Pachet pentru lucru cu variabile aleatoare continue

Țânțaru Dragoș-Constantin, Vasiliu Florin
Vintilă Eduard-Ionuț, Ristea Mihai-Cristian
Grupa 244

4 februarie 2021

1 Introducere

Pachetul oferă un set de operații uzuale în lucrul cu variabile aleatoare continue, de la calcularea diverselor probabilități, la determinarea densităților condiționate și chiar la animații cu graficele unor repartiții cunoscute. Mai exact, pachetul permite:

- construirea unui obiect de tip variabilă aleatoare continuă, unidimensională sau bidimensională, cu suport configurabil de utilizator
- calcularea probabilităților, printr-un apel de forma $P(X <= x)$ (avem implementate operațiile <, <=, >, >=, ==)
- calcularea probabilităților condiționate de forma $P(X \text{ op}_1 \ x_1 \mid X \text{ op}_2 \ x_2)$, unde op_1 și op_2 sunt oricare din operațiile definite mai sus
- calcularea probabilităților condiționate de forma $P(X \text{ op } x \mid Y == y)$, unde Y e orice variabilă aleatoare continuă și op, din nou, orice operație de mai sus
- calcularea mediei, dispersiei și momentelor centrate sau inițiale ale unei variabile aleatoare continue
- aplicarea unor funcții continue asupra variabilelor aleatoare continue și prelucrarea rezultatelor obținute
- afișarea graficelor a diverselor repartiții și densități obișnuite
- observarea efectelor parametrilor asupra densității beta prin intermediul unei animații programate în R
- afișarea unor fișe de sinteză despre repartițiile și densitățiile obișnuite
- și multe alte funcționalități pe care le-am detaliat în documentație

În mod specific, am rezolvat cerințele 1, 2, 3, 4, 5, 6, 7, 8, 10 și 11 din cerințele proiectului 1.

Dificultățile cele mai importante întâmpinate la implementarea proiectului au fost folosirea eficientă a funcției *integrate* și gestionarea corectă a variabilelor R, în contextului scoping-ului făcut de R (problemă explicată în detaliu la documentația pentru exercițiul 5). Pe scurt, pe prima am depășit-o scriind o funcție specifică pachetului *integrala*, care face suma integralelor pe suportul variabilelor și aplicând funcția $Vectorize$ asupra tuturor funcțiilor pe care le-am integrat, iar a doua folosind o funcție de tip „factory".

Principala limitare a pachetului este tipul de densități comune cu care poate lucra. Deoarece nu am reușit să găsim un mod bun de a stoca suporturi care nu sunt reuniuni de dreptunghiuri $[a, b] \times [c, d]$, pachetul nu permite lucrul cu variabile bidimensionale care au suporturi în care ori x, ori y sunt definite pe intervale care depind de cealaltă variabilă. A nu se înțelege că nu se poate lucra cu variabile aleatoare continue dependente, problema apare doar când una dintre variabilele aleatoare are suportul dependent de cealaltă. În schimb, dacă suportul e de forma $\bigcup_i([a_i, b_i] \times [c_i, d_i])$, unde niciuna dintre a_i, b_i, c_i, d_i nu depinde de x sau y, se poate folosi fără probleme pachetul.

În continuare, vom lua fiecare exercițiu pe rând și vom documenta implementarea, ideea de rezolvare și problemele întâmpinate.

2 Cerința 1

1) Fiind dată o funcție f , introdusă de utilizator, determinarea unei constante de normalizare k. În cazul în care o asemenea constantă nu există, afișarea unui mesaj corespunzător către utilizator

Scurtă descriere a antetului funcției:

Parametrul	Tipul	Descriere
Func	function	Funcția data ca parametru
sup	listă de liste	Suportul funcției

```r
Nor_constant <- function(Func, sup) {

  sum <- 0

  for (i in sup) {
    tryCatch(
    sum <- sum + integrate(Vectorize(Func), i[1], i[2], abs.tol = 0)$
       value,
    error= function(err) {
      stop("Integrala e divergenta sau functia nu e integrabila") #
         daca integrala nu poate fi calculata returnez un mesaj de
         eroare
    }
    )

    if (i[1] == -Inf && i[2] == Inf) {
      i[1] <- -1000
      i[2] <-  1000
    }
    else if (i[1] == -Inf)
    i[1] <- i[2] - 1000
    else if (i[2] ==  Inf)
    i[2] <- i[1] + 1000

    if(any(sapply(seq(i[1], i[2], length.out = 1000), Func) < 0))
    stop("Functie negativa") # daca functia are valori negative nu
       pot calcula constanta de normalizare

  }

  if(sum == 0)
  stop("Nu exista constanta de normalizare pentru functia data") #
     daca integrala = 0 inseamna ca nu exista constanta de
     normalizare

  const <- 1 / sum
  return (const)

}
```

Va fi parcurs suportul funcției dată ca parametru, și vom calcula suma pe fiecare interval din suport. În același timp, verificăm dacă funcția e pozitivă pe fiecare interval din suport, folosind $any(sapply(seq(i[1], i[2], length.out = 1000), Func) < 0)$, pentru a alege 1000 de valori echidistante.

La final, dacă integrala este 0 înseamnă că nu există constantă de normalizare și afișează un mesaj corespunzător. În caz contrar, calculează constanta și o returnează.

3 Cerința 2

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

Dificultate întâmpinată: funcția `integrate` returnează valori eronate pentru integranzi cu valoarea 0 într-o mare parte a domeniului. Ca remediu, am ales să transmitem printr-un parametru suportul funcției de integrat (detalii în comentariul din cod).

De asemenea, în cazul funcțiilor pe ramuri, `Vectorize()` devine o necesitate: evaluarea condițiilor din if-uri (de exemplu, $x > 0$) ia în considerare doar primul element al vectorului Boolean $x > 0$. Apare astfel conflict cu modul de lucru al procedurii `integrate`, care evaluează funcția-argument pe un vector de mostre, nu punct-cu-punct.

Definiția funcției este următoarea:

```
1  # Suportul functiei f este reuniunea intervalelor specificate prin parametrul
2  # sup - o lista de vectori de cate doua elemente, reprezentand extremitati de
3  # interval. f este densitate de probabilitate - deci dp(f, sup)==TRUE - daca:
4  # a) f(x) >= 0 pentru orice x din suport,
5  # b) suma integralelor de f peste fiecare interval din sup este 1.
6
7  # Din documentatia pentru integrate: "f must accept a vector of inputs and
8  # produce a vector of function evaluations at those points". Considerand ca
9  # utilizatorul introduce functia dorita in regim scalar -> scalar, aplicand
10 # Vectorize() se obtine argumentul dorit pentru integrate.
11 dp <- function(f, sup) {
12
13   sum <- 0
14   for (i in sup) {
15     tryCatch(
16     sum <- sum + integrate(Vectorize(f), i[1], i[2], abs.tol = 0)$value == 1,
17     error = function(e) {
18       print(sprintf("Integrala divergenta in intervalul [%.2f, %.2f]!", i[1], i[2]))
19       return(FALSE)
20     })
21
22     if (i[1] == -Inf && i[2] == Inf) {
23       i[1] <- -1000
24       i[2] <-  1000
25     }
26     else if (i[1] == -Inf)
27     i[1] <- i[2] - 1000
28     else if (i[2] ==  Inf)
29     i[2] <- i[1] + 1000
```

```
30
31        if (any(sapply(seq(i[1], i[2], length.out = 1000), f) < 0)) {
32          print(sprintf("Valoare negativa in intervalul [%.2f, %.2f]!", i
                [1], i[2]))
33          return(FALSE)
34        }
35      }
36      sum == 1
37    }
```

Exemple:

```
> dp(function(x) 3*x^2, list(c(0, 1)))
[1] TRUE

> f <- function(x) if (x < 0) 1 + x else 1 - x
> dp(f, list(c(-1, 1)))
[1] TRUE

> dp(function(x) x, list(c(1, 3), c(-1, 0)))
[1] "Valoare negativa in intervalul [-1.00, 0.00]!"
[1] FALSE

> dp(function(x) x, list(c(0, Inf)))
[1] "Integrala divergenta in intervalul [0.00, Inf]!"
[1] FALSE
```

4 Cerința 3

3) Crearea unui obiect de tip variabilă aleatoare continuă pornind de la o densitate de probabilitate introdusă de utilizator. Funcția trebuie să aibă opțiunea pentru variabile aleatoare unidimensionale și respectiv bidimensionale.

Pentru a facilita lucrul cu variabile aleatoare continue atât unidimensionale, cât si bidimensionale, am definit clasa de tip S4 numită **contRV**. Toate instanțele acestei clase vor reține o mulțime de informații necesare pentru a efectua diverse operații pe variabile aleatoare continue. Mai precis, fiecare obiect **contRV** va avea asociat:

- O densitate de probabilitate.
- Repartiția variabilei aleatoare continue
- Un Boolean ce indică dacă variabila aleatoare este bidimensională.
- Suportul densității de probabilitate. În cazul variabilelor unidimensionale, acesta este reprezentat de o listă de intervale închise. Pentru o v.a bidimensională (X, Y), suportul este reținut sub forma unei liste ce conține suporturile lui X și Y.
- Pentru o v.a unidimensională X, o referință către v.a bidimensională (X, Y), în cazul in care X s-a format în urma determinării densității marginale. Se folosește pentru a avea acces cu ușurință la densitatea comună în calculul unor probabilități ce implică pe X și Y.

Motivul pentru care este necesară specificarea suportului densității la crearea unui obiect de tip **contRV** este următorul: la calculul integralelor unde unul dintre capete nu este un număr finit, comportamentul funcției `integrate()` poate produce rezultate neașteptate. Astfel, restricționând domeniul la punctele în care integrandul ia valori nenule, calculul integralei devine mult mai precis. În stadiul actual însă, permitem ca suportul densității să fie specificat doar ca o reuniune de intervale închise în cazul v.a unidimensionale, sau dreptunghiuri în cazul celor bidimensionale.

Definiția clasei este următoarea:

```
1  setClass("contRV", representation (
2  densitate="function",
3  val="function",
4  bidimen="logical",
5  suport="list",
6  ref_va_bidimen = "contRV_or_NULL"
7  ))
8
9  # Am folosit acest union pentru a permite referintei catre v.a
       bidimen sa fie nula
10 setClassUnion("contRV_or_NULL", c("contRV", "NULL"))
```

Pentru a crea un obiect **contRV**, am definit următorul constructor:

```
contRV <- function(densitate, val = function(x) x, bidimen = FALSE,
    suport = list(c(-Inf, Inf)), ref_va_bidimen = NULL)
{
  if(length(suport) < 2)
  suport <- list(suport, list())
  if (bidimen & missing(val))
  val = function(x, y) x * y

  obj <- new("contRV", densitate = densitate, val = val, bidimen =
     bidimen,
  suport = suport, ref_va_bidimen = ref_va_bidimen)

  return (obj)
}
```

De exemplu, pentru crearea unui obiect **contRV** ce reprezintă o variabilă aleatoare continuă bidimensională (X, Y) cu densitatea:

$$f(x,y) = \begin{cases} \frac{6}{7}(x+y)^2, & (x,y) \in [0,1] \times [0,1] \\ \\ 0, & \text{în rest} \end{cases}$$

Scriem:

```
XY <- contRV(densitate = function (x, y) 6/7(x+y)^2, bidimen = TRUE,
    suport = list(list(c(0, 1)), list(c(0, 1))))
```

De asemenea, clasa **contRV** pune la dispoziție utilizatorilor pachetului o colecție de metode pentru a efectua operații pe variabile aleatoare continue, precum: calculul probabilităților (cu ajutorul metodei P, detalii în exercițiul 7), calculul mediilor și a dispersiilor (metodele E și Var, detalii în exercițiile 5-6) și obținerea densităților marginale (metodele $marginalaX$ și $marginalaY$).

5 Cerința 4

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

Oferim un set de funcții, câte două pentru fiecare repartiție (normală, exponențială, beta, gamma), care, pe baza parametrilor corespunzători, afișează densitatea sau funcția-repartiție. De exemplu, pentru a obține graficul densității în repartiția beta de parametri (2, 5), apelăm $den.beta(2, 5)$. Unele funcții conțin și valori implicite (repartiția normală este standard „by default" – la apel fără argumente). Pentru repartiția beta, am adăugat și o funcție care afișează graficul densității animat: o buclă apelează succesiv `plot`, urmat de un foarte scurt timp de așteptare, pentru cursivitatea animației. Graficul mișcător nu poate fi exportat însă; este funcțional în interiorul mediului RStudio. Detalii despre funcțiile densitate și repartiție în exercițiul 8.

Funcțiile sunt definite astfel:

```
# Repartitia normala (implicit standard); m = medie, s = deviatie
    standard
den.normala <- function(m = 0, s = 1) {
  curve(expr = 1 / (s * sqrt(2 * pi)) * exp(1) ^ (-(x - m)^2 / (2 * s
     ^2)),
  from = m - 3 * s,
  to   = m + 3 * s,
  ylab = "densitate",
  main = "Densitatea in repartitia normala")
  abline(v = 0, col = "gray")
}

rep.normala <- function(m = 0, s = 1) {
  curve(expr = pnorm(x, m, s),
  from = m - 3 * s,
  to   = m + 3 * s,
  ylab = "probabilitate",
  main = "Functia repartitie normala")
}

# Repartitia exponentiala

den.exponentiala <- function(l = 1) {
  curve(expr = l * exp(1) ^ (-l * x),
  from = 0,
  to   = 20,
  ylab = "densitate",
  main = "Densitatea in repartitia exponentiala")
}

rep.exponentiala <- function(l = 1) {
```

```
30      curve(expr = 1 - exp(1) ^ (-l * x),
31        from = 0,
32        to   = 10,
33        ylab = "probabilitate",
34        main = "Functia repartitie exponentiala")
35    }
36
37    # Repartitia beta
38
39    den.beta <- function(a, b) {
40      curve(expr = x^(a - 1) * (1 - x) ^ (b - 1) /
41      integrate(function(u) u^(a - 1) * (1 - u)^(b - 1),
42      0, 1, abs.tol = 0)$value,
43      from = 0,
44      to   = 1,
45      ylab = "densitate",
46      main = "Densitatea in repartitia beta")
47    }
48
49    # Teste: den.beta(0.5, 0.5)
50    #        den.beta(2, 2)
51    #        den.beta(2, 5)
52
53    # la fel, dar animata; a - fixat, 0 < b <= bmax
54    den.beta_anim <- function(a = 2, bmax = 8) {
55
56      par(bty = "o")
57
58      for (b in seq(0.1, bmax, length.out = 150)) {
59
60        d <- function(x) x^(a - 1) * (1 - x) ^ (b - 1) /
61        integrate(function(u) u^(a - 1) * (1 - u)^(b - 1),
62        0, 1, abs.tol = 0)$value
63
64        x <- seq(0, 1, length.out = 1000)
65        plot(x,
66        d(x),
67        type = "l",
68        xlab = "x",
69        ylab = "densitate",
70        ylim = c(0, 7),
71        main = "Densitatea in repartitia beta")
72
73        legend(0.7, 6.5, legend = sprintf("b = %.2f", b), cex = 0.8)
74
75        Sys.sleep(0.05)
76      }
77
78    }
79
80    # Test: den.beta_anim()
81    #       den.beta_anim(0.8, 15)
82
83    rep.beta <- function(a, b) {
```

```
84      curve(expr = pbeta(x, shape1 = a, shape2 = b),
85        from = 0,
86        to   = 1,
87        ylab = "probabilitate",
88        main = "Functia repartitie beta")
89    }
90
91    # Teste: rep.beta(5, 1)
92    #        rep.beta(2, 2)
93    #        rep.beta(2, 5)
94
95
96    # Repartitia gamma; k - "shape", t - "scale"
97
98    den.gamma <- function(k, t) {
99      curve(expr = dgamma(x, shape = k, scale = t),
100     from = 0,
101     to   = 20,
102     ylab = "densitate",
103     main = "Densitatea in repartitia gamma")
104   }
105
106   # Teste: den.gamma(1, 2)
107   #        den.gamma(2, 2)
108   #        den.gamma(7.5, 1)
109
110   rep.gamma <- function(k, t) {
111     curve(expr = pgamma(x, shape = k, scale = t),
112     from = 0,
113     to   = 20,
114     ylab = "probabilitate",
115     main = "Functia repartitie gamma")
116   }
117
118   # Teste: rep.gamma(0.5, 1)
119   #        rep.gamma(7.5, 1)
```

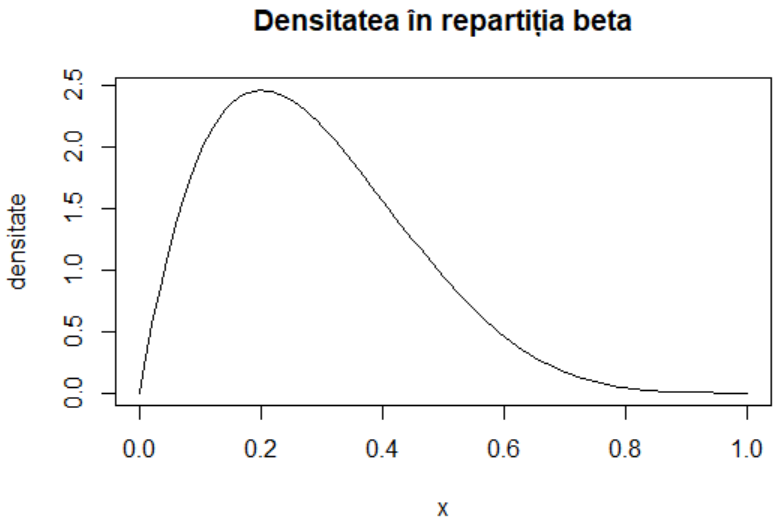

Figura 1: den.beta(2, 5)

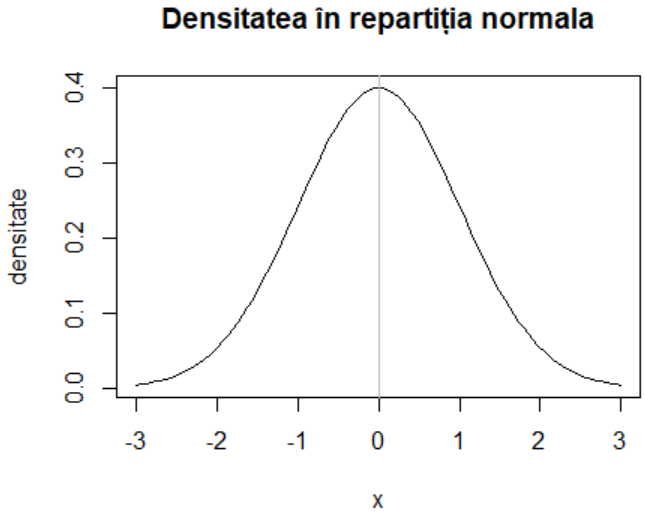

Figura 2: den.normala(0, 1)

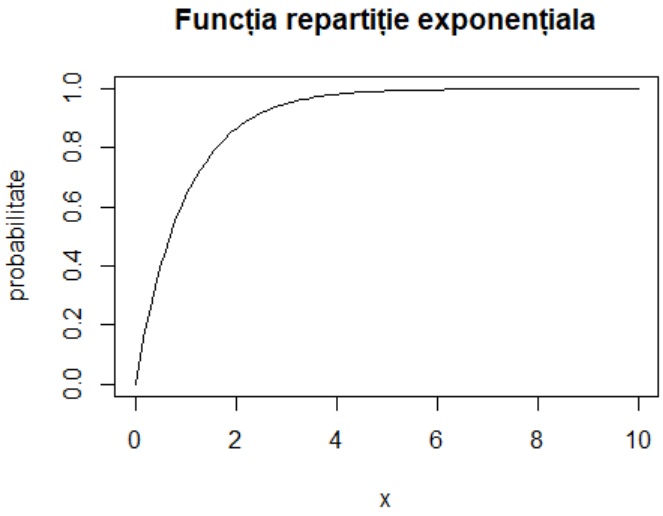

Figura 3: rep.exponentiala(1)

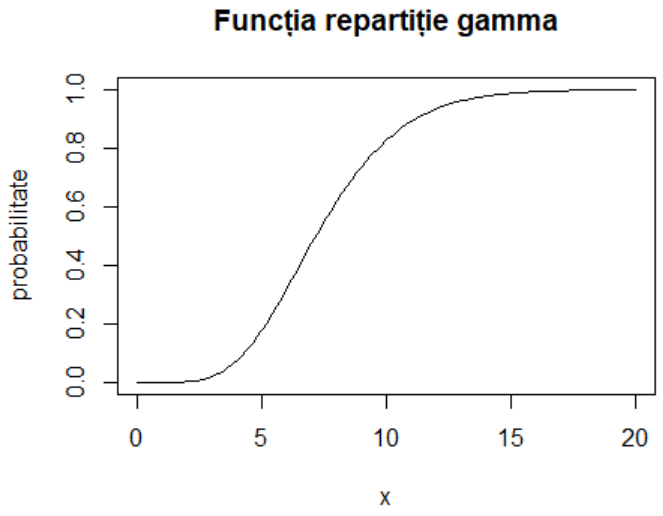

Figura 4: rep.gamma(7.5, 1)

6 Cerința 5 + Cerința 6

5)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.

6)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.

Toate funcțiile legate de aceste două cerințe au fost scrise având cazul general ca scop, ceea ce a a avut ca rezultat că am putut folosi aceleași funcții pentru calcularea mediei lui X, dar și lui $g(X)$ și chiar cuplurilor de forma (X, Y), unde X și Y sunt două variabile aleatoare continue.

6.1 Integrala

Înainte de a începe să prezentăm funcțiile relevante cerințelor, ar fi bine să explicăm ce face mai exact funcția *integrala*, folosită destul de mult de pachetul contRV. Mai întâi, o scurtă descriere a antetului funcției:

Parametrul	Tipul	Descriere
X	contRV	Variabila aleatoare pe care vrem să calculăm o integrală
dt	integer	Variabila după care vrem să integrăm; este folosită explicit doar pentru determinarea densității marginale, altfel implicit are valoarea 0 și se calculeaza doar integrala densității variabilei aleatoare. Pentru dt = 1 se integrează doar după x, iar pentru dt = 2 doar după y.

Scopul parametrului *dt* este doar să faciliteze calcularea densităților marginale, altfel poate fi ignorat.

Funcția se uită mai întâi la ce fel de variabilă este X, unidimensională sau bidimensională. Să luăm mai întâi cazul unidimensionalei, care este cel mai ușor:

```
1  sum <- 0
2  for (i in X@suport[[1]]) {
3    tryCatch(sum <- sum + integrate(Vectorize(X@densitate), i[1], i[2],
         abs.tol = 1.0e-13)$value,
4    error= function(err)
5    {
6      stop("Integrala a esuat.")
7    })
8  }
9  return (sum)
```

După cum se vede, este o simplă sumă pe intervalele din suport. O problemă pe care am întâmpinat-o în lucrul proiectului a fost fapul că pentru densități care sunt nenule pe

intervale mici, funcția *integrate* din R calculează eronat integralele. Astfel, având suportul salvat, putem evita complet această problemă. Valoarea din *abs.tol* este $1.0e-13$ pentru că am observat că, din cauza erorilor de rotunjire ale float-urilor, $abs.tol = 0$ uneori returnează eroare, deși cu $abs.tol = 1.0e-13$ calculează corect integrala. Este posibil sa existe funcții care chiar și cu valoarea curentă să returneze eroare, dar noi n-am întâmpinat astfel de exemple.

Acum să vedem cazul bidimensional, când vrem să calculăm integrala pe densitatea comună a unui cuplu de variabile aleatoare continue:

```
sum <- 0
for(i in X@suport[[2]])
{
  for(j in X@suport[[1]])
  {
    tryCatch(sum <- sum +
    integrate( function(y) {
      sapply(y, function(y) {
        integrate(function(x) X@densitate(x,y),j[1],j[2])$value
      })
    },i[1],i[2])$value,
    error= function(err)
    {
      stop("Eroare la integrarea densitatii.")
    })
  }
}

return(sum)
```

După cum se vede, calculăm integrala pe $S_X \times S_Y$, unde am notat cu S_X suportul lui X din (X, Y) si S_Y suportul lui Y. Motivul pentru care am optat să calculăm pe întreg produsul cartezian este că, în cazul în care X și Y sunt independente, atunci suportul lui (X, Y) va fi chiar $S_X \times S_Y$, fapt ce se vede din formula densității comune $f_{X,Y}(x, y) = f_X(x) \cdot f_Y(y)$. Dacă nu sunt independente, atunci oricum suportul lui (X, Y) va fi sigur inclus în $S_X \times S_Y$, de exemplu, din formula densității marginale $f_X(x) = \int_{-\infty}^{\infty} f_{X,Y}(x, y)\mathrm{d}y$. Atunci, poate vom calcula niște integrale pe intervale în care densitatea comună este 0, dar câștigăm prin faptul că putem folosi același cod pentru două situații diferite, iar timpul pierdut de calcularea unor intervale nule este, în general, mic.

Ultimul caz este cel în care vrem să extragem densitatea marginală. Vom prezenta doar cazul pentru densitatea marginală a lui Y, pentru că la X este identic. Codul:

```
#   Ok, deci ideea care mi a venit aici este sa construiesc un vector
    de functii asa:
#   f_i+1 (y) = (integrala pe [a_i+1,b_i+1]dx) + f_i(y)

funcs <- vector()
factory <- function(i1, i2, pas)
{
```

```
7       i1; i2; pas; # pt closure
8       if(pas == 1)
9       {
10        tmpF <-  Vectorize(function(y) {
11          sapply(y, function(y) {
12            integrate(function(x) X@densitate(x,y),i1,i2)$value
13          })
14        })
15        funcs <<- c(funcs, tmpF)
16      }
17      else
18      {
19        tmpF <-  Vectorize(function(y) {
20          sapply(y, function(y) {
21            integrate(function(x) X@densitate(x,y),i1,i2)$value
22          })
23        })
24
25        newF <- Vectorize(function(y){
26          tmpF(y) + funcs[[pas - 1]](y)
27        })
28        funcs <<- c(funcs, newF)
29      }
30    }
31
32    pas <- 0
33    for(i in X@suport[[1]])
34    {
35      pas <- pas + 1
36      factory(i[1], i[2], pas)
37    }
38
39    return(funcs[[length(funcs)]])
```

Deoarece este posibil ca în formula $f_Y(y) = \int_{-\infty}^{\infty} f_{X,Y}(x,y)\mathrm{d}x$ suportul lui Y să fie spart în intervale, avem deja o problemă la construirea acestei funcții, pentru că va trebui să construim o funcție progresiv, calculând fiecare interval din suport. Soluția pe care am găsit-o este să folosim un vector de funcții, construit dupa următoarele reguli:

- $f_1(y) = \int_{a_1}^{b_1} f_{X,Y}(x,y)\mathrm{d}x$, unde a_1 și b_1 sunt capetele primului interval din suportul lui Y
- $f_{i+1}(y) = \int_{a_{i+1}}^{b_{i+1}} f_{X,Y}(x,y)\mathrm{d}x + f_i(y)$, unde a_{i+1} și b_{i+1} sunt capetele primului celui de-al $i+1$-lea interval din suportul lui Y, iar $f_i(y)$ este functia rezultată din calcularea primelor i integrale

În final, în $f_n(y)$ o să avem densitatea marginală a lui Y, unde n este numărul de intervale din suport.

Încă o dificultate întâlnită a fost la creearea funcțiilor propriu-zise în R. Dacă nu am fi folosit funcția $factory$ și am fi pus direct codul ei în for, am observat că R nu salva corect valoarea lui *pas* sau intervalele din i, ci în schimb punea peste tot ultimul *pas*, respectiv ultimul interval. Din ce am înteles, problema apare din cauza felului în care R implementează scopingul variabilelor. În orice caz, soluția noastră este inspirată din urmatorul răspuns: `https://stackoverflow.com/questions/12481404/how-to-create-a-vector-of-functions`.

Sper că am clarificat destul de bine cum funcționează această funcție, de altfel fundamentală pentru implementarea clasei **contRV**, deoarece am încercat să folosim cât mai des funcția aceasta, în loc de *integrate* singură.

6.2 Media

Implementarea mediei se bazează pe formula ei obișnuită. Pentru aflarea mediei unei variabile continue X, recomandăm folosirea metodei $E(X)$, în loc de apelarea manuală a funcției $media(X)$.

Antetul funcției arată astfel:

Parametrul	Tipul	Descriere
X	contRV	Variabila aleatoare căreia vrem să-i determinăm media

Parametrul poate fi ori unidimensional, ori bidimensional, media este calculată corect în ambele cazuri. Media este implementată astfel:

```
1  media <- function(X)
2  {
3    subIntegrala <- function(...)
4    {
5      X@val(...) * X@densitate(...)
6    }
7
8    subIntegrala <- Vectorize(subIntegrala)
9    tmp <- contRV(densitate = subIntegrala, val = Vectorize(function
       (...) {retval <- 1}), bidimen = X@bidimen, suport = X@suport)
10
11   tryCatch(retval <- integrala(tmp),
12   error=function(err){
13     stop("Media nu exista")
14   }
15   )
16   return(retval)
17 }
```

Codul poate arăta ciudat la prima vedere, probabil din cauza variabilei *tmp*.

În primul rând, după cum se vede, am decis să folosim ... ca parametru pentru implementarea functiei „de sub integrală" a mediei, pentru a putea acoperi și cazul unidimensionalei, și bidimensionalei cu o singură funcție. Motivul pentru care am făcut funcția aceasta este și pentru că am considerat că avem un cod mai elegant dacă nu încărcăm câmpul „densitate" din constructorul lui *tmp* cu o funcție definită pe loc.

În al doilea rând, deoarece deja avem funcția *integrala* care știe să calculeze o integrală pe un anumit suport, ar fi bine să o apelăm cumva, decât să rescriem aceiași secvență de cod de mai multe ori. Desigur, dacă am apela $integrala(X)$, nu am obține rezultatul corect, pentru că vrem să integrăm funcția *subIntegrala*, nu densitatea lui X. Atunci, am găsit soluția să folosim un fel de pseudo-variabilă, căreia îi dăm ca densitate funcția pe care vrem să o integrăm pentru a putea apela *integrala* pe această pseudo-variabilă. Acest artificiu este folosit de mai multe ori în proiect, deoarece am întâmpinat mai multe situații asemănătoare, în care am fi putut folosi o funcție destinată doar variabilelor continue, pentru a calcula diverse valori.

6.3 Dispersia

Dispersia seamănă ca implementare foarte mult cu media, doar că desigur diferă calculele făcute. De asemenea, există o metodă definită pentru dispersie, anume $Var(X)$. Antetul funcției arată astfel:

Parametrul	Tipul	Descriere
X	contRV	Variabila aleatoare căreia vrem să-i determinăm dispersia

```
1  dispersia <- function(X)
2  {
3    # formula clasica cu integrala
4
5    tryCatch(m <- media(X), warning=function(wr)
6    {
7      stop("Calculareaudispersieiuauesuat,umediaunuuexista")
8    })
9
10   xCoef <- function(...)
11   {
12
13     X@val(...) - m
14   }
15
16   coefRaised <- powF(xCoef, 2)
17   coefRaised <- Vectorize(coefRaised)
18
19   subIntegrala <- function(...)
20   {
```

```
21        coefRaised(...) * X@densitate(...)
22      }
23      subIntegrala <- Vectorize(subIntegrala)
24      tmp <- contRV(densitate = subIntegrala, val = Vectorize(function
            (...) {retval <- 1}), bidimen = X@bidimen, suport = X@suport)
25
26      tryCatch(retval <- integrala(tmp),
27      error= function(err)
28      {
29        stop("Dispersia nu exista.")
30      })
31      return(retval)
32    }
```

Cum spuneam, este calculată în mare ca media, folosind o funcție *subIntegrala* și un pseudo-contRV *tmp*. Diferența majoră este introducerea funcției $xCoef$, care reprezintă $(x - E[X])$ din formulă. Voi discuta repede și despre funcția $powF$, care este foarte simplă de altfel.

6.3.1 powF

Scopul acestei funcții este de a lua funcția f din parametru și de a returna o funcție egală cu f ridicată la o anumită putere, dată de asemenea ca parametru. Antetul:

Parametrul	Tipul	Descriere
f	funcție	Funcția pe care vrem să o ridicăm la putere
y	integer	Puterea

Implementarea arată astfel:

```
1   powF <- function(f, y)
2   {
3     ret <- function(...)
4     {
5       f(...) ^ y
6     }
7   }
```

Trebuie ținut cont de modul în care R face scoping, astfel că nu putem scrie ceva de genul:

```
1   func <- powF(func, 3)
2
3   # Body-ul lui func este acum
4   func <- function(...)
5   {
6     f(...) ^ y # atentie!! f este tot func, deci avem o recursie
          infinita
7   }
```

Se poate evita ușor problema aceasta folosind o funcție $funcRaised$ în care să stocăm $powF(func, 3)$. Asta este și soluția folosită de noi în proiect.

Revenind la dispersie, după ce ridicăm $xCoef$ la puterea a 2-a, codul devine aproape identic ca la medie.

6.4 Momentul centrat de ordin r

Momentul centrat este implementat într-o măsură asemănătoare. Să vedem mai întâi antetul:

Parametrul	Tipul	Descriere
X	contRV	Variabila aleatoare pe care vrem să calculăm momentul centrat
ordin	integer	Ordinul

Implementarea este aproape identică cu dispersia, doar că acum ridicăm $xCoef$ la r, în loc de 2. O modificare pe care am făcut-o a fost că, dacă $r < 3$, putem calcula momentul centrat în funcție de funcțiile deja scris în pachet în modul următor:

```
1  moment_centrat <- function(X, ordin)
2  {
3    # cazurile triviale
4    if(ordin == 0)
5    return(1)
6    else if(ordin == 1)
7    tryCatch({ # trebuie totusi verificat daca E(X) exista, ca altfel
         nu va da nici macar 0
8      media(X)
9      return(0)
10   }, error= function(err)
11   {
12     stop(paste("Calculareaumomentuluiucentratudeuordinu", ordin, "uau
         esuat,unuuexistaumedia."))
13   })
14   else if(ordin == 2)
15   tryCatch({  # X dispersia nu exista, vrem un mesaj specific pt
         momente
16     return(dispersia(X))
17   }, error= function(err)
18   {
19     stop(paste("Calculareaumomentuluiucentratudeuordinu", ordin, "uau
         esuat,unuuexistaudispersie."))
20   })
21
22   tryCatch(m <- media(X), warning=function(wr)
23   {
```

```
24        stop(paste("Calcularea momentului centrat de ordin ", ordin, " a
              esuat"))
25      })
26
27      xCoef <- function(...)
28      {
29
30        X@val(...) - m
31      }
32
33      coefRaised <- powF(xCoef, ordin)
34      coefRaised <- Vectorize(coefRaised)
35
36      subIntegrala <- function(...)
37      {
38        coefRaised(...) * X@densitate(...)
39      }
40      subIntegrala <- Vectorize(subIntegrala)
41      tmp <- contRV(densitate = subIntegrala, val = Vectorize(function
            (...) {retval <- 1}), bidimen = X@bidimen, suport = X@suport)
42
43      tryCatch(retval <- integrala(tmp),
44      error= function(err)
45      {
46        stop(paste("Momentul centrat de ordin ", ordin, " nu exista."))
47      })
48      return(retval)
49    }
```

În cazurile "triviale", adică cele în care ne putem folosi de funcții deja scrise, am optat să le apelăm pentru a obține un comportament al funcțiilor cât mai apropiat de cel așteptat. De asemenea, momentele centrate și inițiale se apelează direct cu funcțiile lor, deoarece nu am avut vreo idee de cum să le definim un generic mai convenabil.

6.5 Momente inițiale de ordin r

Codul este aproape identic cu cel al momentului centrat, doar ca $xCoef$ va fi doar $X@val(...)$. Antetul:

Parametrul	Tipul	Descriere
X	contRV	Variabila aleatoare pe care vrem să calculăm momentul inițial
ordin	integer	Ordinul

```
1   moment_initial <- function(X, ordin)
2   {
3     # cazurile triviale
4     if(ordin == 0)
```

```
5       return(1)
6       else if(ordin == 1)
7       tryCatch({ # trebuie totusi verificat daca E(X) exista, ca altfel
            nu va da nici macar 0
8         return(media(X))
9       }, error= function(err)
10      {
11        stop(paste("Calcularea momentului initial de ordin ", ordin, " a
              esuat."))
12      })
13
14
15      xCoef <- function(...)
16      {
17
18        X@val(...)
19      }
20
21      coefRaised <- powF(xCoef, ordin)
22      coefRaised <- Vectorize(coefRaised)
23
24      subIntegrala <- function(...)
25      {
26        coefRaised(...) * X@densitate(...)
27      }
28      subIntegrala <- Vectorize(subIntegrala)
29      tmp <- contRV(densitate = subIntegrala, val = Vectorize(function
            (...) {retval <- 1}), bidimen = X@bidimen, suport = X@suport)
30
31      tryCatch(retval <- integrala(tmp),
32      error= function(err)
33      {
34        stop(paste("Momentul initial de ordin ", ordin, " nu exista."))
35      })
36      return(retval)
37    }
```

6.6 Implementarea pentru exercițiul 6

Pentru a asigura faptul că funcțiile pot fi folosite și în situații de genul $g(X)$, am stocat repartiția lui X în câmpul *val*. De asemenea, am definit genericul *aplica* care, după cum îi spune și numele, ia o funcție g și o variabilă X și returnează o variabilă X', căreia i-am aplicat asupra repartiției g. Antetul lui *aplica* este:

Parametrul	Tipul	Descriere
X	contRV	Variabila aleatoare
g	funcție	Funcția continuă pe care vrem să o aplicăm

Deci, dacă vrem să aflăm, de exemplu, $E[g(X)]$, putem scrie $E(aplica(X, g))$ și, deoarece toate funcțiile de mai sus apeleaza câmpul *val*, sigur va funcționa. Dacă g returnează efectiv

un obiect contRV, putem apela direct $E(g(X))$.

Codul lui *aplica* este:

```
setMethod("aplica", "contRV",
function(object, f){
  retval <- contRV(object@densitate, Vectorize(compunere(f,
     object@val)), object@bidimen, object@suport,
  ref_va_bidimen = object@ref_va_bidimen)
})
```

Iar compunere este definit:

```
compunere <- function(f, g)
{
  function(...) f(g(...))
}
```

7 Cerința 7

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

Pentru a calcula probabilități pe variabile aleatoare continue, am definit funcția P ca o metodă a clasei **contRV**, astfel:

```
setMethod("P", "contRV",
function (object) {
  return (integrala(object)) # integreaza pe suport
})
```

După cum se poate observa, parametrul funcției este un obiect de tip **contRV**. Acest lucru poate părea ciudat la prima vedere; ar fi inutil să putem calcula doar probabilități de tipul $P(X)$, intrucât rezultatul ar fi întodeauna egal cu 1. În contextul pachetului, însă, un obiect de tip **contRV** nu reprezintă întotdeauna o variabilă aleatoare propriu-zisă. Mai precis, orice obiect poate fi rezultatul unei expresii de tipul: $X \leq x$, $(X \leq x) \cap (Y \geq y)$, $(X < a) \cup (X > b)$ etc. Astfel, prin evaluarea expresiilor, restrângem domeniul pe care integrăm densitățile în calculul probabilităților și să păstrăm notații cât mai apropiate de cele matematice(detalii în exemplele de la sfârșit).

Pentru a evalua expresiile, am supraîncărcat următorii operatori:

```
1  setMethod("<", c("contRV", "numeric"), function (e1, e2) {
2    comp(e1, e2, "<=") # P(X < x) = P(X <= x)
3  })
4  setMethod("<=", c("contRV", "numeric"), function (e1, e2) {
5    comp(e1, e2, "<=")
6  })
7  setMethod(">", c("contRV", "numeric"), function (e1, e2) {
8    comp(e1, e2, ">=") # P(X > x) = P(X >= x)
9  })
10 setMethod(">=", c("contRV", "numeric"), function (e1, e2) {
11   comp(e1, e2, ">=")
12 })
13 setMethod("==", c("contRV", "numeric"), function (e1, e2) {
14   comp(e1, e2, "==")
15 })
16 setMethod("%AND%", c("contRV", "contRV"), function (e1, e2) {
17 op(e1, e2, "&") # intersectie
18 })
19 setMethod("%OR%", c("contRV", "contRV"), function (e1, e2) {
20 op(e1, e2, "|") # reuniune
21 })
22 setMethod("|", c("contRV", "contRV"), function (e1, e2) {
23 cond(e1, e2) # conditionare
24 })
```

Pentru operatorii de inegalitate și egalitate, se observă ca aceștia au drept parametri un obiect **contRV** și un număr real. Se apelează funcția *comp*, ce va efectua restrângerea efectivă a suportului variabilei aleatoare pentru calculul probabilităților.

```
1  # Determina suportul pentru expresii de tipul X <= x, X >= x
2  comp <- function(X, x, c)
3  {
4
5  if (X@bidimen)
6  stop("Nu se poate compara o v.a bidimensionala cu un numar real!")
7
8  suportNou <- list()
9  nr <- 1
10
11 # Presupunem ca intervalele sunt in ordine crescatoare dupa capatul
       inferior
12 # si nu se intersecteaza!
13 if (c == "==")
14 {
15   for (i in X@suport[[1]])
16   {
17     a <- i[1]
18     b <- i[2]
19
20     if (x < a)
21     break # nu mai are rost sa cautam
22
23     if (a <= x & b >= x) # daca x se afla in intervalul [a, b]
24     {
25       suportNou[[nr]] <- c(x, x) # suportul va fi intervalul [x, x]
26       break
27     }
28   }
29 }
30 else if (c == "<=")
31 {
32   # Exemplu: Daca suportul densitatii este format din [0, 2] U [4, 7] U
         [9, 11]
33   # Noul suport pentru X <= 5 va fi [0, 2] U [4, 5]
34
35   for (i in X@suport[[1]])
36   {
37     a <- i[1]
38     b <- i[2]
39
40     if (x < a)  # daca x este mai mic decat capatul inferior al
          intervalului
41     break # am terminat de construit suportul
42
43     if (a <= x & b >= x) # daca x se afla in intervalul [a, b]
44     {
45       suportNou[[nr]] <- c(a, x) # adaugam ultimul interval din noul
            suport, adica [a, x]
46       break
```

```
47      }
48
49      # altfel, n-am ajuns la un interval care sa-l contina pe x, deci il
            adaugam in suportul nou
50      suportNou[[nr]] <- c(a, b)
51      nr <- nr + 1
52    }
53  }
54  else # ">="
55  {
56    # Exemplu: Daca suportul densitatii este format din [0, 2] U [4, 7] U
          [9, 11]
57    # Noul suport pentru X >= 5 va fi [5, 7] U [9, 11]
58
59    # parcurgem intervalele in ordine descrescatoare dupa capatul
          inferior
60    for (i in rev(X@suport[[1]]))
61    {
62      a <- i[1]
63      b <- i[2]
64
65      if (x > b)
66      break
67
68      if (a <= x & b >= x) # daca x se afla in intervalul [a, b]
69      {
70        suportNou[[nr]] <- c(x, b) # adaugam ultimul interval din noul
              suport, adica [x, b]
71        break
72      }
73
74      # altfel, inca n-am ajuns la un interval care sa-l contina pe x,
            deci il adaugam in suportul nou
75      suportNou[[nr]] <- c(a, b)
76      nr <- nr + 1
77    }
78
79    # inversam ordinea din noul suport, intrucat am parcurs intervalele
          din suport in ordine inversa
80    suportNou <- rev(suportNou)
81  }
82
83  # atentie! rezultatul obtinut nu mai este o v.a! se foloseste doar pt a
        calcula probabilitati!
84  return (contRV(densitate = X@densitate, val = X@val, bidimen =
        X@bidimen, suport = suportNou,
85  ref_va_bidimen = X@ref_va_bidimen))
86  }
```

Pentru operatorii %AND% și %OR% ce implementează intersecția, respectiv reuniunea de variabile aleatoare continue, se apelează funcția *op*, care va avea un comportament diferit în funcție de relația dintre cei doi parametri X și Y de tip **contRV**. Mai exact, distingem următoarele cazuri:

- X și Y au aceeași densitate de probabilitate(apare de obicei în urma unor expresii de genul: $(X < a) \cap (X > b)$), caz în care formăm un obiect de tip **contRV** cu densitatea cunoscută și drept suport intersecția dintre suporturile lui X și Y.
- X și Y au o referință către aceeași v.a bidimensională, așa că formăm un nou obiect nou obiect **contRV** cu densitatea comună deja cunoscută și drept suport produsul cartezian intre suportul lui X și cel al lui Y
- X și Y au câte o referință către v.a bidimensionale diferite(sau nu au nicio referință), caz în care le considerăm independente și formăm un nou obiect **contRV** cu densitatea comună $f(x, y) = f_X(x) \cdot f_Y(y)$ și drept suport produsul cartezian intre suportul lui X și cel al lui Y

Funcția *op* este definită astfel:

```
1  # Reuniune si intersectie de variabile aleatoare
2  op <- function(X, Y, o)
3  {
4  suportNou <- list()
5  nr <- 1
6
7  if (o == "&") # intersectie
8  {
9
10   if (!identical(X@densitate, Y@densitate))
11   {
12
13     XY <- NULL
14
15     if (!is.null(X@ref_va_bidimen) & identical(X@ref_va_bidimen, Y@ref_
         va_bidimen))
16     {
17       # aici se face o copie
18       XY <- X@ref_va_bidimen
19     }
20     else
21     {
22       # consideram ca sunt independente
23       XY <- contRV(densitate = function(x, y) {X@densitate(x) *
           Y@densitate(y)}, bidimen = TRUE)
24     }
25
26     XY@suport[[1]] <- X@suport[[1]]
27     XY@suport[[2]] <- Y@suport[[1]]
28
29     return (XY)
30   }
```

```
31
32    for (i in X@suport[[1]])
33    {
34      for (j in Y@suport[[1]])
35      {
36        # reuniuni de intersectii ale intervalelor din suport
37
38        A <- interval_intersect(i, j)
39        if (!is.null(A))
40        {
41          suportNou[[nr]] <- A
42          nr <- nr + 1
43        }
44      }
45    }
46
47    return (contRV(densitate = X@densitate, val = X@val, bidimen =
          X@bidimen, suport = suportNou,
48    ref_va_bidimen = X@ref_va_bidimen)) # intoarce contRV pt a integra
          suportul ramas
49  }
50  else # reuniune
51  {
52    return (P(X) + P(Y) - P(X %AND% Y)) # aici intoarce deja
          probabilitatea calculata
53    # problema este ca nu se mai pot aplica alte operatii pe v.a
54  }
55  }
```

O limitare a acestei funcții ar fi că în urma unei operații de reuniune, nu se întoarce un obiect **contRV** pentru a fi folosit în alte expresii, ci direct rezultatul probabilității, adică $P(X \in A) + P(Y \in B) - P(X \in A, Y \in B)$.

Ultimul operator implementat este cel de condiționare:

```
setMethod("|", c("contRV", "contRV"), function (e1, e2) {
cond(e1, e2)
})
```

pentru care se apelează funcția *cond*, care are la rândul ei un comportament diferit în funcție de relația dintre cele 2 obiecte X și Y de tipul **contRV**. Dacă:

- X și Y au densitățile marginale dintr-o v.a bidimensională (X, Y), atunci calculează direct probabilitatea folosind formula: $P(X \in A \mid Y = y) = \int_A f_{X|Y}(x|y)\,dx$, unde densitatea condiționată o obținem din funcția `dens_condit_x_de_y`, descrisă în exercițiul 11.
- Altfel, avem ori X și Y independente, ori au aceeași densitate. În ambele cazuri, putem folosi formula: $P(X \in A \mid Y \in B) = \dfrac{P(X \in A, Y \in B)}{P(Y \in B)}$. Chiar și pentru X, Y independente, rezultatul ar fi cel așteptat, adică $P(X \in A)$.

Funcția cond arată în felul următor:

```
# Calculeaza probabilitatea conditionata
# X si Y pot fi expresii de tipul Z <= x, Z %AND% W etc.
cond <- function (X, Y)
{
if (!is.null(X@ref_va_bidimen) & identical(X@ref_va_bidimen, Y@ref_va_
    bidimen))
{
  # aici probabil ar trebui verificat daca X si Y referintiaza aceeasi
     v.a bidimensionala
  XY <- X@ref_va_bidimen
  fx_cond_y <- dens_condit_x_de_y(XY)

  if (length(Y@suport[[1]]) == 0) # inseamna ca nu a fost gasit y in
     suportul lui Y
  {
    return (0)
  }

  if (length(Y@suport[[1]]) != 1  || Y@suport[[1]][[1]][[1]] !=
     Y@suport[[1]][[1]][[2]])
  {
    stop("Nu pot calcula asa ceva! Suportul lui Y trebuie sa fie un
       singur punct!!")
  }


  yfixat <- Y@suport[[1]][[1]][[1]]

  sum <- 0
  for (i in X@suport[[1]]) {
    tryCatch(sum <- sum + integrate(fx_cond_y, i[1], i[2], y = yfixat,
       abs.tol = 1.0e-13)$value,
    error= function(err)
    {
      stop("Integrala a esuat.")
    })
  }
  return (sum)

}
else
{
  return (integrala(X %AND% Y) / integrala(Y)) # P(X intersect Y) / P(Y
     )
}
}
```

Comparație între notații:

contRV	Notație matematică
`P(X <= x)`	$P(X \leq x)$
`P((X <= a) %AND% (Y >= b))`	$P(X \leq a, Y \geq b)$
`P((X <= a) %OR% (X >= b))`	$P(X \leq a \cup X \geq b)$
`P((X >= a) %AND% (X <= b) | X < c)`	$P(a \leq X \leq b \mid X < c)$
`P(X > x | Y == y)`	$P(X > x \mid Y = y)$

Exemple de cod:

```
> XY <- contRV(densitate = function (x, y) (6/7) * (x+y)^2,
bidimen = TRUE,
suport = list(list(c(0, 1)), list(c(0, 1))))
> X <- marginalaX(XY)
> Y <- marginalaY(XY)
> P((X <= 0.5) %AND% (X >= 0.2) | Y == 0.2)
[1] 0.1622093
> P((X <= 0.7) %AND% (Y >= 0.5))
[1] 0.3815


> func <- function(x)
+ {
if (x < -1)
0
else if (x < 0)
1 + x
else if (x < 1)
1 - x
else
0
}
> Z <- contRV(densitate = Vectorize(func), bidimen = FALSE, suport =
   list(c(-1, 1)))
> P(Z <= 0)
[1] 0.5
> P(((Z <= 0.5) %AND% (Z >= -0.7)) %OR% (Z <= 1))
[1] 1
```

8 Cerința 8

8.) 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 in mod uzual acea repartiție, semnificația parametrilor, media, dispersia etc.

Scurtă descriere al antetului funcției:

Parametrul	Tipul	Descriere
Rep	Unul din vectorii definiți în pachet	Repartiția dată ca parametru

```
1  #Functia afiseaza informatiile despre o repartitie mai compact
2  #compact = fara spatii goale
3  #De asemenea reprezinta o metoda mai intuitiva de a afisa
4  Fisa_sinteza <- function(Rep){
5    for (i in Rep)
6    {
7      print(i)
8    }
9
10 }
```

Funcția Fisa_sinteza afișează informațiile salvate despre repartiția dată ca parametru sub forma unei fișe de sinteză.

Informațiile sunt salvate într-o listă de elemente de tip String. Avem următoarele liste ce pot fi afișate:

- rep_uniforma
- rep_exponentiala
- rep_normala
- rep_gamma
- rep_beta
- rep_X2 (X stă în loc de χ, pentru a nu avea erori de encoding)
- rep_student

9 Cerința 10

10) Calculul covarianței și coeficientului de corelație pentru două variabile aleatoare continue(Atenție: Trebuie să folosiți densitatea comună a celor două variabile aleatoare!)

9.1 Covarianță

Parametrul	Tipul	Descriere
Z	contRV	Variabila aleatoare continuă bidimensională $Z = (X, Y)$

Funcția `Cov` calculează covarianța pe baza formulei:

$$Cov(X, Y) = E[XY] - E[X] \cdot E[Y]$$

```
1  Cov <- function(Z) {
2
3    if (!Z@bidimen) {
4      print("Variabila nu este bidimensionala!")
5      NA
6    }
7    else {
8
9      X <- marginalaX(Z)
10     Y <- marginalaY(Z)
11
12     return (E(Z) - E(X) * E(Y))
13
14   }
15 }
```

9.2 Coeficientul de corelație

Parametrul	Tipul	Descriere
Z	contRV	Variabila aleatoare continuă bidimensională $Z = (X, Y)$

Funcția `Cor` calculează coeficientul de corelație pe baza formulei:

$$\rho(X, Y) = \frac{Cov(X, Y)}{\sqrt{Var(X) \cdot Var(Y))}}$$

```
Cor <- function(Z)
{
  if (!Z@bidimen) {
    print("Variabila nu este bidimensionala!")
    NA
  }
  else {

    X <- marginalaX(Z)
    Y <- marginalaY(Z)

    return (Cov(Z) / sqrt(Var(X) * Var(Y)))

  }
}
```

Exemple:

```
> Z <- contRV(densitate = function (x, y) (6/7) * (x+y)^2,
bidimen = TRUE,
suport = list(list(c(0, 1)), list(c(0, 1))))
> Cov(Z)
[1] -0.008503401
> Cor(Z)
[1] -0.1256281
```

10 Cerința 11

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

Implementarea densităților marginale și condiționate se fac în funcție de funcția *integrala*. Astfel, avem densitățiile marginale:

```
# construieste o v.a continua pornind de la densitatea marginala a
    lui X in v.a bidimen (X, Y)
marginalaX <- function(XY)
{
  return (contRV(densitate = integrala(XY, 2), suport = XY@suport[[1
     ]], bidimen = FALSE, ref_va_bidimen = XY))
}

# construieste o v.a continua pornind de la densitatea marginala a
    lui Y in v.a bidimen (X, Y)
marginalaY <- function(XY)
{
  return (contRV(densitate = integrala(XY, 1), suport = XY@suport[[2
     ]], bidimen = FALSE, ref_va_bidimen = XY))
}
```

Câmpul $ref\_va\_bidimen$ este folosit pentru a putea accesa cuplul (X, Y) din X sau Y când vrem să calculăm vreo probabilitate care le include pe ambele.

Densitățiile condiționate au fost implementate în mod asemănător:

```
dens_condit_x_de_y <- function(Z)
{
  dens_marginala_y <- integrala(Z,1)
  f <- function(x,y) {if (dens_marginala_y(y) != 0) {Z@densitate(x,y)
      / dens_marginala_y(y)} else stop("Densitatea marginala este 0")
     }
  return (Vectorize(f))
}

dens_condit_y_de_x <- function(Z)
{
  dens_marginala_x <- integrala(Z,2)
  f <- function(x,y) {if (dens_marginala_y(y) != 0) {Z@densitate(x,y)
      / dens_marginala_x(x)} else stop("Densitatea marginala este 0")
     }
  return (Vectorize(f))
}
```

11 Concluzii

Pachetul **contRV** oferă posibilitatea de a efectua diverse calcule probabilistice cu variabile aleatoare continue, folosind o sintaxă apropiată de cea matematică. Acoperă un set mare de cazuri care pot apărea în lucru cu probabilități, iar rezultatele sunt precise, având o eroare maximă de $1.0e-13$.

11.1 Surse de inspirație

- https://stackoverflow.com/questions/12481404/how-to-create-a-vector-of-functions
- https://online.stat.psu.edu/stat414/lesson/20/20.1
- https://en.wikipedia.org/wiki/Joint_probability_distribution
- https://stat.ethz.ch/pipermail/r-help/2013-February/347686.html
- http://cs.unitbv.ro/~pascu/stat/Distributii%20continue%20clasice.pdf%7D
- http://math.etc.tuiasi.ro/rstrugariu/cursuri/SPD2015/c7.pdf%7D
- http://images.wikia.com/nccmn/ro/images/3/37/Capitolul_10_REPARTITII_CLASICE.pdf%7D
- https://www.probabilitycourse.com/chapter5/5_2_3_conditioning_independence.php
- https://ro.qaz.wiki/wiki/Normalizing_constant
- https://rdrr.io/cran/discreteRV/f/
- https://rdrr.io/rforge/distr/