Proiect

Construirea unui pachet R pentru lucru cu variabile aleatoare continue

Buruiană Alexandra

Cojocaru Alexandra-Flavia

Florea Maria

Gavrilă Andreea

An II, Grupa 244

I. Introducere

Variabilele aleatoare descriu un concept referitor la studiul matematic al fenomenelor aleatoare.

O funcție $X : E \to R$ se numeste variabilă aleatoare dacă, pentru orice $x \in R$, are loc relația $\{X \leq x\} = \{e \in E \mid X(e) \leq x\} \in K$.

Spunem că variabila aleatoare X este de tip continuu dacă funcția de repartiție $F : R \to [0,1]$ este continuă și există o funcție $f : R \to R$ cu o mulțime cel mult numărabilă de puncte de discontinuitate de primă speță astfel încât $F(x) = \int_{-\infty}^{x} f(t)dt$ pentru orice $x \in R$.

Generalizând aceste definiții, putem să spunem că posibilul rezultat al unui experiment aleator este asociat unei valori reale, fiind precizată regula de asociere. Această regulă de asociere devine astfel valoarea aleatoare. În viața de zi cu zi, acest tip de funcții se întâlnește mai des decât ne-am aștepta, printre exemple se numără extragerea loto, timpul pe care un om îl petrece așteptând într-o stație de autobuz, măsurarea fertilității solului în locații diferite sau numărul clienților serviți la un ghișeu.

Ținând cont de repetiția acestor situații în viața reală, putem afirma că lucrul cu variabile aleatoare, în special continue, este esențială în studiul probabilităților și statisticii.

II. Descrierea Problemei și Soluții

Cerința 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.

Descrierea soluției:

Funcția are ca parametrii o funcție ce va fi intrudusă de utilizator. Vom verifica dacă se poate determina constanta de normalizare k pentru funcția introdusă.

Ne folosim de următoarea formulă pentru constanta de normalizare:

$$k = \frac{1}{\int_{-\infty}^{\infty} f(x)dx}$$

Verificarea pe care o vom face este inclusă într-un block try-catch, în care vedem dacă se poate calcula integrala din formulă – condiție de existență a constantei de normalizare. În cazul în care aceasta nu merge calculată vom afișa un mesaj către utilizator. Dacă este posibilă calcularea integralei, se va returna inversul rezultatului.

Implementare:

```
#' @export
constantaNorm <- function(f)
{
  tryCatch({
    result<-integrate(Vectorize(f), lower = -Inf, upper= Inf)$value
    return(1 / result)
  },
  error = function(err){
    message(paste("Nu se poate calcula constanta de normalizare ->
eroare:", err))
  })
}
```

Teste:

1.constantaNorm(function(x) {exp(-(x^2)/2)})

```
> constantaNorm(function(x) {exp(-(x^2)/2)})
[1] 0.3989423
> |
```

2. constantaNorm(function(x) {exp(x)/((1+exp(x))^2)})

```
> constantaNorm(function(x) {exp(x)/((1+exp(x))^2)})
Nu se poate calcula constanta de normalizare -> eroare: Error in integrate(Vectorize(f), lower = -Inf, upp
er = Inf): non-finite function value

> |
```

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

Descrierea soluției:

Funcția are ca parametri o funcție f care reprezintă densitatea de probabilitate și suportul ei (o listă de vectori cu câte două elemente). Vom verifica cele două proprietăți pe care o funcție trebuie să le îndeplinească pentru a fi densitate de probabilitate:

1. $f(x) \geq 0$, oricare ar fi x din suport
2. $\int_{-\infty}^{\infty} f(x)dx = 1$

Inițial, verificăm prima condiție: pentru fiecare interval din suport, salvăm valorile în 2 variabile, lim_inf și lim_sup și verificăm dacă una este –infinit sau infinit, respectiv dacă amândouă sunt. În caz afirmativ, le modificăm valorile astfel încât să nu obținem eroare când apelăm seq și ne luăm un interval de 10000 de elemente în cazul în care doar un capăt al intervalului nu este finit sau luăm un interval mare pentru a căuta valori negative. Memorăm secvența generată de seq, secvență de lungime de 10000 de elemene. Funcția sapply va returna un vector cu

elementele din vector_auxiliar peste care a fost aplicată funcția f, adică fiecare element x din vector_auxiliar va deveni f(x). În final, verificăm dacă există cel puțin o valoare negativă în noul vector, caz în care afișăm intervalul respectiv și returnăm FALSE.

Verificăm apoi cea de-a doua condiție într-un try în care integrăm funcția primită ca argument pe intervalul curent și rezultatul îl adunăm în valoare_integrala care, la final, trebuie să fie 1 dacă f este densitate de probabilitate. În cazul în care apare eroare la integrare, aceasta va fi prinsă în catch, unde afisăm un mesaj corespunzător și returnăm FALSE.

În final, verificăm dacă suma integralelor este 1.

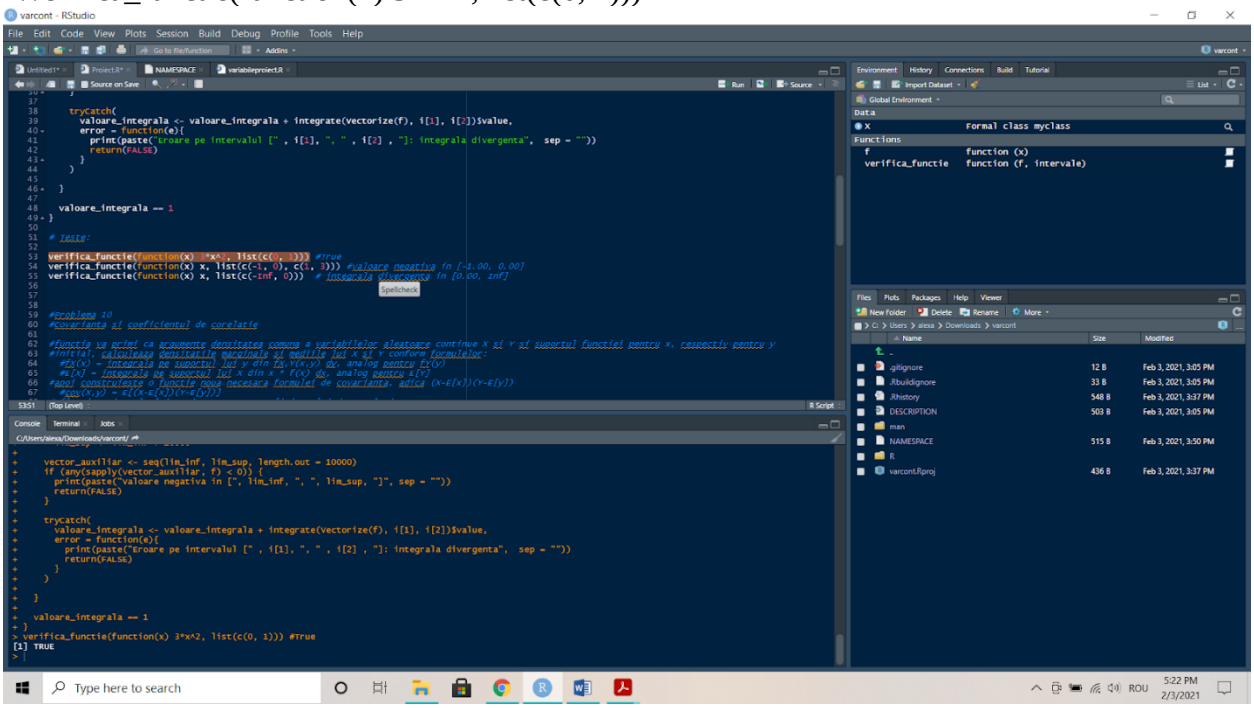
Implementare:

```
#' @export
verifica_functie <- function(f, intervale) {
  valoare_integrala <- 0 #pentru suma integralelor care va fi sau nu 1
  for (i in intervale)
    lim_inf <- i[1]  #copiem valorile capetelor intervalului
    lim_sup <- i[2]
    if (lim_inf == -Inf && lim_sup == Inf) {
      #va aparea eroare la apelarea lui seq, deci luam un interval mai
mare de numere pentru care verificam proprietatea
      lim_inf <- -10000
      lim_sup <-  10000
    }
    else if (lim_inf == -Inf)
      lim_inf <- lim_sup - 10000
    else if (lim_sup ==  Inf)
      lim_sup <- lim_inf + 10000
    vector_auxiliar <- seq(lim_inf, lim_sup, length.out = 10000)
#seq va genera secventa de numere din intervalul delimitat de lim_inf
si lim_sup, secventa care va avea lungimea de 10000
    if (any(sapply(vector_auxiliar, f) < 0)) {
#sapply va returna un vector cu valorile din vector_auxiliar peste
care a fost aplicata functia f, deci fiecare x va deveni f(x)
```

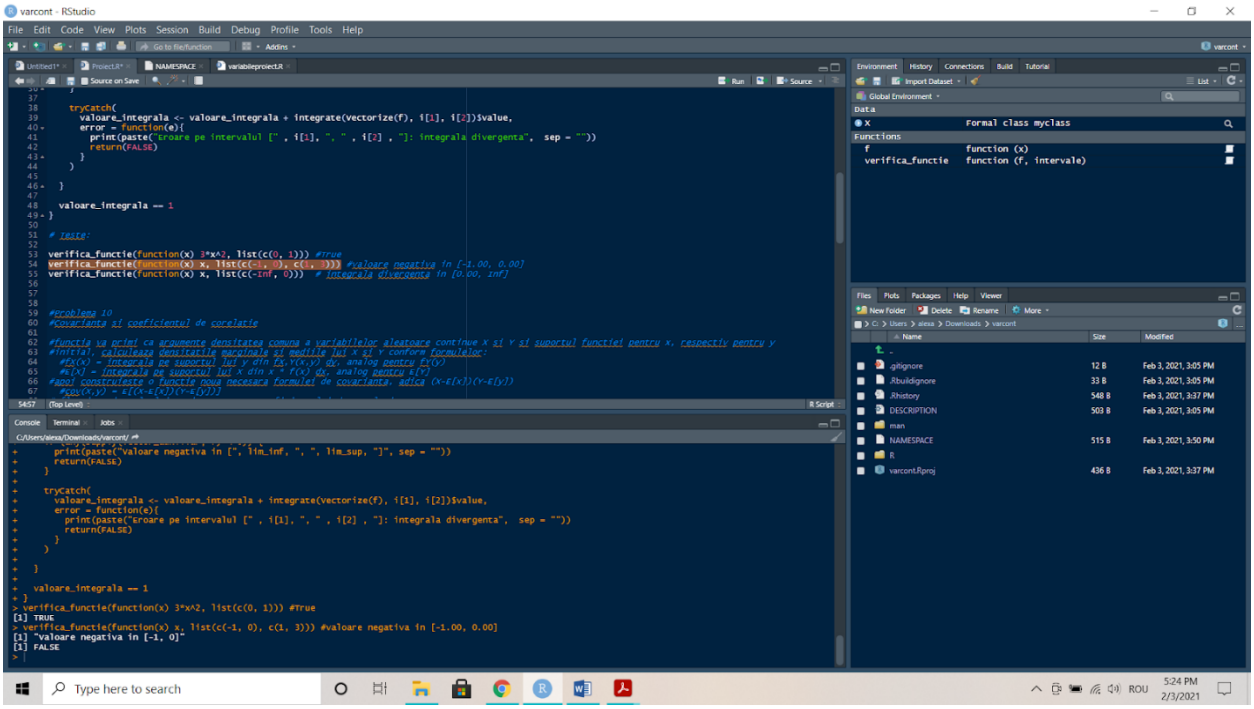
```
      print(paste("Valoare negativa in [", lim_inf, ", ", lim_sup,
"]", sep = ""))

      return(FALSE)

    }

    tryCatch(

#integram pe intervalul curent si adunam la suma, daca apare eroare la
integrare, va fi prinsa in catch si se va afisa un mesaj corespunzator

      valoare_integrala <- valoare_integrala + integrate(Vectorize(f),
i[1], i[2])$value,

      error = function(e){

        print(paste("Eroare pe intervalul [" , i[1], ", " , i[2] , "]:
integrala divergenta",  sep = ""))

        return(FALSE)

      }

    )

  }

  valoare_integrala == 1 #verificarea conditiei necesare pentru ca f
sa fie densitate de probabilitate

}
```

Teste:

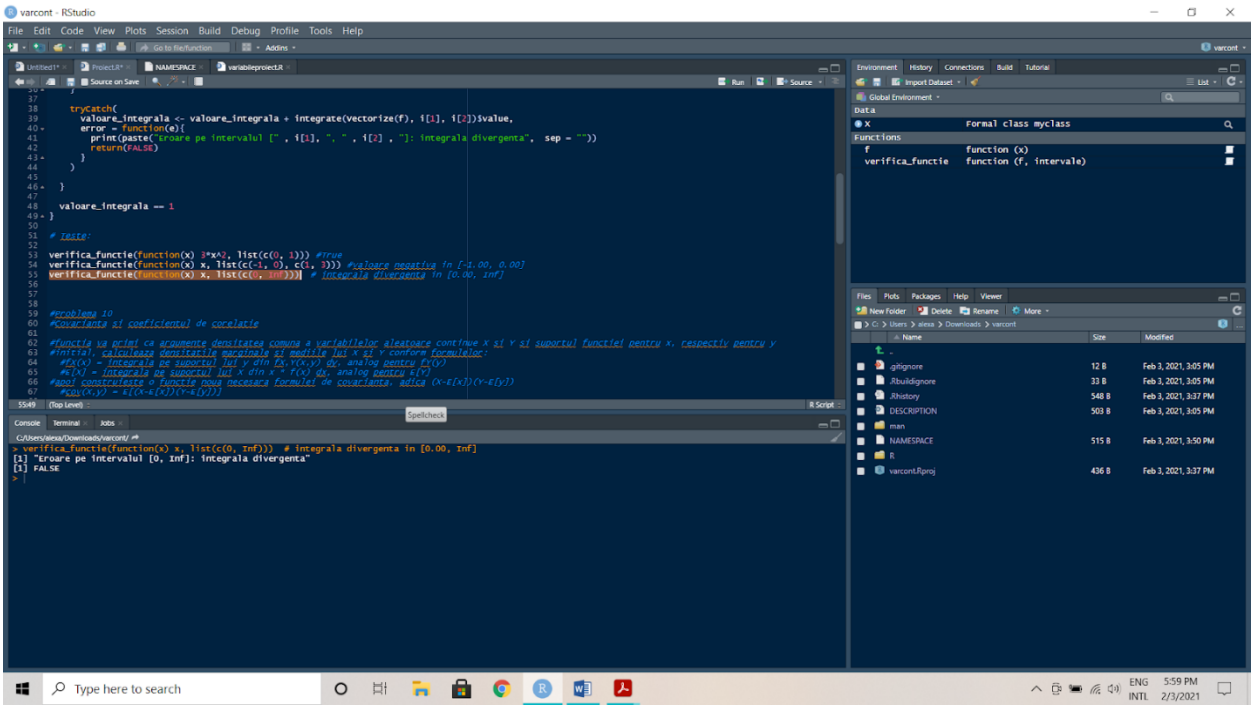
1.verifica_functie(function(x) 3*x^2, list(c(0, 1)))



2.verifica_functie(function(x) x, list(c(-1, 0), c(1, 3)))



3.verifica_functie(function(x) x, list(c(0, Inf)))



Cerința 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.

Descrierea soluției:

Definim clasa *myclass* în care am inclus lista de câmpuri cu următoarele semnificații: în *density* se adaugă densitatea de probabilitate introdusă de utilizator, în *lower*, voi avea capătul inferior din integrală, *bidimen* reprezintă câmpul de tip logic (True/False) dacă opțiunea selectată este pentru variabile aleatoare bidimensionale și respectiv unidimensionale, și *base* va fi o lista de intervale. Când avem variabile aleatoare bidimensionale, baza va fi formată din listele bazei densitatii lui X si Y, iar când avem variabile aleatoare unidimensionale, baza va fi doar lista intervalelor enumerate. În continuare vom verifica lungimea listei din baza folosită, urmând să verificăm și să completăm capătul inferior pentru variabilele aleatoare unidimensionale. În final atribui toate valorile necesare în variabila *result*.

Ne creem următoarele funcții necesare în rezolvarea problemei: ProbFunc, MediaFunc, DispersiaFunc, compoz, method, afisare. *ProbFunc* utilizeză funcția prob și returnează rezultatul; de asemenea tratăm și cazul probabilității condiționate. *MediaFunc* returnează rezultatul în urma aplicării funcției ce calculeaza media, iar *DispersiaFunc* va returna rezultatul după aplicarea funcției ce calculează dispersia. Aceste funcții de care m-am folosit, care calculează media și dispersia au fost definite în exercițiul 5. Avem nevoie de o funcție auxiliară ce va face compunerea de funcții, aceasta fiind *compoz*. Funcția *method* va fi cea care aplică variabilei *retval* toate calculele de care avem nevoie. În final creem propria funcție de *afisare*, prin care utilizatorului i se vor prezenta: densitatea de probabilitate introdusă de acesta, dacă avem variabilă aleatoare bidimensională, se prezintă și baza densității lui X si Y (luând în calcul și cazul când intervalele pot fi egale și afisez o singură dată), sau dacă avem variabilă aleatoare unidimensională, se prezintă doar baza densității lui X, urmând ca la final să afișez și capătul inferior al integralei.

După această rezolvare, testăm prin exemple corespunzătoare. Atribui variabilelor X, Y, Z informațiile corespunzătoare unor obiecte de tip variabile aleatoare continue, pornind de la o densitate de probabilitate introdusă de utilizator.

Implementare:

```
#' @export
setClass("myclass", slots=list (density ="function", densityX =
"function", densityY = "function",lower="function", bidimen="logical",
base="list"))

#' @export
myclass <- function( density, densityX = function(x) NULL, densityY =
function(x) NULL, lower = function(x) x, bidimen = FALSE, base =
list(c(-Inf, Inf)) )

{
  # cand avem v.a bidimensionale, baza va fi formata din listele bazei
densitatii lui X si Y.
  # cand avem v.a unidimensionale, baza va fi lista intervalelor
enumerate.

  if( length(base) < 2 ) # verific lungimea listei din baza
  { base <- list(base, list()) }

  if ( bidimen )
    if ( missing(lower) )# completez capatul inferior pentru v.a
unidimensionale
    { lower = function(x, y) x * y }

  result <- new("myclass", density = density, lower = lower, bidimen =
bidimen,
                base = base, densityX = densityX, densityY = densityY)
  #atribui toate valorile necesare in result
  return (result)
}

#' @export
setGeneric("ProbFunc", function(object) standardGeneric("ProbFunc"))
setMethod("ProbFunc", "myclass",
          function (object) {
            return (prob(object))
          })

#' @export
# tratez si cazul probabilitatii conditionate
setMethod("ProbFunc", "numeric",
          function (object) {
            return (object)
          })

#' @export
setGeneric("MediaFunc", function(object) standardGeneric("MediaFunc"))
setMethod("MediaFunc", "myclass",
          function(object){
```

```
            return(medieVA(object))
          })

#' @export
setGeneric("DispersiaFunc", function(object)
standardGeneric("DispersiaFunc"))
setMethod("DispersiaFunc", "myclass",
          function(object) return(dispersieVA(object)))
#' @export
compoz <- function(f, g)
{
  function(...) f(g(...))
}

#' @export
setGeneric("method", function(object, f) standardGeneric("method"))
setMethod("method", "myclass",
          function(object, f){
            retval <- myclass(object@density, Vectorize(compoz(f,
object@lower)),
                              object@bidimen, object@base[[1]])
            # baza densitatii in cazul v.a unidimensionale, adica al
lui X
          })
#' @export
# functia de afisare dupa cum imi trebuie
setGeneric("afisare", function(object) standardGeneric("afisare"))
setMethod("afisare", "myclass",
          function (object) {
            cat("Densitatea de probabilitate: ")
            print(body(fun = object@density))

            cat("Variabila Aleatoare Bidimensionala: ",
object@bidimen, "\n")

            cat("Baza densitatii: ")

            if ( object@bidimen == TRUE)  # cand avem v.a.
bidimensionale

              for (i in object@base[[1]]) # baza densitatii lui X
                for (j in object@base[[2]]) # baza densitatii lui Y
                {
                  if ( i[1] == j[1] & i[2] == j[2]) # baza densitatii
lui X si Y cand sunt egale
                    cat("[", i[1], ",", i[2], "] ")
                  else
                  { cat("[", i[1], ",", i[2], "] ")
                    cat( "U")
                    cat(" [", j[1], ",", j[2], "] ") }
                }
```

```
            else
              # cand avem v.a. unidimensionale
              for (i in object@base[[1]]) # baza densitatii lui X
                cat("[", i[1], ",", i[2], "] ")

            cat("\n")
            cat("Capatul inferior: ")
            print(body(fun = object@lower))
          })
```

Teste:

```
X <- myclass(density = function(x,y) exp(-(x^2)*y/2), bidimen = TRUE,
             base = list(list(c(-1, 1)), list(c(2, 3))))
afisare(X)

Y <- myclass(density = function (x) 1/((x+1)*sqrt(x)), bidimen =
FALSE,
             base = list(c(0, 1)))
afisare(Y)

Z <- myclass(density = function(x,y) sqrt(exp(-(x^2)*y/2)), bidimen =
TRUE,
             base = list(list(c(0, 2)), list(c(0, 2))))
afisare(Z)
```

```
> # Teste
> X <- myclass(density = function(x,y) exp(-(x^2)*y/2),
+              bidimen = TRUE,
+              base = list(list(c(-1, 1)), list(c(2, 3))))
> afisare(X)
Densitatea de probabilitate: exp(-(x^2) * y/2)
Variabila Aleatoare Bidimensionala:  TRUE
Baza densitatii: [ -1 , 1 ] U [ 2 , 3 ]
Capatul inferior: x * y
> Y <- myclass(density = function (x) 1/((x+1)*sqrt(x)),
+              bidimen = FALSE,
+              base = list(c(0, 1)))
> afisare(Y)
Densitatea de probabilitate: 1/((x + 1) * sqrt(x))
Variabila Aleatoare Bidimensionala:  FALSE
Baza densitatii: [ 0 , 1 ]
Capatul inferior: x
> Z <- myclass(density = function(x,y) sqrt(exp(-(x^2)*y/2)),
+              bidimen = TRUE,
+              base = list(list(c(0, 2)), list(c(0, 2))))
> afisare(Z)
Densitatea de probabilitate: sqrt(exp(-(x^2) * y/2))
Variabila Aleatoare Bidimensionala:  TRUE
Baza densitatii: [ 0 , 2 ]
Capatul inferior: x * y
>
```

Cerința 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.

Descrierea soluției:

Pentru a rezova cerința am creat câte o funcție diferită care să contureze graficul generat de valorile date parametrilor, ținând cont de tipul repartiției. Am afișat densitatea și funcția de repartiție pentru fiecare separat. Repartițiile pentru care am creat funcții de afișare a graficelor sunt urmatoărele :

- Uniformă
- Normală
- Exponențială
- Gamma
- Beta
- Cauchy

Funcții folosite : -dunif, punif, dexp, pexp, dnorm, pnorm, dgamma, pgamma, dbeta, pbeta, dcauchy, pcauchy;

Ne-am folosit de funcțiile de mai sus pentru a genera densitatea și funcția de repartiție a fiecarui tip de repartiție, unde

p+nume_repartiție = funcția de repartiție

d+nume_repartiție = densitatea repartiției

și le-am afișat graficele folosind funcția curve din R, unde exp = p/d + nume_repartiție(parametri specifici fiecarei repartiții).

Implementare:

```
#DENSITATEA SI FUNCTIA DE REPARTITIE PENTRU REP. NORMALA
#' @export
densitate_n <- function(medie = 0, v = 1)
```

```
{
  #de medie Âµ si variantÄƒ Ïƒ^2
  curve(expr = dnorm(x, medie, v),
        from = medie - 3 * v,
        to   = medie + 3 * v,
        ylab = "fi(x)",
        main = "Densitatea repartitiei normale",
        col = "red",
        lwd =2)
}
#' @export
repartitie_n <- function(medie = 0, v = 1) {
  #de medie Âµ si variantÄƒ Ïƒ^2
  curve(expr = pnorm(x, medie, v),
        from = medie - 3 * v,
        to   = medie + 3 * v,
        ylab = "FI(x)",
        main = "Functia de repartitie normala",
        col = "blue",
        lwd =2)
}
#DENSITATEA SI FUNCTIA DE REPARTITIE PENTRU REP. UNIFORMA
#X repartizatÄƒ uniform pe intervalul [a, b]
#' @export
densitate_u <-function(a, b)
{
  curve(expr = dunif(x, a, b),
        from = - 3 * b,
        to   =  3 * b,
```

```
        ylab = "fX(x)",
        main = "Densitatea repartitiei uniforme",
        col = "red",
        lwd =2)
}
#' @export
repartitie_u <- function(a, b)
{
  curve(expr = punif(x, a, b),
        from = - 3 * b,
        to   =  3 * b,
        ylab = "FX(x)",
        main = "Functia de repartitie uniforma",
        col = "blue",
        lwd =2)
}
#DENSITATEA SI FUNCTIA DE REPARTITIE BETA
#X este repartizatÄƒ Beta de parametrii (Î±, Î²), cu Î±, Î² > 0,
#' @export
densitate_b <- function(a, b) {
  curve(expr = dbeta(x, a, b),
        from = 0,
        to   = 1,
        ylab = "f(x)",
        main = "Densitatea in repartitia beta",
        col = "maroon",
        lwd =2)
}
```

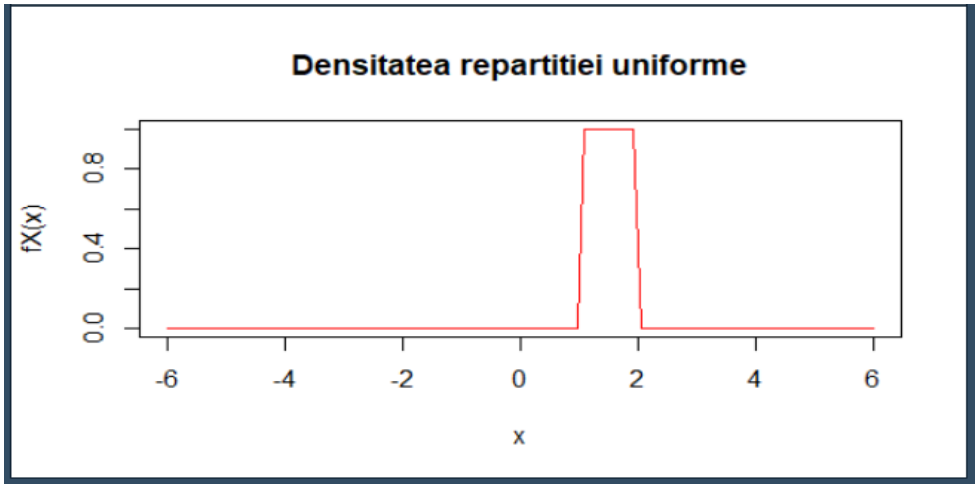
```
#' @export
repartitie_b <- function(a, b)
{
  curve(expr = pbeta(x, a, b),
        from = 0,
        to   = 1,
        ylab = "F(x)",
        main = "Functia de repartitie beta",
        col = "blue",
        lwd =2)
}
#DENSITATEA SI FUNCTIA DE REPARTITIE GAMMA
# X este repartizatÄƒ Gama de parametrii (Î±, Î²), cu Î±, Î² > 0,
#' @export
densitate_g <- function(a, b){

  curve(expr = dgamma(x, a, b),
        from = 0,
        to   = 20,
        ylab = "f(x)",
        main = "Densitatea in repartitia gamma",
        col = "maroon",
        lwd =2)
}
#' @export
repartitie_g <- function(a, b){
  curve(expr = pgamma(x, a, b),
        from = 0,
        to   = 20,
```

```
        ylab = "F(x)",
        main = "Functia de repartitie gamma",
        col = "red",
        lwd =2)
}
#DENSITATEA SI FUNCTIA DE REPARTITIE EXPONENTIALA
#X este repartizatÄƒ exponential de parametru Î»
#' @export
densitate_e <- function(lbd){
  curve(expr = dexp(x, lbd),
        from = 0,
        to   = 20,
        ylab = "f(x)",
        main =" Densitatea in repartitie exponentiala",
        col = "red",
        lwd =2)
}
#' @export
repartite_e <- function(lbd){
  curve(expr = pexp(x, lbd),
        from = 0,
        to   = 5,
        ylab = "F(x)",
        main ="Functia de repartitie exponentiala",
        col = "orange",
        lwd =2)
}
#DENSITATEA SI FUNCTIA DE REPARTITIE CAUCHY
#X este repartizatÄƒ Cauchy de parametrii (0, 1)
```

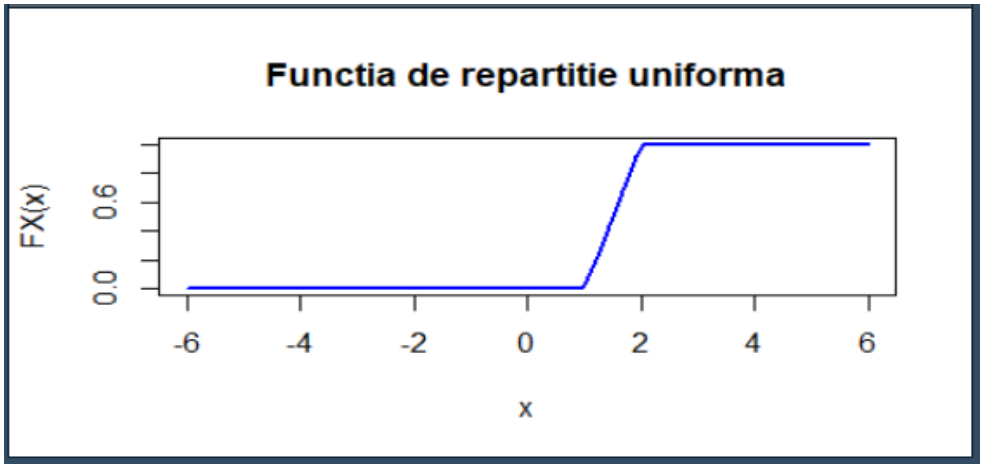
```
#' @export
densitate_c <- function(loc, sc){
  curve(expr = dcauchy(x, loc, sc),
        from = 0,
        to   = 5,
        ylab = "f(x)",
        main ="Densitatea de repartitie Cauchy",
        col = "orange",
        lwd =2)
}
#' @export
repartitie_c <- function(loc, sc){
  curve(expr = pcauchy(x, loc, sc),
        from = 0,
        to   = 5,
        ylab = "F(x)",
        main ="Functia de repartitie Cauchy",
        col = "blue",
        lwd =2)
}
```

Teste:

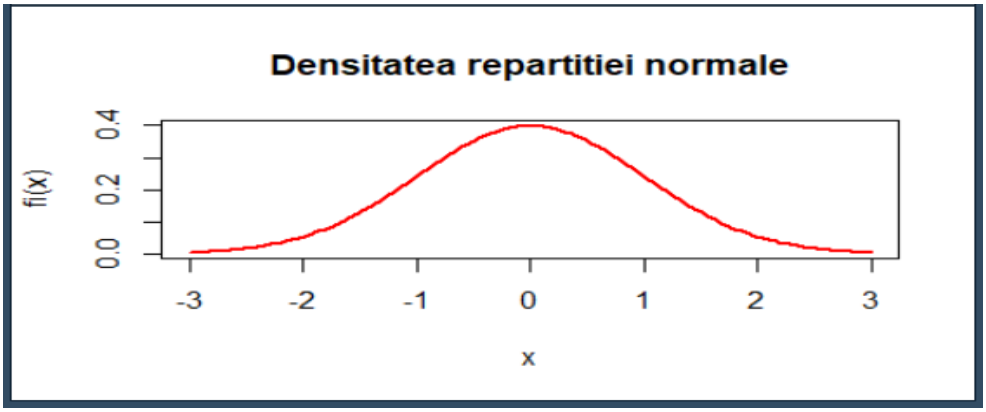
1.densitate_u(1, 2) :



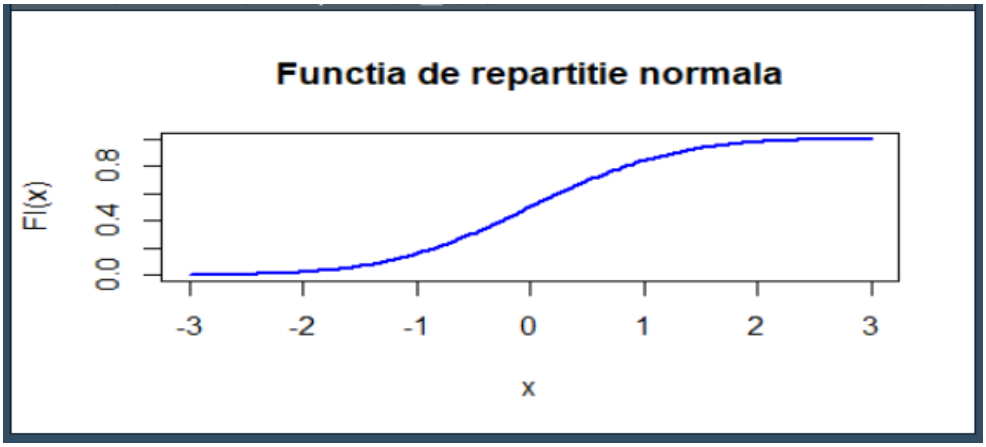
2.repartitie_u(1, 2) :



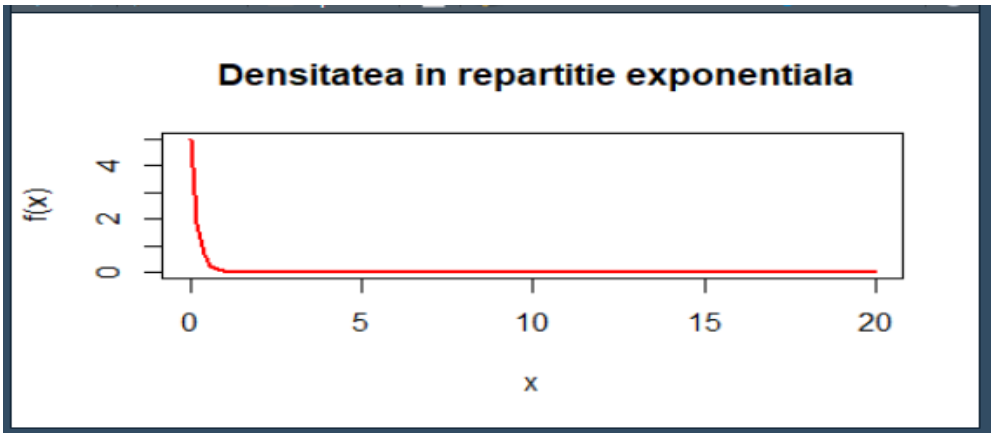
3.densitate_n(0,1):



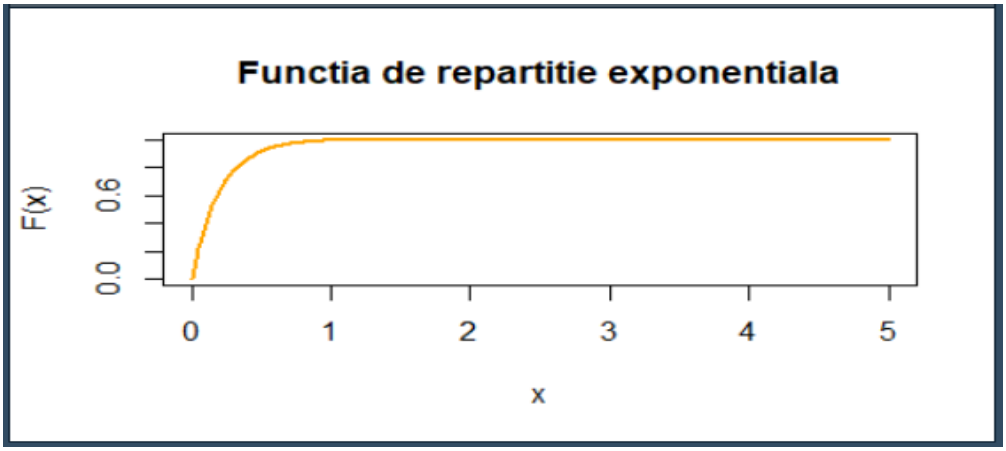
4.repartitie_n(0,1):



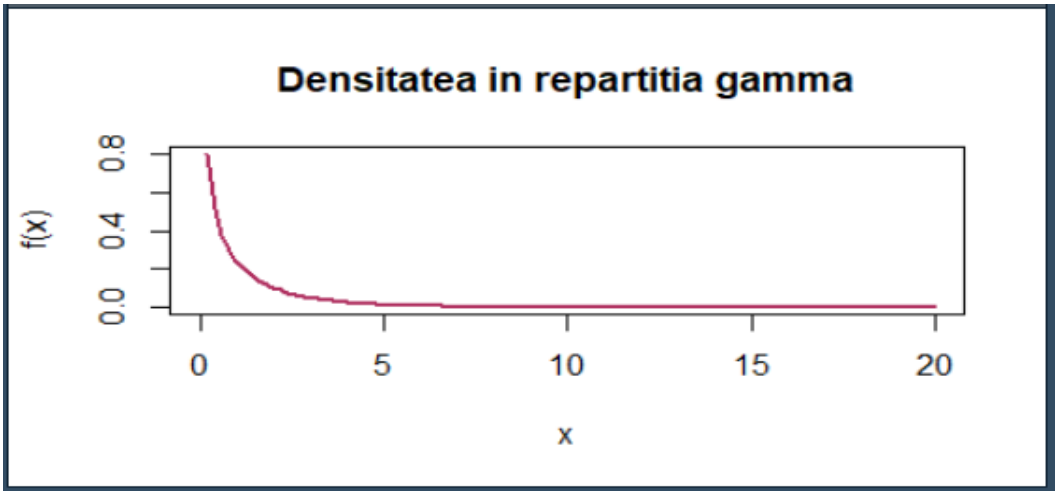
5.densitate_e(5) :



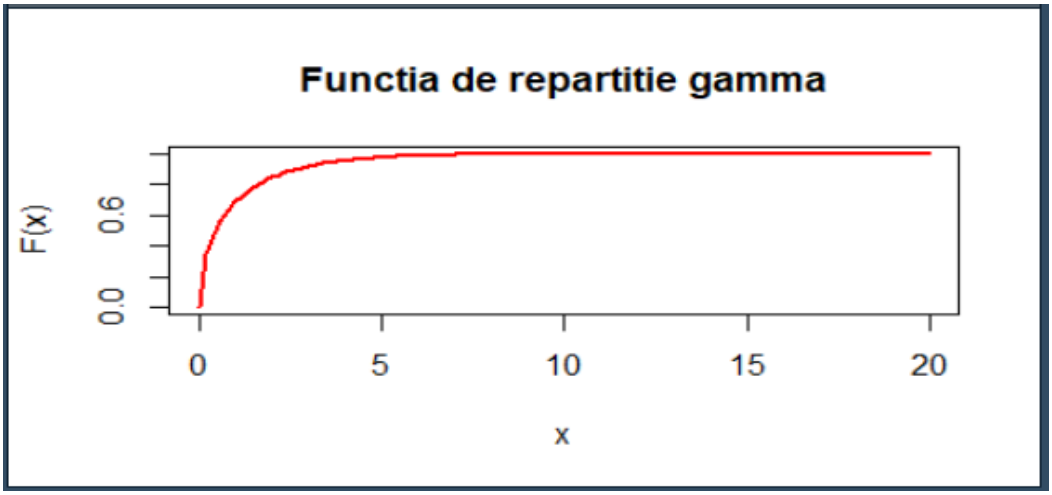
6.repartite_e(5):



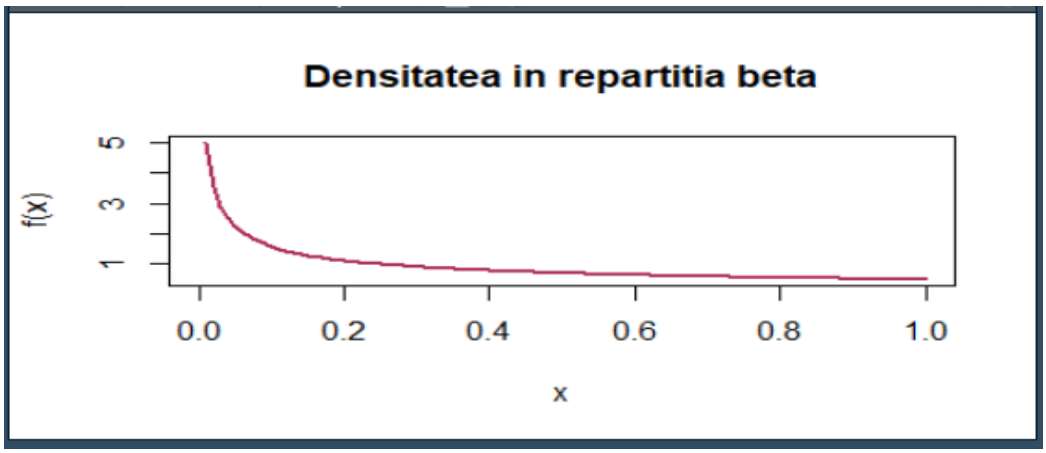
7.densitate_g(0.5, 0.5) :



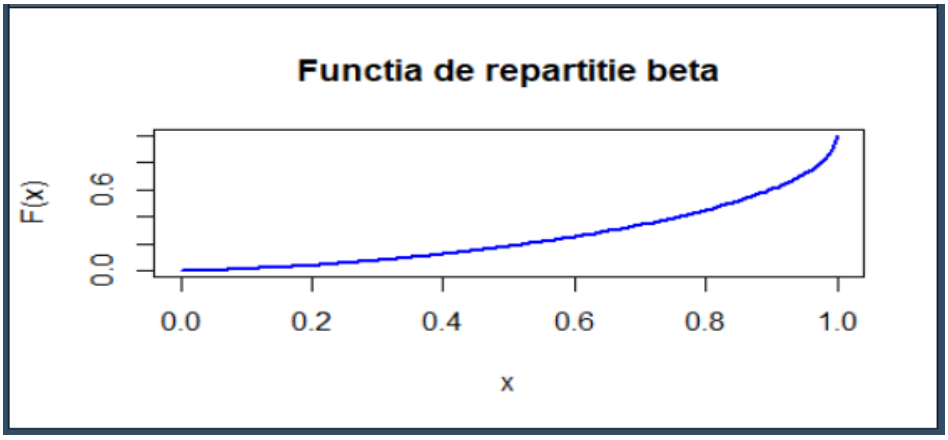
8.repartitie_g(0.5, 0.5) :



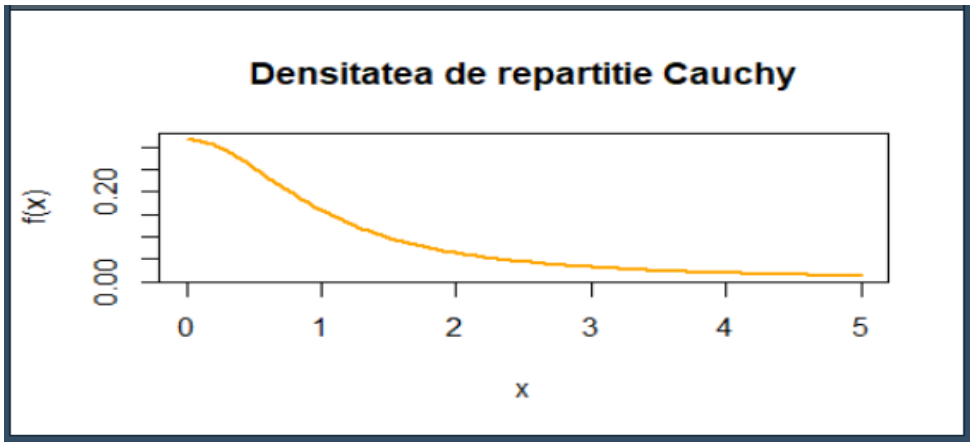
9.densitate_b(0.5, 1):



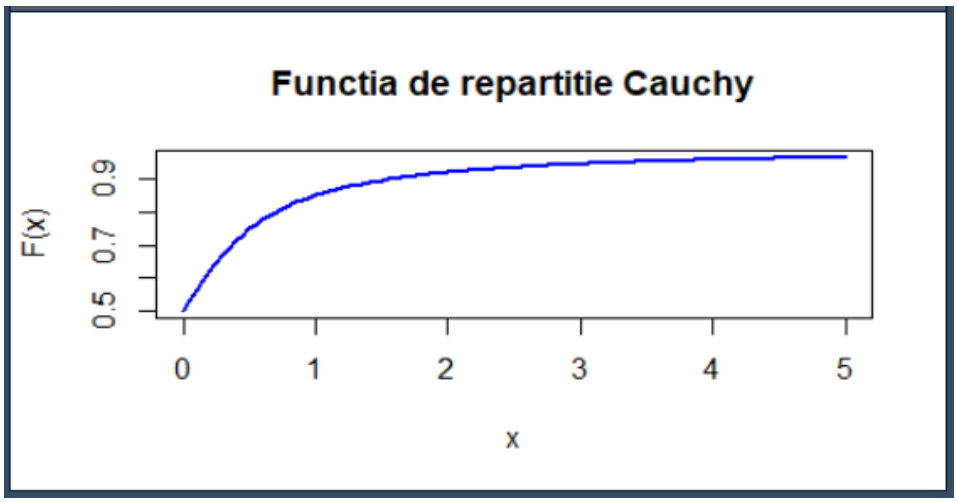
10.repartitie_b(1.5, 0.5) :



11.densitate_c(0, 1) :



12.repartitie_c(0, 0.5):



Cerința 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.

Descrierea soluției:

În rezolvarea acestei cerințe, am construit mai multe funcții, pentru fiecare formulă de calculat. Fiecare funcție se bazează pe aceeași structură cu block-uri try-catch, pentru a putea identifica eventualele erori în calculul integralelor, ceea ce determină inexistența mediei, dispersiei sau unui moment. Pentru funcțiile corespunzătoare momentelor inițiale și centrate vom aplica formula în cadrul unui for-loop care parcurge intervalul natural {1,2,3,4}, calculând astfel momentele până la ordinul 4. În funcțiile corespunzătoare momentelor, vom reține fiecare moment într-o listă pe care o vom returna la final.

Formulele folosite sunt:

- media $= \int_{-\infty}^{\infty} x * f(x)dx$
- dispersia $= \int_{-\infty}^{\infty} (x - media) * f(x)dx$
- momentul inițial de ordinul i $= \int_{-\infty}^{\infty} x^{i} * f(x)dx$
- momentul centrat de ordinul i $= \int_{-\infty}^{\infty} (x\text{-}media)^{i} * f(x)dx$

Implementare:

```
#' @export
medieVA <- function(f)
{   tryCatch({
    fcalcul<- function(x) { x*f(x)}
    result<-integrate(Vectorize(fcalcul),   lower   =   -Inf,   upper=
Inf)$value
    return (result)
  },
  error = function(err){
    message(paste("Nu se poate calcula media -> eroare:", err))
  })
}
```

```
#calcularea dispersiei
#' @export
dispersieVA <- function(f)
{
  tryCatch({
    fcalcul<- function(x) { ((x-medieVA(f))^2)*f(x)}
    result<-integrate(Vectorize(fcalcul), lower = -Inf, upper=
Inf)$value
    return (result)
  },
  error = function(err){
    message(paste("Nu se poate calcula dispersia -> eroare:", err))
  })
}
#calcularea momentelor initiale
#' @export
momenteInitVA <- function(f)
{
  lista<- list()
  tryCatch({
    for(i in 1:4)
    {
      tryCatch({
        fcalcul<- function(x) { ((x)^i)*f(x)}
        result<-integrate(Vectorize(fcalcul), lower = -Inf, upper=
Inf)$value
        ##print(result)
        lista<-append(lista, result)
      },
      error = function(err){
        message(paste("Nu se poate calcula momentul initial de ordin",
i))
      })
    }
```

```
    return(lista)
  },
  error = function(err){
    message(paste("Eroare:", err))
  })
}
#calcularea momentelor centrate
#' @export
momenteCentVA <- function(f)
{
  lista<- list()
  tryCatch({
    for(i in 1:4)
    {
      tryCatch({
        t<-medieVA(f)
        fcalcul<- function(x) { ((x-t)^i)*f(x)}
        result<-integrate(Vectorize(fcalcul), lower = -Inf, upper=
Inf)$value
        ##print(result)
        lista<-append(lista, result)
      },
      error = function(err){
        message(paste("Nu se poate calcula momentul centrat de ordin",
i))
      })
    }
    return(lista)
  },
  error = function(err){
    message(paste("Eroare:", err))
  })
}
```

Teste:

1.medieVA(function(x) {exp(-(x^2)/2)})

medieVA(function(x) {exp(x)/((1+exp(x))^2)})

```
> medieVA(function(x) {exp(-(x^2)/2)})
[1] 0
> medieVA(function(x) {exp(x)/((1+exp(x))^2)})
[1] 3.63644e-17
>
```

2. momenteInitVA(function(x) {exp(-(x^2)/2)})

```
> momenteInitVA(function(x) {exp(-(x^2)/2)})
[[1]]
[1] 0

[[2]]
[1] 2.506628

[[3]]
[1] 0

[[4]]
[1] 7.519885
```

3. momenteCentVA(function(x) {exp(x)/((1+exp(x))^2)})

momenteCentVA(function(x) {exp(-(x^2)/2)})

```
> momenteCentVA(function(x) {exp(x)/((1+exp(x))^2)})
Nu se poate calcula momentul centrat de ordin 2
Nu se poate calcula momentul centrat de ordin 4
[[1]]
[1] 2.603802e-17

[[2]]
[1] -3.795956e-16

> momenteCentVA(function(x) {exp(-(x^2)/2)})
[[1]]
[1] 0

[[2]]
[1] 2.506628

[[3]]
[1] 0

[[4]]
[1] 7.519885
```

Cerința 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.

Descrierea soluției:

În rezolvarea acestei cerințe vom implementa din nou mai multe funcții, una pentru calculul mediei și una pentru calculul dispersiei. Aceste funcții implementează formulele de calcul pentru media și dispersia variabilei g(X). Astfel că, funcțiile vor primi ca parametrii 2 funcții și vor returna rezultatul formulelor.

Formulele folosite:

- $media = \int_{-\infty}^{\infty} g(x) * f(x)dx$
- dispersia $= \int_{-\infty}^{\infty} (g(x) - media) * f(x)dx$

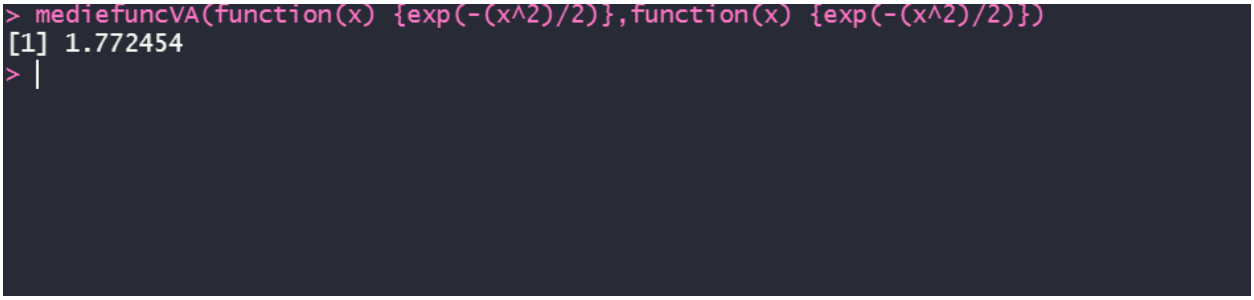
Implementare:

```
#calcularea mediei g

#' @export
mediefuncVA <- function(f,g)
{
  tryCatch({
    fcalcul<- function(x) { g(x)*f(x)}
    result<-integrate(Vectorize(fcalcul),   lower   =   -Inf,   upper=
Inf)$value
    return (result)
  },
  error = function(err){
    message(paste("Nu se poate calcula media cu functia g -> eroare:",
err))
  })
}
#calcularea dispersiei g

#' @export
dispersiefuncVA <- function(f,g)
{
  tryCatch({
```

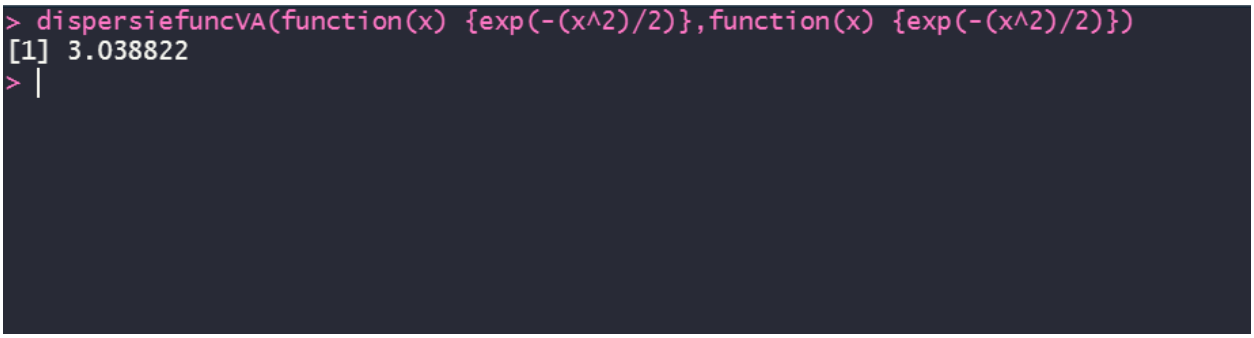
```
    fcalcul<- function(x) { ((g(x)-mediefuncVA(f,g))^2)*f(x)}
    result<-integrate(Vectorize(fcalcul), lower = -Inf, upper=
Inf)$value
    return (result)
  },
  error = function(err){
    message(paste("Nu se poate calcula dispersia cu functia g->
eroare:", err))
  })
}
```

Teste:

1.mediefuncVA(function(x) {exp(-(x^2)/2)},function(x) {exp(-(x^2)/2)})



2.dispersiefuncVA(function(x) {exp(-(x^2)/2)},function(x) {exp(-(x^2)/2)})



Cerința 8: Afișarea unei “fișe de sinteză” care să conțină informații de bază despre respective 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.

Descrierea soluției:

Pentru rezolvarea acestui exercițiu am creat o funcție care primește ca parametru un string, care poate lua valori : “gama”, “beta”, “norm”, “unif”, “lognorm”, “exp”, “student”, “x2”, “cauchy” sau “all”.

Pentru fiecare valoare dată ca parametru se va afișa o “fișă de sinteza” pentru repartiția cu numele parametrului dat. În cazul în care parametrul dat este “all”, atunci se vor afișa informațiile despre toate repartițiile, iar în cazul în care parametrul dat nu se află printre cele de mai sus se va afisa un mesaj corespunzator.

Deși codul este destul de explicit, pe scurt, am luat câte o variabilă care să rețină pentru fiecare tip de repartiție DEFINIȚIA, NOTAȚIA, MEDIA, VARIANȚA și SURSA de inspirație și în funcție de apelul funcției se vor afișa detaliile cerute.

Implementare:

```
#' @export
fisa_sinteza <-function(val){

  normala <- c(

               "    REPARTITIE NORMALA :       ",

               " -> Definitie : Spunem că o variabilă aleatoare X este
repartizată normal sau Gaussian de medie μ s, i variantă σ ^ 2, dacă
densitatea ei are forma f(x) = ϕ(x) = (1 / √(2 * π * σ)) * (e ^ ((- (x
- μ) ^ 2) / ((2 * σ) ^ 2)) , x ∈ R.",

               " -> Notatie : X ~ N (μ, σ2)",

               " -> Media : E[X] = μ",

               " -> Varianta : Var(X) = σ^2",

               " -> Sursa :
https://moodle.unibuc.ro/pluginfile.php/121742/mod_resource/content/1/
Lab6.pdf"

               )

  lognorm  <- c("    REPARTITIE LOGNORMALA :       ",

                " -> Definitie : Spune că o variabilă aleatoare X este
repartizată log-normal de parametrii μ si σ^2, dacă ln(X) este
repartizată normal de parametrii μ si σ^2. Cu alte cuvinte dacă Y ~ N
(μ, σ2) atunci X = e^Y ~ LN(μ, σ^2). ",
```

```
                " -> Notatie : X ~ LN(μ, σ^2)",

                " -> Media : E[X] = e ^(μ + (σ^2)/2)",

                " -> Varianta : Var(X) = (e ^ (σ^2)- 1) * (e ^ (2μ +
σ^2))",

                " -> Sursa : Curs: Probabilităt, i s, i Statistică
(2020-2021), A. Amărioarei"

                )

  uniforma <- c("    REPARTITIE UNIFORMA :       ",

                " -> Definitie : O variabilă aleatoare X repartizată
uniform pe intervalul [a, b], are densitatea dată de f(x) = { 1/(b -
a), x ∈ [a, b] si  0, altfel}",

                " -> Notatie : X ~ U[a, b]",

                " -> Media :  E[X] = (a + b) / 2",

                " -> Varianta : Var(X) = [(a - b) ^ 2] / 12 ",

                " -> Sursa :
https://moodle.unibuc.ro/pluginfile.php/121742/mod_resource/content/1/
Lab6.pdf"

                )


  beta <- c("    REPARTITIE BETA :       ",

            " -> Definitie : Spunem că o variabilă aleatoare X este
repartizată Beta de parametrii (α, β), cu α, β > 0, dacă densitatea ei
are forma f(x) = (1/B(α, β))* (x ^ (α-1))*(1 - x)^(β-1), 0 ≤ x ≤ 1,
unde B(α, β) este functia (Beta, numită s, i integrală Euler de primul
tip) definită prin B(α, β) = integrala de la 0 la infinit din ((x ^
(α-1)) *(1 - x)^(β-1) dx, ∀α, β > 0.",

            " -> Notatie : X ~ B(α, β)",

            " -> Media : E[X] = α/(α + β)",

            " -> Varianta :  Var(X) = (α * β)/(((α + β) ^ 2) * (α + β
+ 1))",

            " -> Sursa :
https://moodle.unibuc.ro/pluginfile.php/121742/mod_resource/content/1/
Lab6.pdf"

            )
```

```
  gama <- c(
            "    REPARTITIE GAMMA :       ",
            " -> Definitie : Spunem că o variabilă aleatoare X este
repartizată Gama de parametrii (α, β), cu α, β > 0, dacă densitatea ei
are forma f(x) = ((β ^ α) / Γ(α)) * (x ^(α-1)) * (e^ (-βx)), ∀x > 0,
unde Γ(α) este funct, ia (Gama, numită s, i integrală Euler de al
doilea tip) definită prin Γ(α) = integrala de la 0 la infint din ((x ^
(α-1)) * (e ^(-x)) dx, ∀α > 0.",
            " -> Notatie : X ~ Γ(α, β)",
            " -> Media :  E[X] = α / β",
            " -> Varianta : Var(X) =  α / (β ^ 2)",
            " -> Sursa :
https://moodle.unibuc.ro/pluginfile.php/121742/mod_resource/content/1/
Lab6.pdf"
            )
  exp <- c(
           "    REPARTITIE EXPONENTIALA :       ",
           " -> Definitie : Spunem că o variabilă aleatoare X este
repartizată exponential de parametru λ, dacă densitatea ei are forma
f(x) = (λe) ^ (-λx), ∀x ∈ R.",
           " -> Notatie : X ~ E(λ)",
           " -> Media : E[X] = 1/λ",
           " -> Varianta : Var(X) = 1/(λ^2)",
           " -> Sursa :
https://moodle.unibuc.ro/pluginfile.php/121742/mod_resource/content/1/
Lab6.pdf"
           )
  cauchy <- c(
              "    REPARTITIE CAUCHY :       ",
              " -> Definitie : Spunem că o variabilă aleatoare X este
repartizată Cauchy de parametrii (0, 1), dacă densitatea ei are
forma  fX(x) = (1/π) * (1/(1 + x^2)), ∀x ∈ R.",
              " -> Notatie : X ~ C(0, 1)",
              " -> Media : NU EXISTA",
```

```
              " -> Varianta : NU EXISTA",

              " -> Sursa :
https://moodle.unibuc.ro/pluginfile.php/121742/mod_resource/content/1/
Lab6.pdf"

              )

  student <- c(

               "    REPARTITIE T/STUDENT :       ",

               " -> Definitie : Spunem că o variabilă aleatoare X este
repartizată Student( are distributie Student) cu n grade de libertate
daca poate fi scrisa sub forma X=Y/√(Z/n), unde Y ~ N(0,1) este o
variabila normala standard iar Z ~ X^2(n) este o variabila aleatoare
X^2 cu n grade de libertate independenta de Y.",

               " -> Notatie : X ~ T(n), n = nr de grade de libertate",

               " -> Media : E[X] = integrala de la -inf la + inf din
x*f(x) dx = 0, n > 1",

               " -> Varianta : Var(X) = E[X^2]-E[X]^2 = n/(n-2)",

               " -> Sursa :
http://cs.unitbv.ro/~pascu/stat/Distributii%20continue%20clasice.pdf"

               )

  x2 <- c(

          "    REPARTITIE X-PATRAT :       ",

          " -> Definitie : O variabila aleatoare X este distribuita X-
patrat cu n grade de libertate daca are densitatea de probabilitate de
forma: f(x) = (1/(2^(n/2) * gamma(n/2))) * x^((n/2)-1) * e^(-x/2),
pentru x > 0",

          " -> Notatie : X ~ X^2(n), n = nr de grade de libertate",

          " -> Media : E[X] = integrala de la -inf la inf din x*f(x)
dx = n",

          " -> Varianta : Var(X) = E[X^2]-E[X]^2 = 2n",

          " -> Sursa :
http://math.etc.tuiasi.ro/rstrugariu/cursuri/SPD2015/c7.pdf"

          )

  if(val == "beta")

    print(noquote(beta), justify = "right"

  else if(val == "norm")
```

```
  print(noquote(normala), justify = "right")
 else if(val =="lognorm")
  print(noquote(lognorm), justify = "right")
 else if(val == "unif")
  print(noquote(uniforma), justify = "right")
 else if(val == "exp")
  print(noquote(exp), justify = "right")
 else if(val =="gama")
  print(noquote(gama), justify = "right")
 else if(val == "student")
  print(noquote(student), justify = "right")
 else if(val == "x2")
  print(noquote(x2), justify = "right")
 else if(val =="cauchy")
  print(noquote(cauchy), justify = "right")
else if(val == "all")
 {
  print(noquote(normala))
  cat("\n\n")
  print(noquote(lognorm))
  cat("\n\n")
  print(noquote(uniforma))
  cat("\n\n")
  print(noquote(exp))
  cat("\n\n")
  print(noquote(beta))
  cat("\n\n")
  print(noquote(gama))
  cat("\n\n")
```

```
    print(noquote(student))
    cat("\n\n")
    print(noquote(x2))
    cat("\n\n")
    print(noquote(cauchy))
  }
else
     print(noquote("Nu exista o astfel de repartitie!"))
}
```

Teste:

fisa_sinteza("unif")

```
> fisa_sinteza("unif")
[1]     REPARTITIE UNIFORMA :

[2]  -> Definitie : O variabila aleatoare X repartizata uniform pe intervalul [a, b], are densitatea data de f(x) = { 1/(b ??? a), x ??? [a
 b] si  0, altfel}
[3]  -> Notatie : X ??? U[a, b]

[4]  -> Media :  E[X] = (a + b) / 2

[5]  -> Varianta : Var(X) = [(a ??? b) ^ 2] / 12

[6]  -> Sursa : https://moodle.unibuc.ro/pluginfile.php/121742/mod_resource/content/1/Lab6.pdf
```

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

Descrierea soluției:

Inițial, verificăm dacă suportul lui x, respetiv y conține –infinit sau infinit, caz în care vom primi eroare la integrări. În caz afirmativ, afișăm un mesaj corespunzător și returnăm FALSE.

Calculăm densitățile marginale și mediile lui X și Y conform formulelor:

- $f_X(x) = \int_{-\infty}^{\infty} f(x,y)dy$, analog pentru $f_Y(y)$
- $E[X] = \int_{-\infty}^{\infty} x * f(x)dx$, analog pentru E[Y]

Apoi calculăm noua funcție pe care o vom folosi pentru aflarea covarianței, anume:

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

Calculăm și variațele lui X și Y conform:

- $Var(X) = E[(X - E[X])^2]$, analog pentru Var(Y)

În final, calculăm coeficientul de corelație conform formulei:

- $\rho(X,Y) = Cov(X,Y)VarX \cdot VarY$

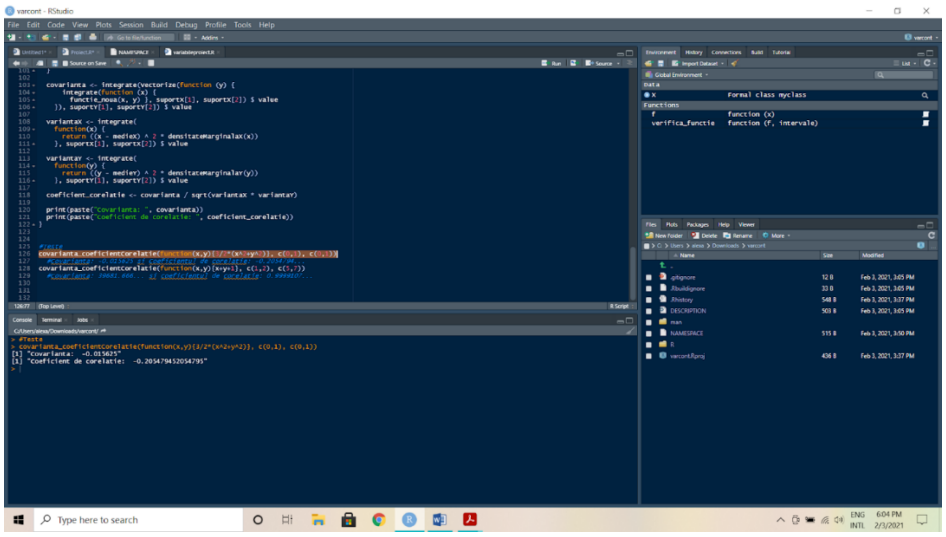
Implementare:

```
#' @export
covarianta_coeficientCorelatie <- function(f, suportX, suportY){

#daca un capat al unui suport nu este finit, vor aparea erori

  if(suportX[1] == -Inf || suportX[2] == Inf || suportY[1] == -Inf ||
suportY[2] == Inf){

    print("Imposibil de calculat: un capat al unui interval este -Inf
sau Inf")

    return(FALSE)

  }

#calculam conform formulelor densitatile marginale si mediile
variabilelor aleatoare X si Y

  densitateMarginalaX <- Vectorize(function(x){

    integrate(function(y){

      f(x,y)}, suportY[1], suportY[2])$value})

 medieX <- integrate(function(x){

    return (x * densitateMarginalaX(x))

  }, suportX[1], suportX[2])$value


  densitateMarginalaY <- Vectorize(function(y){

    integrate(function(x){

      f(x,y)

    }, suportX[1], suportX[2])$value

  })

  medieY <- integrate(function(y){

    return (y * densitateMarginalaY(y))

  }, suportY[1], suportY[2])$value

#functia necesara pentru calculul covariantei

  functie_noua <- function(x,y){

    return ((x - medieX) * (y - medieY) * f(x, y))

  }

  covarianta <- integrate(Vectorize(function (y) {
```

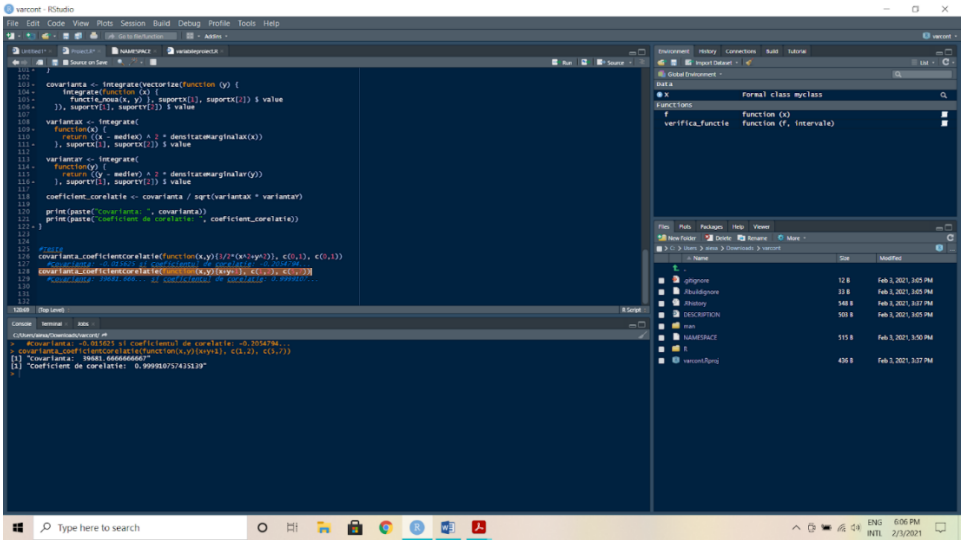
```
      integrate(function (x) {
        functie_noua(x, y) }, suportX[1], suportX[2]) $ value
    }), suportY[1], suportY[2]) $ value
#variantele necesare calculului coeficientului de corelatie
  variantaX <- integrate(
    function(x) {
      return ((x - medieX) ^ 2 * densitateMarginalaX(x))
    }, suportX[1], suportX[2]) $ value
  variantaY <- integrate(
    function(y) {
      return ((y - medieY) ^ 2 * densitateMarginalaY(y))
    }, suportY[1], suportY[2]) $ value
#conform formulei
  coeficient_corelatie <- covarianta / sqrt(variantaX * variantaY)
  print(paste("Covarianta: ", covarianta))
  print(paste("Coeficient de corelatie: ", coeficient_corelatie))
}
```

Teste:

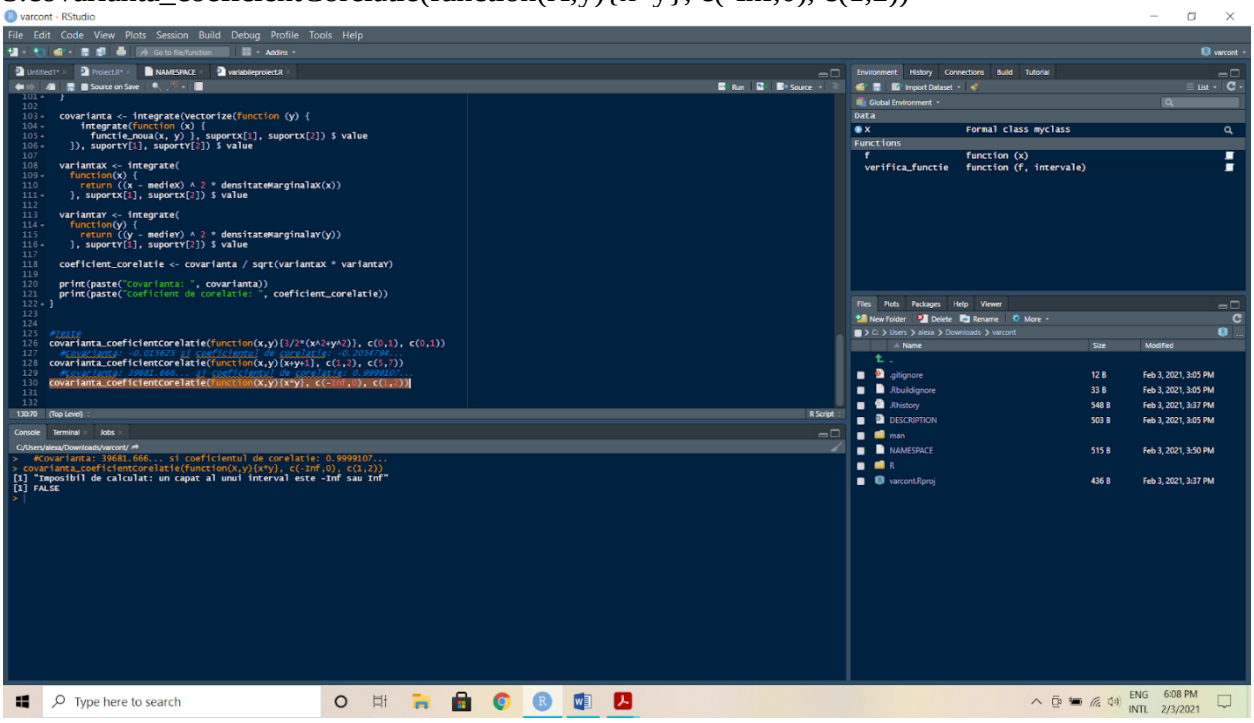
1.covarianta_coeficientCorelatie(function(x,y){3/2*(x^2+y^2)}, c(0,1), c(0,1))



2.covarianta_coeficientCorelatie(function(x,y){x+y+1}, c(1,2), c(5,7))



3.covarianta_coeficientCorelatie(function(X,y){x*y}, c(-Inf,0), c(1,2))



Cerința 12: Construirea sumei și diferenței a două variabile aleatoare continue independente (folosiți formula de convoluție)

Descrierea soluției:

Formula de convoluție: Convoluția lui f și g se scrie $f * g$, denotând operatorul cu simbolul $*$. Este definit ca integral al produsului celor două funcții după ce una este inversată și deplasată. Ca atare, este un anumit tip de transformare integrală :

$$(f * g)(t) \coloneqq \int_{-\infty}^{\infty} f(\tau) * g(t-\tau) d\tau$$

Cu ajutorul acestei formule de convoluție, construim sume, respectiv diferența a două variabile aleatoare continue independente.

Implementare:

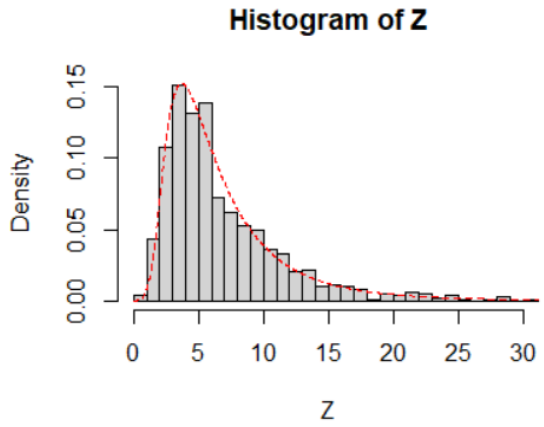
```
# f este o funct densitatea de probabilitate unei variabile aleatoare
X
# g este o funct densitatea de probabilitate unei variabile aleatoare
Y
# convolutionSum va returna o funct densitatea de probabilitate unei
variabile
# Z = X + Y
#' @export
convolutionSum <- function(f, g) {
  function(z) (integrate (function(x) (f(x) * g(z - x)), -Inf,
+Inf)$value)
}

# f este o funct densitatea de probabilitate unei variabile aleatoare
X
# g este o funct densitatea de probabilitate unei variabile aleatoare
Y
# convolutionDif va returna o funct densitatea de probabilitate unei
variabile
# Z = X - Y
#' @export
convolutionDif <- function(f, g) {
  function(z) (integrate (function(x) (f(x) * g(x - z)), -Inf,
+Inf)$value)
}
```

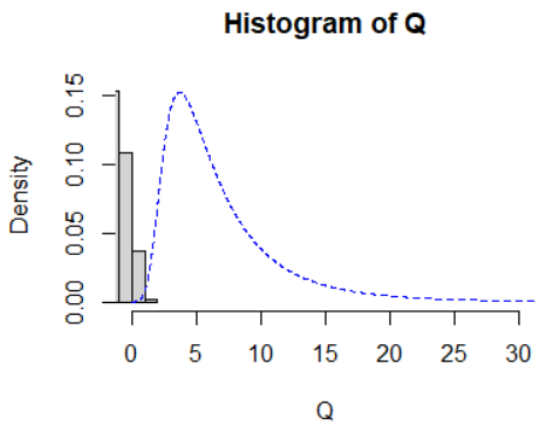
Teste:

```
f.X <- function(X) dnorm(X,1,0.5)         # normal (mu=1.5, sigma=0.5)
f.Y <- function(Y) dlnorm(Y,1.5, 0.75)    # log-normal (mu=1.5,
sigma=0.75)
```

```
# Construirea sumei si diferentei (folositi formula de convolutie)
f.Z <- convolutionSum(f.X, f.Y)
f.Z <- Vectorize(f.Z)                    # este necesar sa aplic
Vectorize

f.Q <- convolutionSum(f.X, f.Y)
f.Q <- Vectorize(f.Q)

set.seed(1)                              # pentru a reproduce
exemplele
X <- rnorm(1000,1,0.5)
Y <- rlnorm(1000,1.5,0.75)
Z <- X + Y
Q <- X - Y

# compare tmetodele folosite
hist(Z,freq=F,breaks=50, xlim=c(0,30))
z <- seq(0,50,0.01)
lines(z,f.Z(z),lty=2,col="red")
hist(Q,freq=F,breaks=50, xlim=c(0,30))
q <- seq(0,50,0.01)
lines(q,f.Q(q),lty=2,col="blue")
```

```
>
>
> f.X <- function(X) dnorm(X,1,0.5)        # normal (mu=1.5, sigma=0.5)
> f.Y <- function(Y) dlnorm(Y,1.5, 0.75)   # log-normal (mu=1.5, sigma=0.75)
>
> # Construirea sumei si diferentei (folositi formula de convolutie)
> f.Z <- convolutionSum(f.X, f.Y)
> f.Z <- Vectorize(f.Z)                    # este necesar sa aplic Vectorize
>
> f.Q <- convolutionSum(f.X, f.Y)
> f.Q <- Vectorize(f.Q)
>
> set.seed(1)                              # pentru a reproduce exemplele
> X <- rnorm(1000,1,0.5)
> Y <- rlnorm(1000,1.5,0.75)
> Z <- X + Y
> Q <- X - Y
>
> # compare tmetodele folosite
> hist(Z,freq=F,breaks=50, xlim=c(0,30))
> z <- seq(0,50,0.01)
> lines(z,f.Z(z),lty=2,col="red")
>
>
> hist(Q,freq=F,breaks=50, xlim=c(0,30))
> q <- seq(0,50,0.01)
> lines(q,f.Q(q),lty=2,col="blue")
>
```





III. Concluzii

Utilizarea acestui tip de pachete de funcții se bazează pe un fundament matematic, ce poate veni în sprijinul specialiștilor care conduc experimente și studii. Proiectul nostru este în sine o metodă, un model prin care aceste elemente teoretice , percepute de foarte mulți ca fiind abstracte, pot fi utilizate în aplicații concrete în viața reală, fapt ce conduce la sporirea rigurozității în tratarea și analiza problemelor precum și la consolidarea concluziilor ce rezultă din studiu. Astfel, deschidem un portal către aplicații pe baza implementării acestor funcții, transformând teoria probabilităților în instrumente informatice.

Așadar, echipa noastră a descris în paragrafele de mai sus soluțiile aduse pentru diferitele problematici apărute în lucrul cu variabilele aleatoare continue, element indispensabil atât în studiul probabilităților cât și consolidarea analizei structurilor statistice.

IV. Bibliografie

1. https://ro.qaz.wiki/wiki/Normalizing_constant
2. https://www.guru99.com/r-apply-sapply-tapply.html
3. https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/seq
4. https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/any
5. https://en.wikipedia.org/wiki/Probability_density_function
6. http://cs.unitbv.ro/~pascu/stat/Variabile%20aleatoare.pdf
7. http://math.etc.tuiasi.ro:81/rosu/didactic/MS%20II_Curs_Variabile%20aleatoare%20continue_I.pdf
8. https://ro.qaz.wiki/wiki/Convolution
9. https://stats.libretexts.org/Bookshelves/Probability_Theory/Book%3A_Introductory_Probability_(Grinstead_and_Snell)/07%3A_Sums_of_Random_Variables/7.01%3A_Sums_of_Discrete_Random_Variables
10. https://ibmi.mf.uni-lj.si/files/vaja2en_s.pdf
11. https://stackoverflow.com/questions/23569133/adding-two-random-variables-via-convolution-in-r
12. Curs – Probabilități și Statistică
13. http://web.mit.edu/insong/www/pdf/rpackage_instructions.pdf