Documentatie proiect Probabilitati si Statistica

Membri echipei: *Anghelache Vlad-Alexandru, Orzata Andrei, Ionescu Cristian-Andrei*
Lider de echipa: *Anghelache Vlad-Alexandru*
Grupa: *232*
Proiect: *Proiectul 1- Construirea unui pachet R pentru lucru cu variabile aleatoare continue*

Crearea pachetului *varAC*:

Pentru crearea pachetului s-au folosit pachetele ***devtools*** si ***roxygen2***, iar ca materiale ajutatoare am folosit documentul suport ***rpackage_instructions.pdf***, https://tinyheero.github.io/jekyll/update/2015/07/26/making-your-first-R-package.html, https://r-pkgs.org/description.html si https://hilaryparker.com/2014/04/29/writing-an-r-package-from-scratch/ .

Anghelache Vlad-Alexandru

Cerinta1:

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

Rezolvare:
Constanta de normalizare este o constanta cu care inmultim o functie pozitiva pentru a obtine o densitate de probabilitate, adica integrala cu capetele -infinit, infinit a functiei obtinute sa fie 1.
Sursa: Normalizing constant, Wikipedia - https://en.wikipedia.org/wiki/Normalizing_constant

Functia ***getNConst*** primeste ca paramentru o functie f si intoarce constanta de normalizare, daca aceasta exista, in caz contrar se returneaza NULL si se afiseaza mesajul '*Nu exista constanta de normalizare pentru functia data!*'. In cazul unei erori, se afiseaza si mesajul corespunzator erorii.
Initial functia verifica daca functia data ca parametru este pozitiva, apeland ***verif,*** care verifica daca minimul functiei f pe intervalul (-10^35, 10^35), limita la infinit si la minus infinit sunt mai mari sau egale cu 0. Daca valoarea returnata de ***verif*** este 0 atunci se afiseaza *'Functia nu este pozitiva!'*, altfel se trece la urmatorul pas.
Se calculeaza integrala de la - infinit la infinit a functiei f cu ajutorul functiei ***integrate*** si se returneaza inversa acestei valori.

Printscreen-uri cod:

Functia ***verif***

```r
#Cerinta 1: constanta de normalizare
#sursa teorie: https://en.wikipedia.org/wiki/Normalizing_constant

#'verifica' daca functia este pozitiva pentru valori intre -10^35 si 10^35 + calculeaza limitele la inf si -inf si verifiva daca acestea sunt negative
verif <- function(f){
  min <- optimise(f, interval = c(-100000000000000000000000000000000000,100000000000000000000000000000000000))$objective
  l1 <- f(-Inf)
  l2 <- f(Inf)
  if(min < 0 || l1 < 0 || l2 < 0)
  {
    return(0)
  }
  return(1)
}
```

Functia ***getNConst***

```r
getNConst <- function(f){
  tryCatch(
  {
    ok <- verif(f)
    if(ok == 0)
    {
      print("Functia nu este pozitiva!")
      return()
    }
    val <- integrate(f,-Inf,Inf)$value

    if(val <= 0)
    {
      print("Nu exista constanta de normalizare pentru functia data!")
      return()
    }
    else{
      return(val^(-1))
    }
  },
  error=function(cond){
    print("Nu exista constanta de normalizare pentru functia data!")
    print(cond)
    return()
  }
  )

}
```

Cerinta2:

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

Rezolvare:

Functia ***isDensity*** verifica primeste o functie f ca parametru si returneaza 0 daca functia nu este densitate de probabilitate si 1 in caz contrar. Cu ajutorul functiei ***verif*** de mai sus se verifica daca functia este pozitiva. Daca functia este pozitiva se verifica daca integrala cu capetele -inf si inf a functiei este 1. In cazul unei erori se afiseaza mesajul erorii si se returneaza 0.

Functia ***isDensity***

```
isDensity <- function(f)
{
  tryCatch(
    {
      if(verif(f)==0) #se verifica daca functia este pozitiva cu ajutorul functiei verif
      {
        return(0)
      }
      else {
        i <- integrate(f,-Inf,Inf)$value
        if(i != 1) #se verifica daca integrala sa de la -inf la inf este 1
        {
          return(0)
        }
      }
      return(1) # se returneaza 1 daca functia f este densitate de probabilitate si 0 in caz contrar
    },
    error=function(cond){ #in cazul unei erori, se afiseaza mesajul eroarii, iar 'isDensity' returneaza 0
      print(cond)
      return(0)
    }
  )

}
```

Cerinta3:

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

Rezolvare:

Cu ajutorul lui ***setClass*** am creat clasa ***contRV*** care reprezinta tipul variabila aleatoare continua. Fiecare instanta a clasei ***contRV*** va avea in componenta doua functii: ***d***, densitatea de probabilitate, care va fi introdusa de utilizator si ***r***, functia de repartitie, care se va calcula plecand de la ***d***.

```
setClass("contRV",representation(d = "function", r = "function"))
```

Am folosit ***setValidity*** pentru a asigura faptul ca functia data ca parametru in momentul crearii obiectului de tip ***contRV*** este densitate de probabilitate. ***setValidity*** primeste ca parametri numele clasei *contRV* si o functie ***validRV*** ,care se foloseste de ***isDensity***, si care returneaza ***TRUE*** daca functia este densitate de probabilitate. Daca aceasta nu indeplineste conditiile necesare se va afisa mesajul *'Functia nu este densitate de probabilitate'*.

```
validRV <- function(object) #se verifica daca functia data este densitate de probabilitate
{
  if(isDensity(object@d) == 1)
  {
    TRUE
  }
  else{
    paste("Functia nu este densitate de probabilitate")
  }
}
setValidity("contRV", validRV)
```

Pentru a initializa functia de repartitie ***r*** cu ajutorul densitatii de probabilitate ***d*** definim o specializare a functiei generice ***initialize*** pentru clasa ***contRV***. Aceasta este apelata indirect prin intermediul lui ***new***.

```r
#se construieste repartitia r la initializare
setMethod("initialize", "contRV",
          function(.Object, ...){
            .Object <- callNextMethod()

            .Object@r <- makeRep(.Object@d)
            .Object
          })
```

Pentru construirea functii de repartitie plecand de la ***d*** am definit functia ***makeRep*** care primeste o functie si returneaza functia de repartitie, creata cu ajutorul lui ***integrate***.

```r
makeRep <- function(f) #functia primeste densitatea de probabilitate si returneaza repartitia
{
  return(function(x)
    {
      integrate(f,-Inf, x)$value
    })

}
```

Crearea unui obiect se va face cu ajutorul functiei ***makeVA***:

```r
#  @export
makeVA <- function(f){

  v1 <- new("contRV", d = f)

  return(v1)

}

v1 <- makeVA(test2)
```

Ionescu Cristian Andrei

Cerinta 4:

Reprezentarea grafică a densităţii şi a funcţiei de repartiţie pentru diferite valori ale parametrilor repartiţiei. În cazul în care funcţia de repartiţie nu este dată într-o formă explicită(ex. repartiţia normală) se acceptă reprezentarea grafică a unei aproximări a acesteia.

Rezolvare:

Cu ajutorul functiei plot(), putem reprezenta grafic densitatea si functia de repartitie.

Am reprezentat grafic pentru distributia binomiala, folosind functia dbinom pentru densitate, iar pbinom pentru functia de repartitie.

Functia ***printContRV*** genereaza reprezentarile grafice ale densitatii de probabilitate si functiei de repartitie pentru un obiect de tip variabila aleatoare continua , tip definit la punctul 3. Functia primeste ca parametru o o v a continua (*contRV*) si doi parametrii care vor reprezenta limita inferioara si superioara a secventei pe care se vor constui graficele. Pentru densitatea de probabilitate graficul se realizeaza printr-un apel al functiei ***plot()***. Pentru functia de repartitie, intrucat este construita cu ajutorul functiei ***integrate()***, care nu poate avea ca parametru pentru limita superioara o colectie, am construit o functie auxiliara ***aux()***. Aceasta functie primeste ca parametrii colectia si functia de repartitie, aplica pentru fiecare element din colectie functia de repartitie, salveaza intr-o alta colectie valorile obtinute, iar apoi returneaza acea colectie. Asadar, pentru reprezentarea grafica a functiei de repartitie vom apela ***plot()***, folosind functia ***aux()*** in locul functiei de repartitie.

```
printContRV <- function(RV, i, j){
  x <- seq(i,j,by = 0.005)
  plot(x, RV@d(x), main = "probability density function")
  aux <-function(f,x)
  {
    vec <- c()
    for (el in x) {
      vec <- append(vec,f(el))
    }
    return (vec)
  }

  plot(x, aux(RV@r,x), main = "cumulative distribution function")
}
```

Cerinta 5:

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

Rezolvare:

Pentru calculul momentelor initiale si centrate, am folosit functia **moment** .
Antetul functiei este:
`moment(x, order=1, center=FALSE, absolute=FALSE, na.rm=FALSE)`

Aceasta functie primeste ca parametrii un vector numeric ce contine valorile pentru care trebuie calculate momentele, ordinul momentului pentru care se face calculul, acesta fiind default 1, bool center care indica daca momentul este centrat sau initial (true daca este centrat, false altfel), absolute daca momentul este absolut, si na.rm care indica daca valorile NA trebuie sterse inainte de a porni calculul.

Atunci cand center si absolute sunt pastrate cu valoarea default (FALSE), atunci momentul este calculat simplu ca: `sum(x ^ order) / length(x)`

Pentru a abstractiza apelul functiei pentru toate cazurile disponibile (momente initiale/centrate, de la ord 1..4), am facut o functie momentsCompute, ce primeste ca parametru vectorul numeric, acesta folosind o structura repetitiva pentru a itera prin toate cele 4 ordine si pentru a calcula momentul centrat, cat si initial, cu acel ordin. In cazul in care momentul nu exista, se va afisa un mesaj de eroare catre utilizator.

Cerinta 6:
Calculul mediei şi dispersiei unei variabile aleatoare g(X), unde X are o repartiţie continuă cunoscută iar g este o funcţie continuă precizată de utilizator.

Rezolvare:
Pentru aceasta cerinta, am implementat functia dispersionCompute(x, g), ce primeste ca parametrii repartitia continua cunoscuta si functia continua precizata de utilizator.

```
dispersionCompute <- function(x, g)
{
    return (integrate(g, lower = 1, upper = Inf)$value)
}
```

Pentru calculul dispersiei, calculam integrala de la 1 la Infinit din g(X), functie transmisa de utilizator.
Pentru calculul mediei, folosim functia din R mean(x)

Orzață Andrei

Pentru cerinţele 10 şi 11 am folosit funcţia care reprezintă prima şi a doua depistare a unor particule radioactive de către un contor Geiger într-un laborator. Particulele apar în faţa contorului conform unui process Poisson cu rata $\lambda = 0.8$

$$f(x,y) = \begin{cases} 0.64e^{-0.8y} & 0 < x < y \\ 0 & \text{otherwise} \end{cases}.$$

```
#Cerinta 10:

#densitatea comuna a celor doua variabile
f <- function(x,y)
{
  if(0 < x && x<y)
    return ((0.64)*(1/exp((0.8)*y)))
  else
    return(0)
}
```

Originea exemplului: https://dlsun.github.io/probability/joint-continuous.html

Cerinta10:

Calculul covarianţei şi coeficientului de corelaţie pentru două variabile aleatoare continue**.**

Rezolvare:

Sursa:
https://en.wikipedia.org/wiki/Probability_density_function
https://en.wikipedia.org/wiki/Pearson_correlation_coefficient
https://en.wikipedia.org/wiki/Covariance
https://en.wikipedia.org/wiki/Expected_value
Curs 8 seria 23.pdf

Covariaţia este o măsură a cât de mult două variabile aleatoare variază împreună. Ce are ca definiţie formula de mai jos, unde X şi Y sunt două variabile aleatoare continue şi E reprezintă media.

$$\mathrm{Cov}[X,Y] = E[XY] - E[X]E[Y].$$

Aducem formula la o formă mai utilă.

$$\begin{aligned}\operatorname{cov}(X,Y) &= \mathrm{E}[(X-\mathrm{E}[X])\,(Y-\mathrm{E}[Y])] \\ &= \mathrm{E}[XY - X\,\mathrm{E}[Y] - \mathrm{E}[X]Y + \mathrm{E}[X]\,\mathrm{E}[Y]] \\ &= \mathrm{E}[XY] - \mathrm{E}[X]\,\mathrm{E}[Y] - \mathrm{E}[X]\,\mathrm{E}[Y] + \mathrm{E}[X]\,\mathrm{E}[Y] \\ &= \mathrm{E}[XY] - \mathrm{E}[X]\,\mathrm{E}[Y],\end{aligned}$$

Vom calcula **E[XY]**, **E[X]**, **E[Y]** dupa formula:

$$\mathrm{E}[X] = \int_{\mathbb{R}} x f(x)\, dx,$$

Astfel avem:

```
covVAC <- function(f)
{

  #Valoarea asteptată/Media fiecaror variabile in parte

  EXY <- integrate(function(y) { sapply(y, function(y) {
    integrate(function(x) x*y*f(x,y), 0, y)$value
  })
  }, 0, Inf)$value

  EX <- integrate(function(y) { sapply(y, function(y) {
    integrate(function(x) x*f(x,y), 0, y)$value
  })
  }, 0, Inf)$value

  EY <- integrate(function(z) { sapply(y, function(y) {
    integrate(function(x,y) y*f(x,y), 0, Inf)$value
  })
  }, 0, Inf)$value


  return (EXY-(EX*EY))
}
```

Unde vom folosi funcţia integrate() şi sapply() (care pastrează dimensiunea rezultatelor funcţiei f) pentru a crea Integralele duble.

Funcţia **covVAC()** preia în f densitatea comuna a variabilelor continue aleatoare X şi Y, calculând valorile aşteptate / mediile lui X, Y şi a combinaţiei dintre cele două şi întoarce conform formulei de mai sus, covariaţia.

Pentru Coeficientul de corelaţie între X şi Y avem următoarea formulă:

$$Cor(X,Y) = \rho = \frac{Cov(X,Y)}{\sigma_X \sigma_Y}.$$

În care elementul lipsă, deviaţia standard va fi calculată utilizând:

$$\sigma = \sqrt{\int_{\mathbf{X}} (x-\mu)^2\, p(x)\,\mathrm{d}x}, \text{ where } \mu = \int_{\mathbf{X}} x\, p(x)\,\mathrm{d}x,$$

```
cvVAC <- function(f)
{
  #Calculam deviatia standard pentru X si Y
  devX <- integrate(function(x) x*f(x,y), 0, y)$value
  devY <- integrate(function(x) y*f(x,y), 0, Inf)$value

  return (covVAC()/(devX*devY))
}
```

Funcţia **cvVAC()** preia în f densitatea comuna a variabilelor continue aleatoare X şi Y, calculează deviaţia fiecăruia şi o stochează în variabilele devX, devY şi conform formulei de mai sus, utilizând funcţia de covariaţie **covVAC(),** returnează valoare coeficientului de corelaţie.

Cerinta11:

Pornind de la densitatea comună a două variabile aleatoare continue, construirea densităţilor marginale şi a densităţilor condiţionate.

Rezolvare:

Sursa:
https://en.wikipedia.org/wiki/Conditional_probability_distribution
https://en.wikipedia.org/wiki/Marginal_distribution

Vom începe utilizând formulele de mai jos:

$$f_X(x) = \int_c^d f(x,y)dy, \text{ and } f_Y(y) = \int_a^b f(x,y)dx$$

where $x \in [a,b]$, and $y \in [c,d]$.

$$f_{Y|X}(y \mid x) = \frac{f_{X,Y}(x,y)}{f_X(x)}$$

acestea pentru densitaţile marginale şi acestea pentru cele condiţionate,

```
#Densitati Marginale
dMarX <- function(f)
{

  val <- integrate(function(x) f(x,y), 0, y)$value

  return (val)
}

dMarY <- function(f)
{

  val <- integrate(function(y) f(x,y), 0, Inf)$value

  return (val)
}
```

```
#Densitati Conditionale
dCondYX <- function(f)
{
  fxy <- integrate(function(y) { sapply(y, function(y) {
    integrate(function(x) x*y*f(x,y), 0, y)$value
  })
  }, 0, Inf)$value
  val <- (fxy/dMarX())

  return (val)

}

dCondXY <- function(f)
{
  fxy <- integrate(function(y) { sapply(y, function(y) {
    integrate(function(x) x*y*f(x,y), 0, y)$value
  })
  }, 0, Inf)$value
  val <- (fxy/dMarX())

  return (val)
}
```

observând legătura dintre ele.

Pentru densitatea conditionata am folosit pentru *f* X,Y formula de mai sus.

$$\Pr(X > 0, Y > 0) = \int_0^\infty \int_0^\infty f_{X,Y}(x,y)\, dx\, dy.$$

în funcţiile **dMarY()** şi **dMarX()** se preia în f densitatea comuna a variabilelor continue aleatoare X şi Y şi conform formulei descrise anterior se returneză valorile densitaţii marginală în variabila val.

În **dCondXY()**, **dCondYX(),** folosind funcţiile menţionate anterior şi media/ valoarea aşteptată a densităţii comune a variabilelor continue aleatoare X şi Y, se calculează conform formulei şi se returnează valorile densitaţii cond în val

Cerinta12:

Construirea sumei şi diferenţei a două variabile aleatoare continue independente folosind formula de convoluţie.

Rezolvare:

Sursa:
https://en.wikipedia.org/wiki/Convolution_of_probability_distributions
https://en.wikipedia.org/wiki/Sum_of_normally_distributed_random_variables
https://dlsun.github.io/probability/sums-continuous.html

Fie X şi Y variabile aleatoare continue independente, rezulta ca T (sau Z), densitatea sumei, lor este convolutia dintre celor două:

$$f_T(t) = \int_{-\infty}^{\infty} f_X(x) \cdot f_Y(t-x)\, dx.$$

De aici putem afla prin calcul matematic că suma şi diferenţa, asumând faptul că variabilele sunt normal distribuite, au ca şi $\mu_{X+Y} = \mu_x + \mu_y$, respectiv, $\mu_{X-Y} = \mu_x - \mu_y$, însă variaţia este la fel pentru amândouă, $\sigma^2_{X+Y} = \sigma^2_x + \sigma^2_y$, $\sigma^2_{X-Y} = \sigma^2_x + \sigma^2_y$.

(Plus) Explicaţie matematică:

$$f_X(x) = \mathcal{N}(x; \mu_X, \sigma_X^2) = \frac{1}{\sqrt{2\pi}\sigma_X} e^{-(x-\mu_X)^2/(2\sigma_X^2)}$$

$$f_Y(y) = \mathcal{N}(y; \mu_Y, \sigma_Y^2) = \frac{1}{\sqrt{2\pi}\sigma_Y} e^{-(y-\mu_Y)^2/(2\sigma_Y^2)}$$

$$\sigma_Z = \sqrt{\sigma_X^2 + \sigma_Y^2}$$

$$\begin{aligned}
f_Z(z) &= \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}\sigma_Y} \exp\left[-\frac{(z-x-\mu_Y)^2}{2\sigma_Y^2}\right] \frac{1}{\sqrt{2\pi}\sigma_X} \exp\left[-\frac{(x-\mu_X)^2}{2\sigma_X^2}\right] dx \\
&= \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}\sqrt{2\pi}\sigma_X\sigma_Y} \exp\left[-\frac{\sigma_X^2(z-x-\mu_Y)^2 + \sigma_Y^2(x-\mu_X)^2}{2\sigma_X^2\sigma_Y^2}\right] dx \\
&= \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}\sqrt{2\pi}\sigma_X\sigma_Y} \exp\left[-\frac{\sigma_X^2(z^2+x^2+\mu_Y^2-2xz-2z\mu_Y+2x\mu_Y) + \sigma_Y^2(x^2+\mu_X^2-2x\mu_X)}{2\sigma_Y^2\sigma_X^2}\right] dx \\
&= \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}\sqrt{2\pi}\sigma_X\sigma_Y} \exp\left[-\frac{x^2(\sigma_X^2+\sigma_Y^2) - 2x(\sigma_X^2(z-\mu_Y)+\sigma_Y^2\mu_X) + \sigma_X^2(z^2+\mu_Y^2-2z\mu_Y) + \sigma_Y^2\mu_X^2}{2\sigma_Y^2\sigma_X^2}\right] dx
\end{aligned}$$

$$f_Z(z) = \frac{1}{\sqrt{2\pi}\sigma_Z} \exp\left[-\frac{(z-(\mu_X+\mu_Y))^2}{2\sigma_Z^2}\right]$$

```r
    return(b)
  }

  funcZT <- function(t)
  {
    val <- integrate(function(x) funcX(x)*funcY(t-x), -Inf, Inf)$value
    return(val)

  }

  miuX <- integrate(funcX, -Inf, Inf)$value
  miuY <- integrate(funcX, -Inf, Inf)$value
  varX <- integrate(function(x) (x^2)*funcX(X),-Inf,Inf)$value - miux^2
  varY <-integrate(function(y)  y^2*funcy(y),-Inf,Inf)$value  - miuy^2

  print("SUMA")
  print(miux+miuY)
  print(varx+varY)

  print("DIFERENTA")
  print(miux-miuY)
  print(varx+varY)


}
```

Pentru calculu variaţiei am folosit formula:

$$\operatorname{Var}(X) = \int_{\mathbb{R}} x^2 f(x)\, dx - \mu^2,$$

Funcţia **SumDif()** preia densităţile a două variabile aleatoare continue X şi Y, în **funcX()** şi **funcY()**, şi folosind formulele matematice anterioare calculează densitatea sumei acestora în

funcZT(), precum variaţia şi media diferenţelor sumei şi diferenţe, urmând ca acestea să fie salvate în variabile **miux, miuy** şi **varx, vary** respectiv, urmând ca acestea să fie calculate pentru sumă şi diferenţă şi afişate în consolă cu ajutorul funcţiei print().