unei aproximări a acesteia.

## Rezolvare

Pentru diferitele tipuri de distributii cunoscute am realizat graficul acestora in urma parsarii numelui si a parametrilor corespunzatori, dar si a gestionarii erorilor. Am implementat si metode suplimentare de realizare a graficului pentru functii aleatoare, pe baza definitiilor.

## Cod

```
1
2 # integreaza pdf ca sa obtin valoarea cdf
3 get_CDF_from_PDF <- function (f, x) {
4   tryCatch ({
5     integrate (f, 0, x)$value
6   },
7   error = function (e) {
8     print(e)
9   }
10 )
11 }
12
13 # verific daca functia de densitate este valida
14 # trebuie sa fie pozitiva
15 # probabilitatea totala trebuie sa fie 1 -> sigur se intampla ceva
16 check_density <- function(f, a, b) {
17   # verific prima conditie
18   for (x in seq(a, b, 0.1)) {
19     if (f(x) < 0) {
20       print(paste("pdf nu poate sa aiba valori negative",
21                 "dar am obtinut", f(x), "in", x, sep=" "))
22       return(FALSE)
23     }
24   }
25
26   # verific a doua conditie
27   total <- integral(Vectorize(f), max(-Inf, a), min(b, +Inf))
28   if (abs(total - 1) > 0.1) {
29     print(paste("probabilitatea cumulata totala trebuie sa fie egala cu 1",
30               "dar am obtinut", total, sep=" "))
31     return(FALSE)
32   }
33
34   return(TRUE)
35 }
36
37 # afisaza graficul densitatii
38 plot_density <- function(f, a, b, name="") {
39   # validez pdf
40   if (!check_density(f, a, b)) {
41     return()
42   }
43
44   # pdf a fost validata
45   # afisez graficul pdf
46
47   xs <- seq(a, b, 0.1)
48   ys <- c()
49   for (x in xs) {
50     ys = append(ys, f(x))
51   }
52   plot(xs, ys, type="l", main=noquote(paste(name, " PDF")), col="red", xlab="x", ylab="y")
53 }
54
55 # o folosesc pentru repartitiile standard carora le stim
56 # functia de repartitie
57 # afisaza graficul repartitiei
58 plot_repartition <- function(F, a, b, name) {
59   # calculez si afisez graficul CDF
60 }
```

```

61   xs <- seq(a, b, 0.01)
62   ys <- c()
63   for (x in xs) {
64     ys = append(ys, F(x))
65   }
66   plot(xs, ys, col="red", type="l", main=noquote(paste(name, " CDF")), xlab="x", ylab
67   ="y")
68 }
69
70 # o folosesc pentru repartitiile carora
71 # nu le stim functia de repartitie
72 # ne folosim de pdf
73 # afisaza graficul repartitiei
74 plot_generic_repartition <- function(f, a, b) {
75   # validez pdf
76   if (!check_density(f, a, b)) {
77     return()
78   }
79
80   xs <- seq(a, b, 0.01)
81   ys <- c()
82   for (x in xs) {
83     ys = append(ys, get_CDF_from_PDF(f, x))
84   }
85   plot(xs, ys, col="red", type="l", main="CDF", xlab="x", ylab="y")
86 }
87
88 # functia primeste ca parametrii numele repartitiei,
89 # un flag care marcheaza daca vrem graficul densitatii,
90 # sau al repartitiei, si, in plus, parametrii corespunzatori
91 # repartitiei selectate
92 # daca repartitia nu exista, sau daca parametrii nu corespund
93 # repartitiei alese, se va intoarce un mesaj corespunzator
94 parse_known_repartition <- function(name, CDF=FALSE, ...) {
95   params <- list(...)
96   if (name == "uniform") {
97     if (!is.null(params$a) && !is.null(params$b)) {
98       a <- params$a
99       b <- params$b
100       if (a >= b) {
101         return("parametrii incorecti")
102       }
103
104       f <- function(x) 1 / (b - a)
105       F <- function(x) (x - a) / (b - a)
106       if (CDF) {
107         plot_repartition(F, a, b, name)
108       }
109       else {
110         plot_density(f, a, b, name)
111       }
112     }
113     else {
114       return("parametrii necesari nu au fost pasati")
115     }
116   }
117   else if (name == "exp") {
118     if (!is.null(params$lambda)) {
119       lambda <- params$lambda
120       if (lambda <= 0) {
121         return("parametrii incorecti")
122       }
123
124       f <- function(x) (lambda * exp(1)^(-lambda * x))
125       F <- function(x) (1 - exp(1)^(-lambda * x))
126       if (CDF) {
127         plot_repartition(F, 0, 50, name)
128       }
129       else {

```



```

130     plot_density(f, 0, 50, name)
131   }
132 }
133 else {
134   return("parametrii necesari nu au fost pasati")
135 }
136 }
137 else if (name == "normal") {
138   if (!is.null(params$mu) && !is.null(params$sigma)) {
139     mu <- params$mu
140     sigma <- params$sigma
141     if (sigma <= 0) {
142       return("parametrii incorecti")
143     }
144
145     f <- function(x) ((1 / (sigma * sqrt(pi * 2))) * (exp(1)^((-x - mu)^2)/(2 *
sigma ^ 2))))
146     F <- function(x) (pnorm(x, mu, sigma))
147     if (CDF) {
148       plot_repartition(F, -25, 25, name)
149     }
150     else {
151       plot_density(f, -25, 25, name)
152     }
153   }
154   else {
155     return("parametrii necesari nu au fost pasati")
156   }
157 }
158 else if (name == "pareto") {
159   if (!is.null(params$m) && !is.null(params$alpha)) {
160     m <- params$m
161     alpha <- params$alpha
162     if (alpha <= 0 || m <= 0) {
163       return("parametrii incorecti")
164     }
165
166     f <- function(x) (alpha * m^alpha) / (x ^ (alpha + 1))
167     F <- function(x) (1 - (m ^ alpha) / (x ^ alpha))
168     if (CDF) {
169       plot_repartition(F, m, m + 50, name)
170     }
171     else {
172       plot_density(f, m, m + 50, name)
173     }
174   }
175 }
176 else if (name == "cauchy") {
177   if (!is.null(params$location) && !is.null(params$scale)) {
178     location <- params$location
179     scale <- params$scale
180     if (scale <= 0) {
181       return("parametrii incorecti")
182     }
183
184     f <- function(x) 1 / (pi * scale * (1 + ((x - location) / (scale))^2))
185     F <- function(x) (1 / pi) * atan((x - location) / scale) + 1 / 2
186
187     if (CDF) {
188       plot_repartition(F, -25, 25, name)
189     }
190     else {
191       plot_density(f, -25, 25, name)
192     }
193   }
194 }
195 else if (name == "logistic") {
196   if (!is.null(params$mu) && !is.null(params$s)) {
197     mu <- params$mu
198     s <- params$s

```

```

199     if (s <= 0) {
200       return("parametrii incorecti")
201     }
202
203     f <- function(x) (exp(1) ^ ((mu - x) / s) / (s * (1 + exp(1) ^ (mu - x) / s) ^
204 2))
205     F <- function(x) 1 / (1 + exp(1) ^ (-(x - mu) / s))
206
207     if (CDF) {
208       plot_repartition(F, -25, 25, name)
209     }
210     else {
211       plot_density(f, -25, 25, name)
212     }
213   }
214   else if (name == "weibull") {
215     if (!is.null(params$scale) && !is.null(params$shape)) {
216       scale <- params$scale
217       shape <- params$shape
218       if (scale <= 0 || shape <= 0) {
219         return("parametrii incorecti")
220       }
221
222       f <- function(x) {
223         if (x >= 0) (shape / scale) * (x / scale) ^ (shape - 1) * (exp(1) ^ (-(x /
scale) ^ shape))
224         else 0
225       }
226       F <- function(x) {
227         if (x >= 0) 1 - (exp(1) ^ (-(x / scale) ^ shape))
228         else 0
229       }
230
231       if (CDF) {
232         plot_repartition(F, 0, 50, name)
233       }
234       else {
235         plot_density(f, 0, 50, name)
236       }
237     }
238   }
239   else {
240     print("repartitie necunoscuta")
241   }
242 }

```

## Teste

```

1  parse_known_repartition("uniform", FALSE, a=0, b=10)
2  parse_known_repartition("uniform", TRUE, a=0, b=10)
3  parse_known_repartition("exp", FALSE, lambda=2)
4  parse_known_repartition("exp", TRUE, lambda=2)
5  parse_known_repartition("normal", FALSE, mu=0, sigma=1)
6  parse_known_repartition("normal", TRUE, mu=0, sigma=1)
7  parse_known_repartition("pareto", FALSE, m=3, alpha=1)
8  parse_known_repartition("pareto", TRUE, m=3, alpha=1)
9  parse_known_repartition("cauchy", FALSE, location=0, scale=1)
10 parse_known_repartition("cauchy", TRUE, location=0, scale=1)
11 parse_known_repartition("logistic", FALSE, mu=0, s=1)
12 parse_known_repartition("logistic", TRUE, mu=0, s=1)
13 parse_known_repartition("weibull", FALSE, scale=1, shape=2)
14 parse_known_repartition("weibull", TRUE, scale=1, shape=2)
15 plot_generic_repartition(function(x) x / 2, 0, 2)

```

### 3.5 Cerința 5

#### Enunț

Calculul mediei, dispersiei și a momentelor inițiale și centrate până la ordinul 4 (dacă există). Atunci când unul dintre momente nu există, se va afișa un mesaj corespunzător către utilizator.

#### Rezolvare

În rezolvarea acestui exercițiu am folosit următoarele formule, preluate de pe [Wikipedia](#):

Momentul Central:  $\mu_n = \int_{-\infty}^{\infty} (x - \mu)^n * f(x) dx$

Momentul Inițial:  $\mu_n = \int_{-\infty}^{\infty} x^n * f(x) dx$

Iar pentru a determina dacă un moment dat nu există, am folosit funcția `safe_integrate` ce returnează `FALSE` în cazul în care pdf-ul dat nu poate fi integrat, afișând pe ecran un mesaj corespunzător.

#### Cod

```
1
2 # Calculating central moment of order o
3 central_moment <- function(f, o){
4   m = median(f)
5   f_new <- function(x){
6     ((x - m) ^ o) * f(x)
7   }
8   res = safe_integrate(f)
9   if(typeof(res)=="logical" && res==FALSE){
10     print("Momentul nu exist !")
11     return(0)
12   }
13   res $ value
14 }
15
16 # Calculating initial moment of order o
17 initial_moment <- function(f, o){
18   f_new <- function(x){
19     x ^ o * f(x)
20   }
21   res = safe_integrate(f)
22   if(typeof(res)=="logical" && res==FALSE){
23     print("Momentul nu exist !")
24     return(0)
25   }
26   res $ value
27 }
```

#### Teste

```
1 f = function (x) {
2   if(x<0 || x>1)
3     return(0)
4   x ^ 3
5 }
6
7 initial_moment(f, 2)
8 central_moment(f, 2)
```

### 3.6 Cerința 6

#### Enunț

Calculul mediei și dispersiei unei variabile aleatoare  $g(X)$ , unde  $X$  are o repartiție continuă cunoscută iar  $g$  este o funcție continuă precizată de utilizator.

## Rezolvare

Am creat o functie care primeste ca parametri functia data  $g$ , respectiv pdf-ul si domeniul de valori al lui  $X$ . Dupa aceea, am aplicat formula de calcul a mediei functiilor de  $X$ . Dispersia am calculat-o cu formula

$$\text{Var}(X) = E(X^2) - E(X)^2.$$

Cum pe  $E(X)$  l-am calculat deja, mai ramane sa il calculez pe  $E(X^2)$ , care poate fi interpretat ca un  $E(h(X))$  unde  $h(x) = x^2$ , deci pot reaplica formula mediei functiilor de  $X$ .

## Cod

```
1 # am nevoie de functia g, pdf-ul lui X si domeniul de definitie
2 median_and_dispersion <- function(g, fx, domeniu_valori) {
3   # Y = g(X) poate fi vazut ca o v.a.c. noua cu pdf-ul g(f(x)), ambele functii fiind
4   # continue le compun
5
6   # folosesc formula pt media functiilor de x
7   # folosesc integral(din pachetul pracma) deoarece integer imi da eroare cand incerc
8   # sa integrez la +-inf
9   e_y <- integral(Vectorize(function(x){g(x) * fx(x)}), domeniu_valori[1], domeniu_
10  valori[2])
11
12   # pt dispersie, mai folosesc o data formula mediei functiilor de x
13   # pentru a afla X^2
14   e_y2 <- integral(Vectorize(function(x){x^2 * g(x) * fx(x)}), domeniu_valori[1],
15  domeniu_valori[2])
16   dispersie <- e_y2 - e_y^2
17
18   print(paste("Media: ", e_y))
19   print(paste("Dispersia:", dispersie))
20 }
```

## Teste

```
1 # exemplu
2 f1 <- function(x) (x^2)
3 f2 <- function(x) (1 * exp(1)^(-1 * x))
4
5 median_and_dispersion(f1,f2, c(0,Inf)) # 2, 20
```

## 3.7 Cerința 7

### Enunț

Crearea unei funcții  $P$  care permite calculul diferitelor tipuri de probabilități asociate unei variabile aleatoare continue(similar funcției  $P$  din pachetul discreteRV)

## Cod

```
1 #####
2 # Cerinta: Crearea unei func ii P care permite calculul diferitelor tipuri de
3 # probabilit i asociate unei variabile aleatoare continue(similar func iei P din
4 # pachetul discreteRV)
5 #
6 # Header functie: myP(f, p)
7 # - unde f este o functie densitate de probabilitate (pdf)
8 # - iar p este un string ce reprezinta probabilitatea (conditionata sau
9 # independenta).
10 # Obligatoriu, var se va afla in stanga operatorului
11 #####
12 myP <- function(f, p) {
```

```

13   operatii_posibile=c("<=", ">=", "=", "<", ">")
14
15   ##### Functii ajutatoare #####
16   # transforma string-ul dat in ceva ce pot utiliza
17   parseaza_expresie <- function(expresie) {
18
19       # scot whitespace
20       expresie <- gsub(" ", "", expresie)
21
22       # iau fiecare operatie posibila pe rand si incerc sa dau split dupa ea, daca pot,
23       # mi-am gasit operatorul si returnez parametri
24       for(op in operatii_posibile) {
25
26           # am dat split corect => in stanga am variabila, in dreapta am bound-ul
27           split <- unlist(strsplit(expresie, op, fixed = TRUE))
28           splitSize <- length(split)
29
30           if (splitSize == 2) {
31               # returnez (v.a.c, operatie, bound)
32               return (c(split[1], op, split[2]))
33           }
34       }
35       #daca am ajuns aici => nu am operator, deci eroare
36       return(c(-1))
37
38   }
39
40   # calculez cdf, adica integrala -inf, bound, adica P(X <= bound)
41   cdf <- function(bound) { return (integrate(f, -Inf, bound) $ value)}
42
43   # transform din string in double
44   compute_bound <-function(bound) {
45       rez <- switch(bound,
46                     "-Inf" = -Inf,
47                     "+Inf" = +Inf,
48                     as.double(bound))
49       return (rez)
50   }
51
52   ## Calculeaza probabilitatea ##
53   evalueaza <- function(operator, bound) {
54
55       # parsez bound-ul, transform in double sau in +-inf
56       bound = compute_bound(bound)
57       integrala <- cdf(bound)
58
59       #pentru >, >= din toata aria(1) scad cdf-ul si obtin restul
60       ans <- switch(
61           operator,
62           "=" = 0,
63           "<=" = integrala,
64           "<" = integrala,
65           ">=" = 1 - integrala,
66           ">" = 1 - integrala)
67
68       return(ans)
69   }
70
71   # daca am o singura expresie e prob independenta
72   prob_independenta <- function(expresie) {
73
74       # scot parametri din expresie
75       parametri <- parseaza_expresie(expresie)
76
77       if(length(parametri) != 3)
78           return("Eroare la parsarea probabilitatii")
79
80       # aici presupun ca expresiile mele sunt mereu de forma x operator bound

```

```

82     # ar trebui o verificare, poate, a ordinii
83
84     operator <- parametri[2]
85     bound <- parametri[3]
86
87     print(evalueaza(operator, bound))
88
89 }
90
91 #daca am 2 expresii e prob conditionata
92 prob_conditionata <- function(expresie1, expresie2) {
93
94     # scot parametri in variabile ca sa fie readable ce fac mai jos
95     parametri1 <- parseaza_expresie(expresie1)
96     parametri2 <- parseaza_expresie(expresie2)
97     op1 <- parametri1[2]
98     op2 <- parametri2[2]
99     bound1 <- parametri1[3]
100    bound2 <- parametri2[3]
101
102    # am formula  $P(a \text{ depinde de } b) = P(a \text{ intersectat } b)/P(b)$ 
103    # calculez cdf pentru fiecare functie si iau pe cazuri
104    ans1 <- evalueaza(op1, bound1)
105    ans2 <- evalueaza(op2, bound2)
106
107    # daca vreuna e 0, intersectia va da 0
108    if(ans1 == 0)
109        return(0);
110    if(ans2 == 0)
111        return ("Cannot divide by zero")
112
113    ## cazuri
114
115    ## caz in care conditionarea face probabilitatea sa fie imposibila
116    #  $p(x < 3 \mid x > 5) = 0$ 
117    if (op1 %in% c("<=", "<") && op2 %in% c(">=", ">") && bound1 >=bound2)
118        return (0);
119
120    #  $p(x > 5 \mid x < 3) = 0$ 
121    if (op1 %in% c(">=", ">") && op2 %in% c("<=", "<") && bound1 >=bound2)
122        return (0);
123
124    ## caz in care am acelasi fel de operator, facand intersectia defapt doar aleg
125    intervalul cel mai restrans
126    #  $p(x > 3 \mid x > 7) \Rightarrow$  iau  $(-\infty, 3)$ 
127    if(op1 %in% c(">=", ">") && op2 %in% c(">=", ">"))
128        if(bound1 > bound2)
129            return (ans1/ans2)
130        else return (1);
131    #  $p(x < 3 \mid x < 7) \Rightarrow$  iau  $(-\infty, 3)$ 
132    if(op1 %in% c("<=", "<") && op2 %in% c("<=", "<"))
133        if(bound1 < bound2)
134            return (ans1/ans2)
135        else return (1)
136
137    ## daca nu e niciunul de mai sus, e intersectie de forma  $x > 5 \mid x < 7$  si fac
138    diferenta
139    ## din cdf mai mare scad cdf mai mic si mi da bucata de dintre => intersectia
140    return ((cdf(compute_bound(bound2))-cdf(compute_bound(bound1)))/ans2)
141
142 }
143
144 ##### Body functie principala #####
145
146 # parsez parametri
147 parti = unlist(strsplit(p, "|", fixed = TRUE))
148 len = paste(length(parti))
149 switch(len,
150     "0" = return("Eroare"),
151     "1" = return(prob_independenta(p)),

```

```

150         "2" = return(prob_conditionata(parti[1],parti[2])),
151     )
152     return ("eroare");
153
154 }

```

## Teste

```

1  g <- function (x) {
2      fun <- 0.1*(3*(x^2) + 1)
3      fun[x<0] = 0
4      fun[x>2]=0
5      return ( fun )
6  }
7
8  h <- function(x)(dunif(x))
9
10 myP(g, "x>1|x<1.5") # 0.5897078
11 myP(h, "x>0.6") # 0.4

```

## 3.8 Cerința 8

### Enunț

Afișarea unei “fișe de sinteză” care să conțină informații de bază despre respectiva repartiție (cu precizarea sursei informației!). Relevant aici ar fi să precizați pentru ce e folosită în mod uzual acea repartiție, semnificația parametrilor, media, dispersia etc.

### Rezolvare

Am afișat detalii despre diverse distribuții din curs, dar și găsite în urma căutărilor proprii, precum ”Cauchy”, sau ”Weibull”.

### Dificultati și Abordare

Dificultatea cerinței a constat în gruparea datelor împreună. După parcurgerea mai multor pachete auxiliare am ales să folosesc obiectul *Dictionary* din *mlr3misc*. Modul de folosire este puțin complicat, întrucât se bazează pe utilizarea unui *R6Class*, dar în final, rezultatul este bine structurat și ușor de folosit pentru distribuirea și afișarea datelor.

### Cod

```

1
2 library(mlr3misc)
3 library(R6)
4
5 data = Dictionary$new()
6
7 item = R6Class("Item")
8
9 ##### UNIFORM #####
10
11 data_aux = Dictionary$new()
12 data_aux$add("Sursa", R6Class(public = list(print = "Curs + Wikipedia")))
13 data_aux$add("Notatie", R6Class(public = list(print = "U(a, b)")))
14 data_aux$add("Parametri", R6Class(public = list(print = "-Inf < a < b < Inf")))
15 data_aux$add("Domeniu", R6Class(public = list(print = "x in [a, b]")))
16 data_aux$add("PDF", R6Class(public = list(print = "1 / (b - a)")))
17 data_aux$add("CDF", R6Class(public = list(print = "(x - a) / (b - a)")))
18 data_aux$add("Mediana", R6Class(public = list(print = "(1 / 2) * (a + b)")))
19 data_aux$add("Media", R6Class(public = list(print = "(1 / 2) * (a + b)")))
20 data_aux$add("Utilizari", R6Class(public = list(print = "In economie, pentru managerierea inventarului, cand un produs complet nou este analizat, distributia uniforma e cea mai utila.")))

```

```

21 data$add("uniform", data_aux)
22
23 ##### EXPONENTIAL #####
24
25 data_aux = Dictionary$new()
26 data_aux$add("Sursa", R6Class(public = list(print = "Curs + Wikipedia")))
27 data_aux$add("Parametri", R6Class(public = list(print = "lambda > 0")))
28 data_aux$add("Domeniu", R6Class(public = list(print = "x in [0, +Inf]")))
29 data_aux$add("PDF", R6Class(public = list(print = "lambda * exp(1)^(-lambda * x)")))
30 data_aux$add("CDF", R6Class(public = list(print = "1 - exp(1)^(-lambda * x)")))
31 data_aux$add("Media", R6Class(public = list(print = "1 / lambda")))
32 data_aux$add("Mediana", R6Class(public = list(print = "ln2 / lambda")))
33 data_aux$add("Utilizari", R6Class(public = list(print = "Dupa observarea unui set de n
    puncte dintr-o distributie exponentiala necunoscuta, se pot face predictii destul de
    precise despre viitor, pe baza datelor sursa.")))
34 data$add("exponential", data_aux)
35
36 ##### NORMAL #####
37
38 data_aux = Dictionary$new()
39 data_aux$add("Sursa", R6Class(public = list(print = "Curs + Wikipedia")))
40 data_aux$add("Notatie", R6Class(public = list(print = "N(mu, sigma^2)")))
41 data_aux$add("Parametri", R6Class(public = list(print = "mu in R, sigma ^ 2 >= 0")))
42 data_aux$add("Domeniu", R6Class(public = list(print = "x in [-Inf, +Inf]")))
43 data_aux$add("PDF", R6Class(public = list(print = "(1 / (sigma * sqrt(pi * 2)))*(exp(1)
    ^((-x - mu)^2/(2 * sigma ^ 2)))")))
44 data_aux$add("CDF", R6Class(public = list(print = "(1 / 2) * (1 + erf((x - mu) / (sigma *
    sqrt(2))))")))
45 data_aux$add("Media", R6Class(public = list(print = "mu")))
46 data_aux$add("Mediana", R6Class(public = list(print = "mu")))
47 data_aux$add("Utilizari", R6Class(public = list(print = "Poate fi observata in ziua de zi
    cu zi. De exemplu distributia notelor unor studenti la examenul de Probabilitati si
    Statistica.")))
48 data$add("normal", data_aux)
49
50 ##### PARETO #####
51
52 data_aux = Dictionary$new()
53 data_aux$add("Sursa", R6Class(public = list(print = "Curs + Wikipedia")))
54 data_aux$add("Parametri", R6Class(public = list(print = "m > 0, alpha > 0")))
55 data_aux$add("Domeniu", R6Class(public = list(print = "x in [m, +Inf]")))
56 data_aux$add("PDF", R6Class(public = list(print = "(alpha * m^alpha) / (x ^ (alpha + 1))"
    )))
57 data_aux$add("CDF", R6Class(public = list(print = "(1 - (m ^ alpha) / (x ^ alpha))")))
58 data_aux$add("Media", R6Class(public = list(print = "Inf, alpha <= 1\n (alpha * m) / (
    alpha - 1), alpha > 1")))
59 data_aux$add("Mediana", R6Class(public = list(print = "m * sqrt_alpha(2)")))
60 data_aux$add("Utilizari", R6Class(public = list(print = "A fost folosita de catre
    creatorul ei, Vilfredo Pareto, pentru analiza distributiei banilor intre indivizi.
    Aceasta arata destul de precis cum cea mai mare cantitate de bani este controlata de
    un procent foarte mic din populatie.")))
61 data$add("pareto", data_aux)
62
63 ##### CAUCHY #####
64
65 data_aux = Dictionary$new()
66 data_aux$add("Sursa", R6Class(public = list(print = "Curs + Wikipedia")))
67 data_aux$add("Parametrii", R6Class(public = list(print = "x0, y > 0")))
68 data_aux$add("Domeniu", R6Class(public = list(print = "x in [-Inf, Inf]")))
69 data_aux$add("PDF", R6Class(public = list(print = "1 / (pi * y * (1 + (x - x0) / y)^2)")))
70 data_aux$add("CDF", R6Class(public = list(print = "(1 / pi) * arctan((x - x0) / y) + (1/
    2)")))
71 data_aux$add("Mediana", R6Class(public = list(print = "x0")))
72 data_aux$add("Media", R6Class(public = list(print = "undefined")))
73 data_aux$add("Utilizari", R6Class(public = list(print = "Distributia Cauchy este folosita
    des in analiza distributiei observatiilor facute de catre oameni de stiinta despre
    despre obiecte care se invartesc.")))
74 data$add("cauchy", data_aux)
75

```



```

76 ##### WEIBULL #####
77
78 data_aux = Dictionary$new()
79 data_aux$add("Sursa", R6Class(public = list(print = "Wikipedia")))
80 data_aux$add("Parametri", R6Class(public = list(print = "lambda in (0,+inf) da scara, k
    in (0,+inf) da forma")))
81 data_aux$add("Domeniu", R6Class(public = list(print = "x in [0, +inf]")))
82 data_aux$add("PDF", R6Class(public = list(print = "k/lambda * (x/lambda) ^ (k-1) * e^(-(x
    /lambda)^k), x>=0, altfel 0")))
83 data_aux$add("CDF", R6Class(public = list(print = "1-e^((-x/lambda)^k), x>=0, altfel 0")))
84 data_aux$add("Mediana", R6Class(public = list(print = "lambda * (ln 2)^(1/k)")))
85 data_aux$add("Media", R6Class(public = list(print = "(lambda * gamma(1+1/k)")))
86 data_aux$add("Utilizari", R6Class(public = list(print = "In meteorologie, folosita la
    prezicerea distributiei vitezei vantului, in ingineria electrica.")))
87 data$add("weibull", data_aux)
88
89
90 ##### LOGISTIC #####
91
92 data_aux = Dictionary$new()
93 data_aux$add("Sursa", R6Class(public = list(print = "Curs + Wikipedia")))
94 data_aux$add("Parametrii", R6Class(public = list(print = "u, s>0")))
95 data_aux$add("Domeniu", R6Class(public = list(print = "x in [-Inf, Inf]")))
96 data_aux$add("PDF", R6Class(public = list(print = "(e^(-(x-u)/s))/(s*(1+e^(-(x-u)/s)))^2
    ")")))
97 data_aux$add("CDF", R6Class(public = list(print = "1/(1+e^(-(x-u)/s))")))
98 data_aux$add("Mediana", R6Class(public = list(print = "u")))
99 data_aux$add("Media", R6Class(public = list(print = "u")))
100 data_aux$add("Utilizari", R6Class(public = list(print = "Este adesea folosita in regresia
    logistica si in Retelele Neuronale de tip 'feedforward'.")))
101 data$add("logistic", data_aux)
102
103
104 ##### FUNCTIE DE AFISARE #####
105
106 output_details <- function(distribution_name) {
107   out_data = data$get(distribution_name)
108   for (key in out_data$keys()) {
109     print(noquote(paste(key, ": ", out_data$get(key)$print, sep="")))
110   }
111 }

```

## Teste

```

1
2 output_details("uniform")
3 output_details("exponential")
4 output_details("normal")
5 output_details("pareto")
6 output_details("cauchy")
7 output_details("logistic")
8 output_details("weibull")

```

## 3.9 Cerința 9

### Enunț

Generarea a  $n$  valori (unde  $n$  este precizat de utilizator!) dintr-o repartiție de variabile aleatoare continue.

### Rezolvare

Sursa de informare: [Generation of Random Variables](#)

### Cod

```

1 #function to integrate the probability density function
2 integrate_pdf <- function (f, X) {
3   tryCatch (
4     expr = {
5       integrate (f, 0, X)$value
6     },
7     error = function (e) {
8       print(e)
9     }
10  )
11 }
12
13 # compute the inverse function
14 inverse_function <- function (f, u, lowerBound = -10000, upperBound = 10000) {
15   tryCatch (
16     expr = {
17       uniroot((function (x) f(x) - u), lower = lowerBound, upper = upperBound, extendInt
18         = 'yes')
19     },
20     error = function (e) {
21       print(e)
22     }
23   )
24 }
25
26 random_variable_generator <- function(f, n, lowerBound, upperBound) {
27   # generates random deviates
28   values <- runif(n, min = 0, max = 1)
29   # combines its arguments into a list
30   r <- c()
31   for (v in values) {
32     r <- append(r, inverse_function(function (x) (integrate_pdf(f, x)), v, lowerBound,
33       upperBound)[1]$root)
34   }
35   r
36 }

```

## Teste

```

1 > f <- function(x) (3*x^2 + 1) * (0 < x & x <= 2)
2 > random_variable_generator (f, 50, 0, 2)
3 [1] 0.334985580 0.284023545 0.504948054 0.580059082 0.387038202 0.603904189 0.336524849
4 [8] 0.143201842 0.526265054 0.052287091 0.254292691 0.315695332 0.573806730 0.015253400
5 [15] 0.069285160 0.188473281 0.212269327 0.296630054 0.304598423 0.295342073 0.398116253
6 [22] 0.025906030 0.383253613 0.303505851 0.097724837 0.669221135 0.329533718 0.440763350
7 [29] 0.137255726 0.654135788 0.622121891 0.226515546 0.458965068 0.221886751 0.005993441
8 [36] 0.573435632 0.494637763 0.518431927 0.644874958 0.268270492 0.524442856 0.565096638
9 [43] 0.436779008 0.245007153 0.504013239 0.442178901 0.277991793 0.402183559 0.426914696
10 [50] 0.570814779

```

## 3.10 Cerința 10

### Enunț

Calculul covarianței și coeficientului de corelație pentru două variabile aleatoare continue.

### Rezolvare

În rezolvarea problemei am folosit următoarele formule de calcul ale covarianței și a coeficientului de corelație preluate din Cursurile 8 și 6.

De asemenea, pentru a determina funcțiile pdf ale variabilelor  $x$  și  $y$  din densitatea comună, am integrat pdf.comun în funcție de  $x$  și  $y$  pe domeniile aferente.

$$\mu_x = E(x) = \int_a^b x * f(x) dx$$

$$\text{cov}(x, y) = \left( \int_c^d \int_a^b x * y * f(x, y) dx dy \right) - \mu_x * \mu_y$$

$$\text{cor}(x, y) = \frac{\text{cov}(x, y)}{\sigma_x * \sigma_y}$$

## Cod

```

1
2 # Calculates medium
3 medium <- function(f, d = c(-Inf, Inf)){
4   integrate(function(x){ x * f(x)}, d[1], d[2]) $ value
5 }
6
7 # Extracts x out of common density of x and y
8 extract_x_marginal <- function(f, dx){
9   Vectorize(function(y){
10     integrate(function(x){f(x, y)}, dx[1], dx[2]) $ value
11   })
12 }
13
14 # Extracts y out of common density of x and y
15 extract_y_marginal <- function(f, dy){
16   Vectorize(function(x){
17     integrate(function(y){f(x, y)}, dy[1], dy[2]) $ value
18   })
19 }
20
21 # Calculates Covariance and Correlation coefficient
22 covariance_and_correlation <- function(pdf, dx, dy){
23
24   fx = extract_x_marginal(pdf, dx)
25   fy = extract_y_marginal(pdf, dy)
26
27   mx = medium(fx, dx)
28   my = medium(fy, dy)
29
30   cov = double_integrate(function(x,y){x*y*pdf(x,y)}, dx, dy) - (mx*my)
31
32   cor = cov / (mx*my)
33
34   list(cov = cov, cor = cor)
35 }

```

## Teste

```

1 f <- function (x, y) {
2   return (3/2 * (x^2+y^2))
3 }
4 covariance_and_correlation(f, c(0,1), c(0,3))
5
6 # $cov
7 # [1] -78.04688
8 # $cor
9 # [1] -0.8222222

```

## Probleme întâlnite

În rezolvarea problemei am găsit dificil determinarea pdf\_x și pdf\_y din pdf\_comun, lucru pe care l-am rezolvat prin funcțiile extract\_x\_marginal și extract\_y\_marginal.

## 3.11 Cerința 11

### Enunț

Pornind de la densitatea comună a două variabile aleatoare continue, construirea densităților marginale și a densităților condiționate.

## Dificultati

Mi s-a parut dificil la inceput sa imi dau seama cum sa realizez integrarea partiala cu un singur parametru setat.

## Cod

```
1  # conditia 1 pentru o functie de densitate corecta
2  # f trebuie sa aiba valori pozitive pentru
3  # toate valorile din domeniu
4  # cum in domeniu sunt o infinitate de puncte
5  # am ales sa verific 'pe sarite' o parte dintre ele
6  check_positive_2d <- function(f, ab, cd, step) {
7    for (x in seq(ab[1], ab[2], step)) {
8      for (y in seq(cd[1], cd[2], step)) {
9        if (f(x, y) < 0) return(FALSE)
10     }
11   }
12
13   return(TRUE)
14 }
15
16 # conditia 2 pentru o functie de densitate corecta
17 # f trebuie sa aiba probabilitatea totala egala cu 1
18 # pentru a calcula integrala dubla am folosit functia
19 # 'integral2' din pachetul 'pracma'
20 check_integral_is_1 <- function(f, ab, cd) {
21   integral <- integral2(f, ab[1], ab[2], cd[1], cd[2])$Q
22   return(abs(integral - 1) < 0.0000000001)
23 }
24
25 # intoarce o noua functie g(y) definita
26 # ca f(x, y), unde x este o valoare fixata
27 f_set_x <- function(x, f) function(y) f(x, y)
28
29 # intoarce o noua functie g(x) definita
30 # ca f(x, y), unde y este o valoare fixata
31 f_set_y <- function(y, f) function(x) f(x, y)
32
33 # calculeaza densitatea marginala in X
34 # il setam pe x si integram peste y
35 x_marginal_pdf <- function(f, x, c, d) {
36   return (integrate(f_set_x(x, f), c, d)$value)
37 }
38
39 # calculeaza densitatea marginala in Y
40 # il setam pe y si integram peste x
41 y_marginal_pdf <- function(f, y, a, b) {
42   return (integrate(f_set_y(y, f), a, b)$value)
43 }
44
45 # calculeaza densitatea in X conditionata de Y = y
46 h_x_conditioned_by_y <- function(y, x, f, c, d) {
47   f(x, y) / x_marginal_pdf(f, x, c, d)
48 }
49
50 # calculeaza densitatea in Y conditionata de X = x
51 h_y_conditioned_by_x <- function(x, y, f, a, b) {
52   f(x, y) / y_marginal_pdf(f, y, a, b)
53 }
54
55 marginal_conditional_densities <- function(f, x, y, ab, cd) {
56
57   tryCatch({
58     if (!check_positive_2d(f, ab, cd, 0.01)) {
59       print("functia nu este pozitiva pentru valorile din domeniu")
60       return ()
61     }
62   })
```

```

63     if (!check_integral_is_1(f, ab, cd)) {
64         print("probabilitatea totala este diferita de 1")
65         return ()
66     }
67
68     print(paste("testez cu x=", x, " y=", y, " pe [", ab[1], ",",
69               ab[2], "]x[", cd[1], ",", cd[2], "]", sep=""))
70
71     # densitatea marginala pentru X cu x fixat
72     print(paste("pdf marginal cu x fixat:", x_marginal_pdf(f, x, cd[1], cd[2])))
73
74     # densitatea marginala pentru Y cu y fixat
75     print(paste("pdf marginal cu y fixat:", y_marginal_pdf(f, y, ab[1], ab[2])))
76
77     # densitatea conditionata in x cand Y = y
78     print(paste("pdf conditionat de y=", y, ": ", h_x_conditioned_by_y(y, x, f, cd
79 [1], cd[2]), sep=""))
80
81     # densitatea conditionata in y cand X = x
82     print(paste("pdf conditionat de x=", x, ": ", h_y_conditioned_by_x(x, y, f, ab
83 [1], ab[2]), sep=""))
84     }, error = function(e) {
85         print(paste("A aparut o eroare", e, sep = " "))
86     })
87 }

```

## Teste

```

1  test_1 <- function() {
2      f <- function(x, y) (8 / 3) * x ^ 3 * y
3      marginal_conditional_densities(f, 0.5, 1.5, c(0, 1), c(1, 2))
4  }
5
6  test_2 <- function() {
7      f <- function(x, y) (3 / 2) * (x ^ 2 + y ^ 2)
8      marginal_conditional_densities(f, 0, 0.5, c(0, 1), c(0, 1))
9  }
10
11  test_1()
12  test_2()

```

## 3.12 Cerința 12

### Enunț

Construirea sumei și diferenței a două variabile aleatoare continue independente (folosiți formula de convoluție)

### Documentare

Întâi a trebuit să mă documentez despre formula de convoluție, așa că m-am orientat spre următoarele surse:

1. Pagina [Convolutions](#) de pe StatLect, unde am găsit formula pentru convoluția a două pdf-uri ale unor variabile aleatoare continue independente.
2. O parte dintr-un curs de pe MIT OpenCourseWare despre [suma variabilelor aleatoare continue independente](#).
3. Acest [video de pe YouTube](#) care vorbește despre formula pentru diferență.

Pentru rezolvarea cerinței, am folosit formulele următoare care nu sunt prezente în curs:

$$f_Z(z) = \int_{-\infty}^{\infty} f_X(z-y)f_Y(y) dy \text{ pentru } Z = X + Y$$

$$f_Z(z) = \int_{-\infty}^{\infty} f_X(y-z)f_Y(y) dy \text{ pentru } Z = X - Y.$$

## Cod

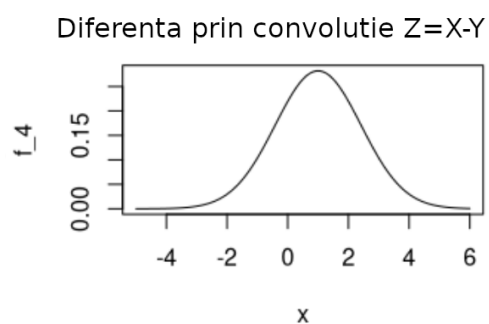
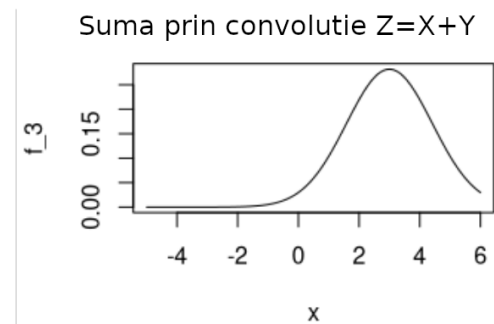
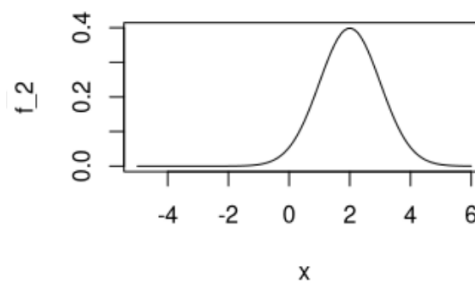
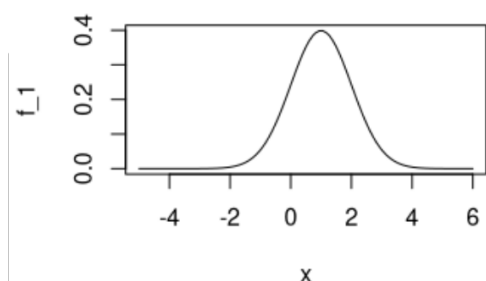
```
1  # suma a doua variabile aleatoare continue independente folosind formula de
   convolutie
2
3  convolution_sum <- function(fx,fy) {
4    res <-function(z) {
5      integrate(function(y) {
6        fx(z-y) * fy(y)
7      },-Inf,Inf) $ value
8    }
9  }
10
11 # diferenta a doua variabile aleatoare continue independente folosind formula de
   convolutie
12 convolution_diff <- function(fx,fy) {
13   res <-function(z) {
14     integrate(function(y) {
15       g(y-z)*f(y)
16     },-Inf,Inf) $ value
17   }
18 }
```

## Teste

```
1  # doua distributii normale, suma si diferenta facuta cu formula de convolutie
2  f_1 <- function(x)(dnorm(x,mean=1))
3  f_2 <- function(x) (dnorm(x,mean=2))
4  f_3 <- Vectorize(convolution_sum(f_1, f_2))
5  f_4 <- Vectorize(convolution_diff(f_1,f_2))
6
7  plot(f_1,from=-10,to=10,type="l")
8  plot(f_2,from=-10,to=10,type="l")
9  plot(f_3(t),from=-10,to=10,type="l")
10 plot(f_4(t),from=-10,to=10,type="l")
```

## Output

Pentru două v.a.c cu distributii normale:



## Probleme întâlnite

A fost nevoie sa vectorizez pdf-urile rezultate prin convolutie ca sa pot sa le plotez.

## 4 Concluzie

Prin urmare putem spune că prin rezolvarea acestui proiect, am învățat să aplicăm diferite formule de variabile aleatoare continue în limbajul R, cât și să construim un pachet complet R, cu documentație și manual.