Documentație Proiect 1

Probabilități și Statistică

January, 2021

Contents

1 Echipă

Membri: Ștefan Radu, Comorașu Ana-Maria, Buhai Darius și Ciaușu 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

```
# compute the normalizing constant of a pdf

compute_normalizing_constant <- function(f, lowerBound=-Inf, upperBound=Inf){
  tryCatch(
    expr = {
      # compute the integral
      integral <- integrate(f, lower = lowerBound, upper = upperBound)$value
      # the constant is 1 / computed integral
      normalizing_constant <- 1 / integral
      return(normalizing_constant)
    },
    error = function(e){
      message('Caught an error!')
      print(e)
      return(0)
    },
    warning = function(w){
      message('Caught a warning!')
      print(w)
    },
    finally = {
      message('')
    }
  )
}
```

Teste

```
f <- function(x){
  return (exp((-x^2)/2))
}
compute_normalizing_constant(f, 0, 1)
# [1] 1.168737
compute_normalizing_constant(f)
# [1] 0.3989423

g <- function(x){
  x^2
}
# Caught an error!
# <simpleError in integrate(f, lower = lowerBound, upper = upperBound): the integral is
   probably divergent>
# [1] 0
```

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 probabiliate, 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

```

  # Integrate a function and return False if any errors are generated
  safe_integrate <- function(f, d = c(-Inf, Inf)){
    tryCatch(integrate(Vectorize(f), d[1], d[2]),
             error = function(e){
               print(e)
               FALSE
             })
  }

  # Check if a function is positive in a given interval
  is_positive <- function(f, d = c(-10000000, 10000000), step=10000000){
    # Set maximum lower and upper bounds
    if(d[2] == Inf)
      d[2] = 10000000;
    if(d[1] == -Inf)
      d[1] = -10000000;
    # Calculate values in that intervals
    vals <- seq(d[1], d[2], len=step)
    # Check if all values are positive
    all(f(vals)>=0)
  }

  check_pdf <- function(f, d = c(-Inf, Inf)){
    # Check if the function is positive
    if(!is_positive(f, d)){
```

```
        print("Function is negative")
        return(FALSE)
      }
      # Safe integrate, handle errors
      i = safe_integrate(f, d)
      # Cannot integrate
      if(typeof(i)=='logical' && i==FALSE)
        return(FALSE)
      v = i $ value
      round(v) == 1
    }
```

Teste

```
f1 <- function(x){
  if (x > 0 && x < 2){
    3/8 * (4*x-2*x^2)
  }else{
    0
  }
}

check_pdf(f1)
# TRUE

f2 <- function(x){
  x * (4*x-2*x^2)
}

check_pdf(f2)
# 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, functia normalizata.

Cod

```
    # Continuous RV class

    Continuous_RV <- setClass(
      "continuous_RV",
      slots = list(
        pdf = "function",
        normalized = "function",
        cdf = "function",
        norm_constant = "numeric",
        mean = "numeric"
      )
    )

```

```
init <- function(.Object, pdf)
  # initialization
  setMethod("init", "Continuous_RV", function(.Object, pdf){
    tryCatch({
      rand_var <- .Object
      rand_var@pdf = pdf
      rand_var@norm_constant = compute_normalizing_constant(pdf)
      rand_var@normalized <- rand_var@pdf * rand_var@norm_constant
      rand_var@cdf <- rand_var@pdf
      rand_var@deviation <- rand_var@pdf
      rand_var@mean <- E(rand_var@pdf)
    })
  })


show <- function(.Object)
  setMethod("show", "Continuous_RV", function(.Object){
    print(rand_var@norm_constant)
    print(rand_var@pdf)
    print(rand_var@mean)
    print(rand_var@normalized)
  })

# compute the mean
E <- function (probability_density_function, normalization_constant = 1){
  # int (x * pdf(x)dx)
  forIntegration <- function(x){
    return (x * pdf(x) / normalization_constant)
  }
  # try to compute the integral
  tryCatch(
    expr = {
      integral <- integrate(forIntegration, lower = -Inf, upper = Inf)$value
      return(integral)
    },
    error = function(e){
      message('Caught an error when calculating the mean!')
      print(e)
      return(0)
    },
    warning = function(w){
      message('Caught a warning!')
      print(w)
    },
    finally = {
      message('')
    })
}
```

Teste

```
f <- function(x){
  return (exp((-x^2)/2))
}
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 ntr-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
```

```
    xs <- seq(a, b, 0.01)
    ys <- c()
    for (x in xs) {
      ys = append(ys, F(x))
    }
    plot(xs, ys, col="red", type="l", main=noquete(paste(name, " CDF")), xlab="x", ylab
="y")
  }

  # o folosesc pentru repartitiile carora
  # nu le stim functia de repartitie
  # ne folosim de pdf
  # afisaza graficul repartitiei
  plot_generic_repartition <- function(f, a, b) {
    # validez pdf
    if (!check_density(f, a, b)) {
      return()
    }

    xs <- seq(a, b, 0.01)
    ys <- c()
    for (x in xs) {
      ys = append(ys, get_CDF_from_PDF(f, x))
    }
    plot(xs, ys, col="red", type="l", main="CDF", xlab="x", ylab="y")
  }


  # functia primeste ca parametrii numele repartitiei,
  # un flag care marcheaza daca vrem graficul densitatii,
  # sau al repartitiei, si, in plus, parametrii corespunzatori
  # repartitiei selectate
  # daca repartitia nu exista, sau daca parametrii nu corespund
  # repartitiei alese, se va intoarce un mesaj corespunzator
  parse_known_repartition <- function(name, CDF=FALSE, ...) {
    params <- list(...)
    if (name == "uniform") {
      if (!is.null(params$a) && !is.null(params$b)) {
        a <- params$a
        b <- params$b
        if (a >= b) {
          return("parametri incorecti")
        }

        f <- function(x) 1 / (b - a)
        F <- function(x)(x - a) / (b - a)
        if (CDF) {
          plot_repartition(F, a, b, name)
        }
        else {
          plot_density(f, a, b, name)
        }
      }
      else {
        return("parametrii necesari nu au fost pasati")
      }
    }
    else if (name == "exp") {
      if (!is.null(params$lambda)) {
        lambda <- params$lambda
        if (lambda <= 0) {
          return("parametrii incorecti")
        }

        f <- function(x) (lambda * exp(1)^(-lambda * x))
        F <- function(x) (1 - exp(1)^(-lambda * x))
        if (CDF) {
          plot_repartition(F, 0, 50, name)
        }
        else {
```

```
            plot_density(f, 0, 50, name)
          }
        }
        else {
          return("parametrii necesari nu au fost pasati")
        }
      }
      else if (name == "normal") {
        if (!is.null(params$mu && !is.null(params$sigma))) {
          mu <- params$mu
          sigma <- params$sigma
          if (sigma <= 0) {
            return("parametrii incorecti")
          }

          f <- function(x) ((1 / (sigma * sqrt(pi * 2)))*(exp(1)^((-(x - mu)^2)/(2  *
    sigma ^ 2))))
          F <- function(x) (pnorm(x, mu, sigma))
          if (CDF) {
            plot_repartition(F, -25, 25, name)
          }
          else {
            plot_density(f, -25, 25, name)
          }
        }
        else {
          return("parametrii necesari nu au fost pasati")
        }
      }
      else if (name == "pareto") {
        if (!is.null(params$m) && !is.null(params$alpha)) {
          m <- params$m
          alpha <-params$alpha
          if (alpha <= 0 || m <= 0) {
            return("parametrii incorecti")
          }

          f <- function(x) (alpha * m^alpha) / (x ^ (alpha + 1))
          F <- function(x) (1 - (m ^ alpha) / (x ^ alpha))
          if (CDF) {
            plot_repartition(F, m, m + 50, name)
          }
          else {
            plot_density(f, m, m + 50, name)
          }
        }
      }
      else if (name == "cauchy") {
        if (!is.null(params$location && !is.null(params$scale))) {
          location <- params$location
          scale <- params$scale
          if (scale <= 0) {
            return("parametrii incorecti")
          }

          f <- function(x) 1 / (pi * scale * (1 + ((x - location) / (scale))^2))
          F <- function(x) (1 / pi) * atan((x - location) / scale) + 1 / 2

          if (CDF) {
            plot_repartition(F, -25, 25, name)
          }
          else {
            plot_density(f, -25, 25, name)
          }
        }
      }
      else if (name == "logistic") {
        if (!is.null(params$mu && !is.null(params$s))) {
          mu <- params$mu
          s <- params$s
```

```
        if (s <= 0) {
          return("parametrii incorecti")
        }

        f <- function(x) (exp(1) ^ ((mu - x) / s) / (s * (1 + exp(1) ^ (mu - x) / s) ^ 2))
        F <- function(x) 1 / (1 + exp(1) ^ (-(x - mu) / s))

        if (CDF) {
          plot_repartition(F, -25, 25, name)
        }
        else {
          plot_density(f, -25, 25, name)
        }
      }
    }
    else if (name == "weibull") {
      if (!is.null(params$scale && !is.null(params$shape))) {
        scale <- params$scale
        shape <- params$shape
        if (scale <= 0 || shape <= 0) {
          return("parametrii incorecti")
        }

        f <- function(x) {
          if (x >= 0) (shape / scale) * (x / scale) ^ (shape - 1) * (exp(1) ^ (-(x / scale) ^ shape))
          else 0
        }
        F <- function(x) {
          if (x >= 0) 1 - (exp(1) ^ (-(x / scale) ^ shape))
          else 0
        }

        if (CDF) {
          plot_repartition(F, 0, 50, name)
        }
        else {
          plot_density(f, 0, 50, name)
        }
      }
    }
    else {
      print("repartitie necunoscuta")
    }
  }
```

Teste

```
parse_known_repartition("uniform", FALSE, a=0, b=10)
parse_known_repartition("uniform", TRUE, a=0, b=10)
parse_known_repartition("exp", FALSE, lambda=2)
parse_known_repartition("exp", TRUE, lambda=2)
parse_known_repartition("normal", FALSE, mu=0, sigma=1)
parse_known_repartition("normal", TRUE, mu=0, sigma=1)
parse_known_repartition("pareto", FALSE, m=3, alpha=1)
parse_known_repartition("pareto", TRUE, m=3, alpha=1)
parse_known_repartition("cauchy", FALSE, location=0, scale=1)
parse_known_repartition("cauchy", TRUE, location=0, scale=1)
parse_known_repartition("logistic", FALSE, mu=0, s=1)
parse_known_repartition("logistic", TRUE, mu=0, s=1)
parse_known_repartition("weibull", FALSE, scale=1, shape=2)
parse_known_repartition("weibull", TRUE, scale=1, shape=2)
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

```

    # Calculating central moment of order o
    central_moment <- function(f, o){
      m = medium(f)
      f_new <- function(x){
        ((x - m) ^ o) * f(x)
      }
      res = safe_integrate(f)
      if(typeof(res)=="logical" && res==FALSE){
        print("Momentul nu exist !")
        return(0)
      }
      res $ value
    }

    # Calculating initial moment of order o
    initial_moment <- function(f, o){
      f_new <- function(x){
        x ^ o * f(x)
      }
      res = safe_integrate(f)
      if(typeof(res)=="logical" && res==FALSE){
        print("Momentul nu exist !")
        return(0)
      }
      res $ value
    }
```

Teste

```
  f = function (x) {
    if(x<0 || x>1)
      return(0)
    x ^ 3
  }

  initial_moment(f, 2)
  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

$Var(X) = E(X^2) - E(X)^2$.

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

```
# am nevoie de functia g, pdf-ul lui X si domeniul de definitie
median_and_dispersion <- function(g, fx, domeniu_valori) {
  # Y = g(X) poate fi vazut ca o v.a.c. noua cu pdf-ul g(f(x)), ambele functii fiind continue le compun

  # folosesc formula pt media functiilor de x
  # folosesc integral(din pachetul pracma) deoarece integer imi da eroare cand incerc sa integrez la +-inf
  e_y <- integral(Vectorize(function(x){g(x) *fx(x)}),domeniu_valori[1],domeniu_valori[2])

  # pt dispersie, mai folosesc o data formula mediei functiilor de x
  # pentru a afla X^2
  e_y2 <- integral(Vectorize(function(x){x^2 * g(x) *fx(x)}),domeniu_valori[1],domeniu_valori[2])
  dispersie <- e_y2 - e_y^2

  print(paste("Media: ",e_y))
  print(paste("Dispersia:", dispersie))
}
```

Teste

```
# exemplu
f1 <- function(x)(x^2)
f2 <- function(x) (1 * exp(1)^(-1 * x))

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

```
####
# Cerinta: Crearea unei func ii P care permite calculul diferitelor tipuri de
# probabilit  i asociate unei variabile aleatoare continue(similar func iei P din pachetul discreteRV)
#
# Header functie: myP(f, p)
#    - unde f este o functie densitate de probabilitate (pdf)
#    - iar p este un string ce reprezinta probabilitatea (conditionata sau independenta).
#
# Obligatoriu, var se va afla in stanga operatorului
####

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,
    mi-am gasit operatorul si returnez parametri
23        for(op in operatii_posibile) {
24
25          # am dat split corect => in stanga am variabila, in dreapta am bound-ul
26          split <- unlist(strsplit(expresie, op, fixed = TRUE))
27          splitSize <- length(split)
28
29          if (splitSize == 2) {
30            # returnez (v.a.c, operatie, bound)
31            return (c(split[1],op,split[2]))
32          }
33        }
34        #daca am ajuns aici => nu am operator, deci eroare
35        return(c(-1))
36
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
72    # daca am o singura expresie e prob independenta
73    prob_independenta <- function(expresie) {
74
75      # scot parametri din expresie
76      parametri <- parseaza_expresie(expresie)
77
78      if(length(parametri) != 3)
79        return("Eroare la parsarea probabilitatii")
80
81      # 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 depinde de b) = P(a 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 | x > 5) = 0
117     if (op1 %in% c("<=","<") && op2 %in% c(">=", ">") && bound1 >=bound2)
118       return (0);
119
120     # p(x > 5 | 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
    intervalul cel mai restrans
125     # p(x> 3 | x>7) => iau (-inf,3)
126     if(op1 %in% c(">=",">") && op2 %in% c(">=",">"))
127       if(bound1 > bound2)
128         return (ans1/ans2)
129       else return (1);
130    # p( x<3 | x<7) => iau (-inf,3)
131     if(op1 %in% c("<=","<") && op2 %in% c("<=","<"))
132       if(bound1 < bound2)
133       return (ans1/ans2)
134       else return (1)
135
136     ## daca nu e niciunul de mai sus, e intersectie de forma x > 5 | x < 7 si fac
    diferenta
137     ## din cdf mai mare scad cdf mai mic si mi da bucata de dintre => intersectia
138     return ((cdf(compute_bound(bound2))-cdf(compute_bound(bound1)))/ans2)
139
140   }
141
142   ####### Body functie principala ##########
143
144     # parsez parametri
145     parti = unlist(strsplit(p, "|", fixed = TRUE))
146     len = paste(length(parti))
147     switch(len,
148            "0" = return("Eroare"),
149            "1" = return(prob_independenta(p)),
```

```
            "2" = return(prob_conditionata(parti[1],parti[2])),
            )
      return ("eroare");

    }
```

Teste

```
     g <- function (x) {
          fun <- 0.1*(3*(x^2) + 1)
          fun[x<0] = 0
          fun[x>2]=0
          return ( fun )
     }

    h <- function(x)(dunif(x))

    myP(g,"x>1|x<1.5") # 0.5897078
    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 n mod uzual acea repartiție, semnificația parametrilor, media, dispersia etc.

Rezolvare

Am afisate detalii despre diverse distibutii din curs, dar si gasite in urma cautarilor proprii, precum "Cauchy", sau "Weibull".

Dificultati si Abordare

Dificultatea cerintei a constat in gruparea datelor impreuna. Dupa parcurgerea mai multor pachete auxiliare am ales sa folosesc obiectul *Dictionary* din *mlr3misc*. Modul de folosire este putin complicat, intrucat se bazeaza pe utilizarea unui *R6Class*, dar in final, rezultatul este bine structurat si usor de folosit pentru distribuirea si afisarea datelor.

Cod

```

library(mlr3misc)
library(R6)

data = Dictionary$new()

item = R6Class("Item")

########## UNIFORM ###########

data_aux = Dictionary$new()
data_aux$add("Sursa", R6Class(public = list(print = "Curs + Wikipedia")))
data_aux$add("Notatie", R6Class(public = list(print = "U(a, b)")))
data_aux$add("Parametri", R6Class(public = list(print = "-Inf < a < b < Inf")))
data_aux$add("Domeniu", R6Class(public = list(print = "x in [a, b]")))
data_aux$add("PDF", R6Class(public = list(print = "1 / (b - a)")))
data_aux$add("CDF", R6Class(public = list(print = "(x - a) / (b - a)")))
data_aux$add("Mediana", R6Class(public = list(print = "(1 / 2) * (a + b)")))
data_aux$add("Media", R6Class(public = list(print = "(1 / 2) * (a + b)")))
data_aux$add("Utilizari", R6Class(public = list(print = "In economie, pentru managerierea
    inventarului, cand un produs complet nou este analizat, distributia uniforma e cea
    mai utila.")))
```

```
data$add("uniform", data_aux)

########## EXPONENTIAL ############

data_aux = Dictionary$new()
data_aux$add("Sursa", R6Class(public = list(print = "Curs + Wikipedia")))
data_aux$add("Parametri", R6Class(public = list(print = "labmda > 0")))
data_aux$add("Domeniu", R6Class(public = list(print = "x in [0, +Inf]")))
data_aux$add("PDF", R6Class(public = list(print = "lambda * exp(1)^(-lambda * x)")))
data_aux$add("CDF", R6Class(public = list(print = "1 - exp(1)^(-lambda * x)")))
data_aux$add("Media", R6Class(public = list(print = "1 / labmda")))
data_aux$add("Mediana", R6Class(public = list(print = "ln2 / lambda")))
data_aux$add("Utilizari", R6Class(public = list(print = "Dupa observarea unui set de n
    puncte dintr-o distibutie exponentiala necunoscuta, se pot face predictii destul de
    precise despre viitor, pe baza datelor sursa.")))
data$add("exponential", data_aux)

########## NORMAL ############

data_aux = Dictionary$new()
data_aux$add("Sursa", R6Class(public = list(print = "Curs + Wikipedia")))
data_aux$add("Notatie", R6Class(public = list(print = "N(mu, sigma^2)")))
data_aux$add("Parametri", R6Class(public = list(print = "mu in R, sigma ^ 2 >= 0")))
data_aux$add("Domeniu", R6Class(public = list(print = "x in [-Inf, +Inf]")))
data_aux$add("PDF", R6Class(public = list(print = "(1 / (sigma * sqrt(pi * 2)))*(exp(1)
    ^((-(x - mu)^2)/(2  * sigma ^ 2)))")))
data_aux$add("CDF", R6Class(public = list(print = "(1 / 2) * (1 + erf((x - mu) / (sigma *
     sqrt(2)))")))
data_aux$add("Media", R6Class(public = list(print = "mu")))
data_aux$add("Mediana", R6Class(public = list(print = "mu")))
data_aux$add("Utilizari", R6Class(public = list(print = "Poate fi observata in ziua de zi
     cu zi. De exempul distributia notelor unor studenti la examenul de Probabilitati si
    Statistica.")))
data$add("normal", data_aux)

########## PARETO ############

data_aux = Dictionary$new()
data_aux$add("Sursa", R6Class(public = list(print = "Curs + Wikipedia")))
data_aux$add("Parametri", R6Class(public = list(print = "m > 0, alpha > 0")))
data_aux$add("Domeniu", R6Class(public = list(print = "x in [m, +Inf]")))
data_aux$add("PDF", R6Class(public = list(print = "(alpha * m^alpha) / (x ^ (alpha + 1))"
    )))
data_aux$add("CDF", R6Class(public = list(print = "(1 - (m ^ alpha) / (x ^ alpha))")))
data_aux$add("Media", R6Class(public = list(print = "Inf, alpha <= 1\n (alpha * m) / (
    alpha - 1), alpha > 1")))
data_aux$add("Mediana", R6Class(public = list(print = "m * sqrt_alpha(2)")))
data_aux$add("Utilizari", R6Class(public = list(print = "A fost folosita de catre
    creatorul ei, Vilfredo Pareto, pentru analiza distributiei banilor intre indivizi.
    Aceasta arata destul de precis cum cea mai mare cantitate de bani este controlata de
    un procent foarte mic din populatie.")))
data$add("pareto", data_aux)

########## CAUCHY ############

data_aux = Dictionary$new()
data_aux$add("Sursa", R6Class(public = list(print = "Curs + Wikipedia")))
data_aux$add("Parametrii", R6Class(public = list(print = "x0, y > 0")))
data_aux$add("Domeniu", R6Class(public = list(print = "x in [-Inf, Inf]")))
data_aux$add("PDF", R6Class(public = list(print = "1 / (pi * y * (1 + (x - x0) / y)^2)"))
    )
data_aux$add("CDF", R6Class(public = list(print = "(1 / pi) * arctan((x - x0) / y) + (1/
    2)")))
data_aux$add("Mediana", R6Class(public = list(print = "x0")))
data_aux$add("Media", R6Class(public = list(print = "undefined")))
data_aux$add("Utilizari", R6Class(public = list(print = "Distributia Cauchy este folosita
     des in analiza distributiei observatiilor facute de catre oameni de stiinta despre
    despre obiecte care se invartesc.")))
data$add("cauchy", data_aux)

```

```
########## WEIBULL ###########

data_aux = Dictionary$new()
data_aux$add("Sursa", R6Class(public = list(print = "Wikipedia")))
data_aux$add("Parametri", R6Class(public = list(print = "lambda in (0,+inf) da scara, k
    in (0,+inf) da forma")))
data_aux$add("Domeniu", R6Class(public = list(print = "x in [0, +inf]")))
data_aux$add("PDF", R6Class(public = list(print = "k/lambda * (x/lambda) ^ (k-1) * e^(-(x
    /lambda)^k), x>=0, altfel 0")))
data_aux$add("CDF", R6Class(public = list(print = "1-e^((-x/lambda)^k), x>=0, altfel 0"))
    )
data_aux$add("Mediana", R6Class(public = list(print = "lambda * (ln 2)^(1/k)")))
data_aux$add("Media", R6Class(public = list(print = "(lambda * gamma(1+1/k)")))
data_aux$add("Utilizari", R6Class(public = list(print = "In meteorologie, folosita la
    prezicerea distributiei vitezei vantului, in ingineria electrica.")))
data$add("weibull", data_aux)


########## LOGISTIC ###########

data_aux = Dictionary$new()
data_aux$add("Sursa", R6Class(public = list(print = "Curs + Wikipedia")))
data_aux$add("Parametrii", R6Class(public = list(print = "u, s>0")))
data_aux$add("Domeniu", R6Class(public = list(print = "x in [-Inf, Inf]")))
data_aux$add("PDF", R6Class(public = list(print = "(e^(-(x-u)/s))/(s*(1+e^(-(x-u)/s)))^2
    ")))
data_aux$add("CDF", R6Class(public = list(print = "1/(1+e^(-(x-u)/s))")))
data_aux$add("Mediana", R6Class(public = list(print = "u")))
data_aux$add("Media", R6Class(public = list(print = "u")))
data_aux$add("Utilizari", R6Class(public = list(print = "Este adesea folosita in regresia
     logistica si in Retelele Neuronale de tip 'feedforward'.")))
data$add("logistic", data_aux)


########## FUNCTIE DE AFISARE ###########

output_details <- function(distribution_name) {
  out_data = data$get(distribution_name)
  for (key in out_data$keys()) {
    print(noquote(paste(key, ": ", out_data$get(key)$print, sep="")))
  }
}
```

Teste

```

output_details("uniform")
output_details("exponential")
output_details("normal")
output_details("pareto")
output_details("cauchy")
output_details("logistic")
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

```r
#function to integrate the probability density function
integrate_pdf <- function (f, X) {
  tryCatch (
    expr = {
      integrate (f, 0, X)$value
    },
    error = function (e) {
      print(e)
    }
  )
}

# compute the inverse function
inverse_function <- function (f, u, lowerBound = -10000, upperBound = 10000) {
  tryCatch (
    expr = {
      uniroot((function (x) f(x) - u), lower = lowerBound, upper = upperBound, extendInt
    = 'yes')
    },
    error = function (e) {
      print(e)
    }
  )
}

random_variable_generator <- function(f, n, lowerBound, upperBound) {
  # generates random deviates
  values <- runif(n, min = 0, max = 1)
  # combines its arguments into a list
  r <- c()
  for (v in values) {
    r <- append (r, inverse_function(function (x) (integrate_pdf(f, x)), v, lowerBound,
    upperBound)[1]$root)
  }
  r
}
```

Teste

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

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

$cov(x, y) = (\int_c^d \int_a^b x * y * f(x, y)\, dx\, dy) - \mu x * \mu y$
$cor(x, y) = \frac{cov(x,y)}{\sigma x * \sigma y}$

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

```
# conditia 1 pentru o functie de densitate corecta
# f trebuie sa aiba valori pozitive pentru
# toate valorile din domeniu
# cum in domeniu sunt o infinitate de puncte
# am ales sa verific 'pe sarite' o parte dintre ele
check_positive_2d <- function(f, ab, cd, step) {
  for (x in seq(ab[1], ab[2], step)) {
    for (y in seq(cd[1], cd[2], step)) {
      if (f(x, y) < 0) return(FALSE)
    }
  }

  return(TRUE)
}

# conditia 2 pentru o functie de densitate corecta
# f terbuie sa aiba probabilitatea totala egala cu 1
# pentru a calcula integrala dubla am folosit functia
# 'integral2' din pachetul 'pracma'
check_integral_is_1 <- function(f, ab, cd) {
  integral <- integral2(f, ab[1], ab[2], cd[1], cd[2])$Q
  return(abs(integral - 1) < 0.0000000001)
}

# intoarce o noua functie g(y) definita
# ca f(x, y), unde x este o valoare fixata
f_set_x <- function(x, f) function(y) f(x, y)

# intoarce o noua functie g(x) definita
# ca f(x, y), unde y este o valoare fixata
f_set_y <- function(y, f) function(x) f(x, y)

# calculeaza densitatea marginala in X
# il setam pe x si integram peste y
x_marginal_pdf <- function(f, x, c, d) {
  return (integrate(f_set_x(x, f), c, d)$value)
}

# calculeaza densitatea marginala in Y
# il setam pe y si integram peste x
y_marginal_pdf <- function(f, y, a, b) {
  return (integrate(f_set_y(y, f), a, b)$value)
}

# calculeaza densitatea in X conditionata de Y = y
h_x_conditioned_by_y <- function(y, x, f, c, d) {
  f(x, y) / x_marginal_pdf(f, x, c, d)
}

# calculeaza densitatea in Y conditionata de X = x
h_y_conditioned_by_x <- function(x, y, f, a, b) {
  f(x, y) / y_marginal_pdf(f, y, a, b)
}

marginal_conditional_densities <- function(f, x, y, ab, cd) {

  tryCatch({
    if (!check_positive_2d(f, ab, cd, 0.01)) {
      print("functia nu este pozitiva pentru valorile din domeniu")
      return ()
    }

```

```
        if (!check_integral_is_1(f, ab, cd)) {
          print("probabilitatea totala este diferita de 1")
          return ()
        }

        print(paste("testez cu x=", x, " y=", y, " pe [", ab[1], ",",
                    ab[2], "]x[", cd[1], ",", cd[2], "]", sep=""))

        # densitatea marginala pentru X cu x fixat
        print(paste("pdf marginal cu x fixat:", x_marginal_pdf(f, x, cd[1], cd[2])))

        # densitatea marginala pentru Y cu y fixat
        print(paste("pdf marginal cu y fixat:", y_marginal_pdf(f, y, ab[1], ab[2])))

        # densitatea conditionata in x cand Y = y
        print(paste("pdf conditionat de y=", y, ": ", h_x_conditioned_by_y(y, x, f, cd
    [1], cd[2]), sep=""))

        # densitatea conditionata in y cand X = x
        print(paste("pdf conditionat de x=", x, ": ", h_y_conditioned_by_x(x, y, f, ab
    [1], ab[2]), sep=""))
      }, error = function(e) {
        print(paste("A aparut o eroare", e, sep = " "))
      })
    }
```

Teste

```
    test_1 <- function() {
      f <- function(x, y) (8 / 3) * x ^ 3 * y
      marginal_conditional_densities(f, 0.5, 1.5, c(0, 1), c(1, 2))
    }

    test_2 <- function() {
      f <- function(x, y) (3 / 2) * (x ^ 2 + y ^ 2)
      marginal_conditional_densities(f, 0, 0.5, c(0, 1), c(0, 1))
    }

    test_1()
    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 urmatoarele 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 in curs:

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

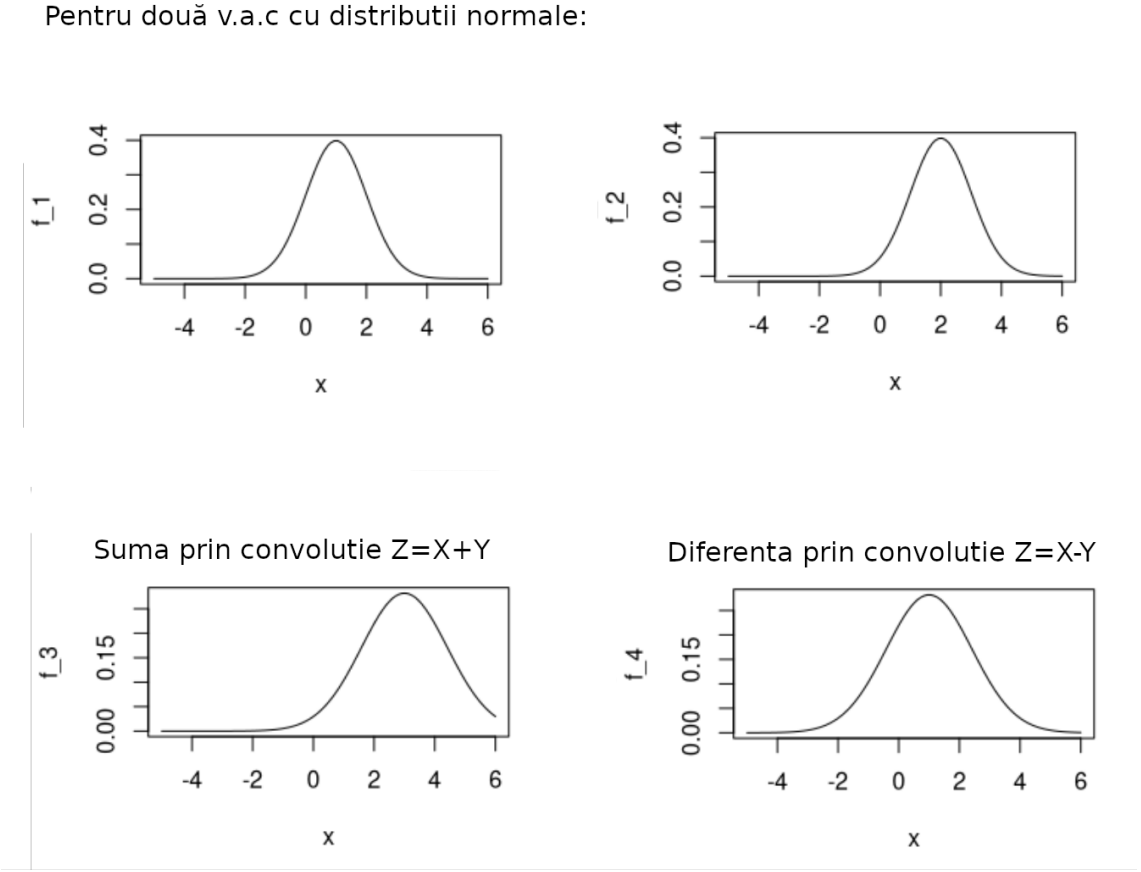
Cod

```
# suma a doua variabile aleatoare continue independente folosind formula de convolutie

convolution_sum <- function(fx,fy) {
  res <-function(z) {
    integrate(function(y) {
      fx(z-y) * fy(y)
      },-Inf,Inf) $ value
  }
}

# diferenta a doua variabile aleatoare continue independente folosind formula de convolutie
convolution_diff <- function(fx,fy) {
  res <-function(z) {
    integrate(function(y) {
      g(y-z)*f(y)
    },-Inf,Inf) $ value
  }
}
```

Teste

```
# doua distributii normale, suma si diferenta facuta cu formula de convolutie
f_1 <- function(x)(dnorm(x,mean=1))
f_2 <- function(x) (dnorm(x,mean=2))
f_3 <- Vectorize(convolution_sum(f_1, f_2))
f_4 <- Vectorize(convolution_diff(f_1,f_2))

plot(f_1,from=-10,to=10,type="l")
plot(f_2,from=-10,to=10,type="l")
plot(f_3(t),from=-10,to=10,type="l")
plot(f_4(t),from=-10,to=10,type="l")
```

Output



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.