Facultatea de Matematică și Informatică, Universitatea din București

PROIECT PROBABILITĂȚI ȘI STATISTICĂ

Membrii Echipei (Grupa 241)

Lungu Laura-Vanesa

Negoiță-Crețu Raluca-Marina

Popa Jasmine

Zamfirescu Alexandra

1. Exercițiul 1

a. Funcția frepcomgen – Repartitia comună incompletă a lui X și Y

Ideea algoritmului nostru:

1. Generăm valorile pe care le pot lua variabilele noastre aleatoare X și Y (numere din intervalul 1-n, respectiv 1-m)
2. Generăm valorile din repartiția comună. Generăm o matrice de dimensiune n*m cu valori aleatoare uniform distribuite
3. Normalizăm valorile pe care le iau elementele repartiției comune pentru a ne asigura că suma acestora este 1
4. Calculăm valorile probabilităților pe care le au X și Y (i.e. valorile repartițiilor marginale)
5. Generăm 3 poziții aleatoare în matrice pe care să le completăm cu NaN (ștergem 3 valori din matrice pentru a avea o repartiție incompletă)
6. Concatenăm matricea cu valorile reparițiilor marginale (coloana probabilităților pi, respectiv linia probabilităților qj)
7. Pentru o afișare frumoasă, configurăm numele rândurilor și coloanelor din tabel
8. La final generăm tabelul matricei folosindu-ne de librăria gt

```
frepcomgen <- function(n, m) {
  # valorile lui X si Y
  xv <- 1:n
  yv <- 1:m

  # generăm valori din repartitia comuna
  xycomp <- matrix(runif(n * m), nrow = n, ncol = m)

  # facem normalizarea fiecărei linii pentru a ne asigura că suma fiecărei linii
este 1
  xycomp <- t(apply(xycomp, 1, function(row) row / sum(row)))

  # facem normalizarea globală pentru a ne asigura că suma totală a elementelor
este 1
  xycomp <- xycomp / sum(xycomp)

  # rotunjim elementele la două zecimale pentru ușurința calculelor
  xycomp <- round(xycomp, digits = 2)

  xycomp[1,1] <- xycomp[1,1] +(1-sum(xycomp))
  # calculăm sumele pe linii si coloane
  pi <- rowSums(xycomp, na.rm = TRUE)
  qj <- colSums(xycomp, na.rm = TRUE)
```

```r
  # generam 3 pozitii random ale unor elemente din matrice pe care sa le stergem
din repartitia comuna
  random_col <- sample(c(1,m), 3, replace=TRUE)
  random_row <- sample(c(1,n), 3, replace=TRUE)
  xycomp[random_row[1], random_col[1]] <- NaN
  xycomp[random_row[2], random_col[2]] <- NaN
  xycomp[random_row[3], random_col[3]] <- NaN

  # adăugăm probabilitățile lui X si Y (ultima coloana / linie)
  xycomp <- rbind(cbind(xycomp, pi), c(qj, 1))

  xycomp <- cbind(c(as.character(xv), "∑qj"), xycomp)
  rownames(xycomp) <- NULL
  colnames(xycomp) <- c("X\\Y", as.character(yv), "∑pi")

  # transformăm matricea in data.frame
  xycomp <- as.data.frame(xycomp)

  # tabelul de repartitie comuna cu gt
  tbl <- gt(xycomp) %>%
    tab_header(title = "Repartitia Comuna a v.a. X si Y (incompleta)") %>%
    sub_missing(columns = everything(), missing_text = " ") %>%
    cols_align(align = "left")

  return(list(tbl = tbl, xycomp = xycomp))
}

# Exemplu de utilizare
result <- frepcomgen(4,5)

# Extragem tabelul si matricea din rezultatul functiei:
table_result <- result$tbl
matrice_xycomp <- result$xycomp

# Afisam rezultatul
print(table_result)
print(matrice_xycomp)
View(matrice_xycomp)
```

Repartitia Comuna a v.a. X si Y (incompleta)						
X\Y	1	2	3	4	5	∑pi
1	NaN	0.02	0.06	0.07	NaN	0.27
2	0.08	0.04	0.03	0.06	0.03	0.24
3	0.07	0.03	0.03	0.07	0.04	0.24
4	0.01	0.06	0.06	0.05	NaN	0.25
∑qj	0.2	0.15	0.18	0.25	0.22	1

b. Funcția fcomplrepcom – Completarea Repartiției Comune a lui X și Y

Ideea algoritmului nostru:

1. Prima dată parcurgem pe rând liniile din matrice
2. Dacă găsim un element pe linie care are valoare NaN îl numărăm și salvăm indicele coloanei pe care l-am găsit
3. Dacă terminăm de parcurs o linie și pe acea linie am găsit o singură valoare NaN înseamnă că putem calcula valoarea lipsă. Dacă sunt mai multe valori NaN pe o linie atunci nu putem calcula valorile corespunzătoare de pe pozițiile pe care găsim NaN
4. Dacă găsim un singur element apelăm funcția auxiliar completare care calculează valoarea lipsă
5. Procedăm analog și în cazul coloanelor
6. Parcurgem pe rând coloanele din matrice
7. Dacă găsim un element pe coloană care are valoare Nan îl numărăm și salvăm indicele liniei pe care l-am găsit
8. Dacă terminăm de parcurs o coloană și pe acea coloană am găsit o singură valoare NaN înseamnă că putem calcula valoarea lipsă. Dacă sunt mai multe valori NaN pe o coloană atunci nu putem calcula valorile corespunzătoare de pe pozițiile pe care găsim NaN.
9. Dacă găsim un singur element apelăm funcția auxiliar completare care calculează valoarea lipsă.
10. La final returnăm matricea care conține repartiția comună completă a variabilelor aleatoare X și Y.

```r
#FUNCTIE AUXILIAR CARE CALCULEAZA VALOAREA CE TREBUIE COMPLETATA PE O ANUMITA
POZITIE
completare <- function(mtx, indice_linie, indice_coloana, tip_completare){
  n <- nrow(mtx) #extragem nr de linii
  m <- ncol(mtx) #extragem nr de coloane
  suma_pi <- as.numeric(mtx[indice_linie,m]) #extragem valoarea probabilității
marginale pe linia dată
  suma_qi <- as.numeric(mtx[n,indice_coloana]) #extragem valoarea probabilității
marginale pe coloana dată
  #Daca tip_completare este 1 înseamnă că vom face completarea la nivel de linie
  if(tip_completare==1){
    for(j in 2:(m-1)){
      if(indice_coloana!=j){
        #calculăm valoarea lipsă prin scăderi repetate
        suma_pi <- suma_pi - as.numeric(mtx[indice_linie,j])
      }
    }
    mtx[indice_linie,indice_coloana] <- round(suma_pi,digits=2)
    #print(mtx)
  }

  #Dacă tip_completare este 0 înseamnă că vom face completarea la nivel de
coloană
  else{
    for(i in 1:(n-1)){
      if(indice_linie!=i){
        #calculăm valoarea lipsă prin scăderi repetate
        suma_qi <- suma_qi - as.numeric(mtx[i,indice_coloana])
      }
    }
    mtx[indice_linie,indice_coloana] <- round(suma_qi,digits=2)
    #print(mtx)
  }
  return (mtx)
}
#matrice_xycomp <- completare(matrice_xycomp, 4, 2, 0)

#- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

#FUNCȚIA PRINCIPALĂ CARE IDENTIFICĂ VALORILE LIPSĂ(NaN) PE CARE LE PUTEM COMPLETA
fcomplrepcom <- function(mtx) {
  n <- nrow(mtx) #extragem nr de randuri
  m <- ncol(mtx) #extragem nr de coloane

  #parcurgem liniile matricei
```

```
  for (i in 1:(n-1)) {

    #variabilă în care reținem numărul de valori NaN de pe linia i
    contor_linie <- 0

    for (j in 2:(m-1)) {
      if (is.nan(as.numeric(mtx[i, j]))) {
        #daca gasim o valoare NaN incrementăm contorul și salvăm coloana pe care
am găsit elementul
        contor_linie  <- contor_linie + 1
        indice_col <- j
      }
    }
    #dacă am găsit un singur element lipsă pe linia i înseamnă că putem face
completarea
    if(contor_linie == 1)
      #completăm valoarea corespunzătoare elementului mtx[i][indice_col]
      #apelăm o funcție auxiliar care face completarea
      mtx <- completare(mtx,i,indice_col,1)
  }

  #parcurgem coloanele matricei
  for(j in 2:(m-1)){
    #variabilă în care reținem numărul de valori NaN de pe coloana j
    contor_coloana <- 0
    for(i in 1: (n-1)){
      if (is.nan(as.numeric(mtx[i, j]))) {
        #daca gasim o valoare NaN incrementăm contorul și salvăm linia pe care am
găsit elementul
        contor_coloana  <- contor_coloana + 1
        indice_lin <- i
      }
    }

    #dacă am găsit un singur element lipsă pe coloana j înseamnă că putem face
completarea
    if(contor_coloana==1){
      #completăm valoarea corespunzătoare elementului mtx[indice_lin][j]
      #apelăm o funcție auxiliar care face completarea
      mtx <- completare(mtx,indice_lin,j,0)
    }
  }
  return (mtx)
}
```

```
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

#Exemplu de utilizare
matrice_xycomp <- fcomplrepcom(matrice_xycomp)
print(matrice_xycomp)
View(matrice_xycomp)

#Afișăm frumos tabelul
df <- as.data.frame(matrice_xycomp)
gt(df)
```

X\Y	1	2	3	4	5	∑pi
1	0.04	0.02	0.06	0.07	0.08	0.27
2	0.08	0.04	0.03	0.06	0.03	0.24
3	0.07	0.03	0.03	0.07	0.04	0.24
4	0.01	0.06	0.06	0.05	0.07	0.25
∑qj	0.2	0.15	0.18	0.25	0.22	1

c. Funcția frepmarginal – Repartițiile marginale pentru X Și Y

Ideea algoritmului nostru:

1. Pentru repartiția lui X formăm o nouă matrice cu 2 linii corespunzătoare valorilor care se regăsesc pe prima și pe ultima coloană
2. Pentru repartiția lui Y procedăm analog, cu mențiunea că vom considera primul și ultimul rând din repartiția comună

```r
frepmarginal <- function(mtx) {
  n <- nrow(mtx)  # extragem nr de linii
  m <- ncol(mtx)  # extragem nr de coloane

  # construim repartitia marginala a lui X
  matrice_X <- rbind(format(as.numeric(mtx[, 1]),nsmall=0), as.numeric(mtx[, m]))

  #mtx[,1] - selecteaza toate elementele din prima coloana a matricei mtx
  #mtx[,m] - selecteaza toate elementele din coloana m a matricei mtx (cea care
conține valorile probabilităților)
  #rbind(mtx[,1],mtx[,m]) - uneste cei doi vectori selectati intr-o matrice noua
formata din 2 linii corespunzatoare vectorilor
  #folosim funcția format pentru a nu ne afișa numărul cu 2 zecimale

  matrice_X <- matrice_X[, -ncol(matrice_X)]
#eliminam ultima coloana din matricea rezultat pentru a nu prelua si etichetele
pe care le definisem initial matricii

  # construim repartitia marginala a lui Y
  elementele_Y <- 0:(m - 1) #valorile pe care le ia variabila aleatoare Y
  #procedam analog cu repartitia lui X
  matrice_Y <- rbind(format(as.numeric(elementele_Y),nsmall=0), as.numeric(mtx[n,
-m]))

  # eliminăm prima coloană (etichetele) și ultima coloană (suma) din matricea Y
  matrice_Y <- matrice_Y[, -c(1, ncol(matrice_Y))]

  # convertim matricile într-un data frame pentru a putea lucra cu librăria gt
  tabel_X <- as.data.frame(matrice_X)
  tabel_Y <- as.data.frame(matrice_Y)


  # utilizam functia gt() pentru a crea tabelul
  tb1 <- gt(tabel_X) %>%
    tab_header(title = "Repartitia marginala a lui X") %>%
    cols_align(align = "left")

  tb2 <- gt(tabel_Y) %>%
    tab_header(title = "Repartitia marginala a lui Y") %>%
    cols_align(align = "left")

  return(list(tb1 = tb1, tb2 = tb2, matrice_X = matrice_X, matrice_Y =
matrice_Y))
}
```

```r
#Exemplu de utilizare
rezultat <- frepmarginal(matrice_xycomp)
rep_X <- rezultat$matrice_X
rep_Y <- rezultat$matrice_Y
print(rep_X)
print(rep_Y)
print(rezultat$tb1)
print(rezultat$tb2)
```

Repartiția lui X și a lui Y

```
> print(rep_X)
     [,1]   [,2]   [,3]   [,4]
[1,] " 1"    " 2"    " 3"    " 4"
[2,] "0.27" "0.24" "0.24" "0.25"
> print(rep_Y)
     [,1]  [,2]   [,3]   [,4]   [,5]
[1,] "1"   "2"    "3"    "4"    "5"
[2,] "0.2" "0.15" "0.18" "0.25" "0.22"
> print(rezultat$tb1)
```

d. Funcția fpropcov – Calculul Covarianței

```r
# Z <- aX + bY
# T <- cX + dY
# avem de calculat:
# Cov (Z, T) = Cov(aX + bY, cX + dY) = acVar(X) + (ad+bc)Cov(X,Y) + bdVar(Y)


#FUNCTII AUXILIAR
#calculam E[X]
calculMedie<-function(mtx)
{
  medie <-0
  m <-ncol(mtx)
  for (i in 1:m)
  {
    medie <- medie + as.numeric(mtx[1,i])*as.numeric(mtx[2,i])
  }
  return(medie)
}
```

```r
#calculam E[Y]
calcul_Medie_XY <- function(mtx){
  n <- nrow(mtx)
  m <- ncol(mtx)
  suma <- 0
  for(i in 1:(n-1)){
    for (j in 2:(m-1)){
      suma <- suma + i * (j-1) * as.numeric(mtx[i,j])
    }
  }
  return(suma)
}


#calculam cov(X,Y)
calcul_Covarianta <- function(m_X,m_Y,m_XY){
  covarianta <- m_XY - m_X*m_Y
  return(covarianta)
}

#calculam Var(X)
calcul_Varianta  <- function(mtx){
  #Var(X)=E[x^2]-E[x]^2
  mtx_patrat <- mtx
  mtx_patrat[1,] <- as.numeric(mtx[1,])^2
  var <- calculMedie(mtx_patrat)-calculMedie(mtx)^2
  return(var)
}

#FUNCTIA PRINCIPALA
fpropcov <- function(a, b, c, d, X, Y, mtx) {

  var_X <- calcul_Varianta(X)
  var_Y <- calcul_Varianta(Y)
  medie_XY <- calcul_Medie_XY(mtx)
  medie_X <- calculMedie(X)
  medie_Y <- calculMedie(Y)
  cov_X_Y <- calcul_Covarianta(medie_X, medie_Y, medie_XY)

  # aplicam proprietatea covariantei
  cov_Z_T <- a*c*var_X + (a*d + b*c)*cov_X_Y + b*d*var_Y

  return(cov_Z_T)
}
```

```r
#Exemple de utilizare
medie_X <- calculMedie(rep_X)
print(medie_X)

medie_Y <- calculMedie(rep_Y)
print(medie_Y)

medie_XY <- calcul_Medie_XY(matrice_xycomp)
print(medie_XY)

cov <- calcul_Covarianta(medie_X,medie_Y,medie_XY)
print(cov)

print(calcul_Varianta(rep_X))
print(calcul_Varianta(rep_Y))
print(fpropcov(1,3,2,4,rep_X, rep_Y,matrice_xycomp))
```

e. Funcția fPcond – Calculul Probabilității condiționate

```r
# P(X=x|Y=y) = P(X=x,Y=y)/P(Y=y)
# functia pentru P(X | Y = y)
fPcond_X_de_Y <- function(mtx, x, y) {
  # probabilitatea comuna P(X = xi, Y = yj)
  p_xy <- as.numeric(mtx[x, y+1])

  # probabilitatea P(Y = y)
  p_y <- as.numeric(mtx[nrow(mtx), y+1])

  # calculam probabilitatea cond P(X | Y = y)
  p_cond <- p_xy / p_y

  return(p_cond)
}

# P(Y=y | X=x) = P(X=x,Y=y)/P(X=x)
# functia pentru P(Y | X = x)
fPcond_Y_de_X <- function(mtx, x, y) {

  p_xy <- as.numeric(mtx[x, y+1])
  print(p_xy)

  # probabilitatea P(X = x)
  p_x <- as.numeric(mtx[x,ncol(mtx)])
```

```
  print(p_x)

  # calculam probabilitatea cond P(Y | X = x) = p_xy / p_x
  p_cond <- p_xy / p_x

  return(p_cond)
}

# functia principala fPcond
fPcond <- function(mtx, x, y, cond = "x|y") {
  if (cond == "x|y") {
    return(fPcond_X_de_Y(mtx, x, y))
  } else if (cond == "y|x") {
    return(fPcond_Y_de_X(mtx, x, y))
  } else {
    stop("Argumentul 'cond' trebuie sa fie 'x|y' sau 'y|x'")
  }
}

fPcond(matrice_xycomp, 1, 2, cond= "x|y")
fPcond(matrice_xycomp, 2, 1, cond= "y|x")
```

f. Funcția fPcomun – Calculul unei probabilități legate de perechea (X,Y)

```
# P(a≤X≤b,c≤Y≤d)
fPcomun <- function(a,b,c,d,mtx){
  suma <- 0
  for(i in a:b){
    for(j in (c+1):(d+1)){
      suma <- suma+ as.numeric(mtx[i,j])
    }
  }
  return(suma)
}
fPcomun(1,2,3,5,matrice_xycomp)
```

g. Calcul de probabilități

```
#1.Cov(5X+9,-3Y-2) = Cov(5X,-3Y) + Cov(5X, -2) + Cov(9,-2) + Cov(9,-3Y)
                # = Cov(5X,-3Y)
                # = Cov(5X+0Y , 0X-3Y)
                # = Cov(Z,T) - aplicam functia de la subpunctul d)
print(fpropcov(5,0,0,-3,rep_X,rep_Y,matrice_xycomp))

#2. P(0<X<0.8|Y>0.3) = P(0<X<0.8 , Y>0.3)/P(Y>0.3)

calcul_g.2 <- function (mtx){
  suma <- 0
  suma_y <- 0
  for(i in 1:(nrow(mtx)-1)){
    for(j in 2:(ncol(mtx)-1)){
      #dintr-o eroare de logistica nu putem accesa indicii valorilor lui Y
deoarece este un rand care nu face efectiv parte din matrice
      #Oups...
      if(mtx[i,1]>0 && mtx[i,1]<0.8 && (j-1) > 0.3 ){
        suma <- suma + as.numeric(mtx[i,j])
      }

      if((j-1)>0.3){
        suma_y <- suma_y + as.numeric(mtx[nrow(mtx),j])
      }
    }
  }
  if(suma_y!=0){
    rezultat <- suma/suma_y
  }
  else{
    rezultat <- 0
  }
  return(rezultat)
}
print(calcul_g.2(matrice_xycomp))
```

```r
#3. P(X>0.2,Y<1.7)
calcul_g.3 <- function(mtx){
  suma <- 0
  for(i in 1:(nrow(mtx)-1)){
    for(j in 2:(ncol(mtx)-1)){
      if(mtx[i,1]>0.2 && (j-1) < 1.7 ){
        suma <- suma + as.numeric(mtx[i,j])
      }

      }
    }
  return(suma)
}
print(calcul_g.3(matrice_xycomp))
```

h. Funcțiile fverind și fvernecor - independență și corelație

```r
#x si Y independente => P(X=x, Y=y) = P(X=x)*p(Y=y)
# P(X=x, Y=y) = matrice_xycomp[x,y+1]
# P(X=x) = matrice_xycomp[x, m+2]
# P(Y=y) = matrice_xycomp[n+1, y+1]
#parcurgem fiecare element matrice_xycomp[x,y+1] si verificam
#daca egalitatea are loc pt fiecare element
#facem asta cu o variabila bool ok=1, cand inegalitatea nu mai e indeplinita =>
ok=0
#Daca ok=1, X si Y sunt independente, daca ok=0, X si Y nu sunt independente


fverind <- function(mtx){
  n <- nrow(mtx)
  m <- ncol(mtx)

  ok <- 1
  for (i in 1:(n-1)) {
    for (j in 2:(m-1)) {
      produs <- as.numeric(mtx[i, m]) * as.numeric(mtx[n, j])

      if(isTRUE(all.equal(as.numeric(mtx[i,j]), produs))== FALSE)
      {
        ok <- 0
      }
    }
  }
  return(ok)
}
```

```r
#Verificam pentru o repartitie generata, care cel mai probabil NU are X si Y
independente
verifindep <- fverind(matrice_xycomp)
if (verifindep == 0) {
  print("X și Y nu sunt independente")
} else {
  print("X și Y sunt independente")
}
```

REZULTATUL PENTRU REPARTITIA AFISATA LA SUBPUNCTELE ANTERIOARE

```
[1] "X și Y nu sunt independente"
```

```r
#Verificam si pentru o repartitie data de noi, special ca sa fie independenta

result <- frepcomgen(2,2)
matrice_xycomp <- result$xycomp
View(matrice_xycomp)
matrice_xycomp <- fcomplrepcom(matrice_xycomp)
View(matrice_xycomp)


#transformam repartitia generata random astfel incat sa fie independenta
matrice_xycomp[1,4] <- 0.6
matrice_xycomp[2,4] <- 0.4
matrice_xycomp[3,2] <- 0.7
matrice_xycomp[3,3] <- 0.3
matrice_xycomp[1,2] <- 0.42
matrice_xycomp[2,2] <- 0.28
matrice_xycomp[1,3] <- 0.18
matrice_xycomp[2,3] <- 0.12
```

REPARTIȚIA CU DOUĂ VARIABILE ALEATOARE INDEPENDENTE

Filter

	X\Y	1	2	∑pi
1	1	0.42	0.18	0.6
2	2	0.28	0.12	0.4
3	∑qj	0.7	0.3	1

```r
verifindep <- fverind(matrice_xycomp)
if (verifindep == 0) {
  print("X și Y nu sunt independente")
} else {
  print("X și Y sunt independente")
}
```

REZULTATUL PENTRU NOUA REPARTIȚIE

```
[1] "X și Y sunt independente"
```

```r
fnercor <- function(mtx){

  rezultat <- frepmarginal(mtx)
  rep_X <- rezultat$matrice_X
  rep_Y <- rezultat$matrice_Y
  #print(rep_X)
  #print(rep_Y)
  medie_X <- calculMedie(rep_X)
  medie_Y <- calculMedie(rep_Y)
  medie_XY <- calcul_Medie_XY(matrice_xycomp)
  #print(medie_X)
  #print(medie_Y)
  #print(medie_XY)
  cov_XY <- 0
  cov_XY <- calcul_Covarianta(medie_X,medie_Y,medie_XY)
  #print(cov_XY)
  #print(round(as.numeric(cov_XY),digits=2))
  if(round(as.numeric(cov_XY),digits=2) == 0){
    print("X si Y sunt necorelate")
  } else{
    print("X si Y sunt corelate")
  }
}
fnercor(matrice_xycomp)
```

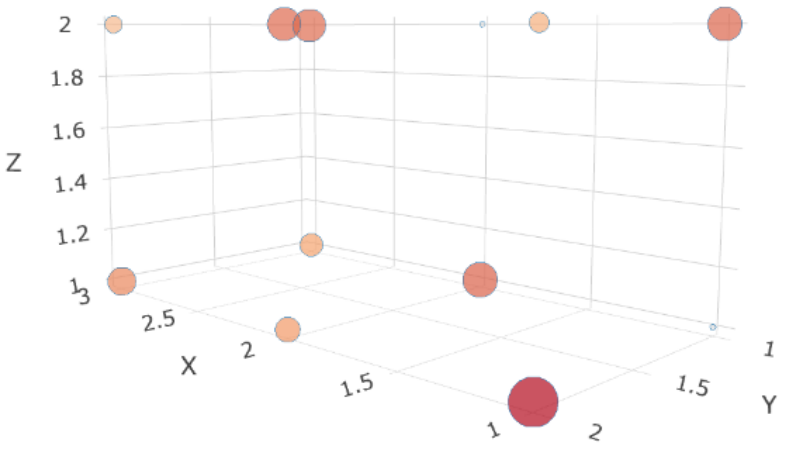
i. Repartiția comună a 3 varibile aleatoare

```
#Reprezentare 3D a repartitiei comune a  3 variabile aleatoare discrete (X, Y, Z)
#si obtine repartitiile lor marginale
#Cu ajutorul librariei plotly vom construi reprezentarea vizuala
#Cu ajutorul librariei dplyr vom construi repartitiile marginale
#
#Pe axa x este valoarea variabilei aleatoare X
#Pe axa y este valoarea variabilei aleatoare Y
#Pe axa z este valoarea variabilei aleatoare Z.
#Probabilitatea din repartitia comuna a celor 3 variabile este, de fapt,
#un punct(bulina) din interiorul reprezentarii vizuale, si difera ca dimensiune
#in functie de valoarea ei (dimensiunea unui punct
#e direct proportionala cu valoarea probabilitatii).

install.packages('plotly')
install.packages('dplyr')

library(plotly)
library(dplyr)

#pentru exemplificare vom lua valori mici pentru a putea observa punctele
rezultate
n <- 3  # Numar de etichete din X
m <- 2  # Numar de etichete din Y
k <- 2  # Numar de etichete din Z

# Generarea celor 3 variabile aleatoare cu probabilitatea in fiecare punct
#(Generarea repartitiei comune sub forma: "eticheta X", "eticheta Y", "eticheta
Z", probabilitate)
rep_comuna <- expand.grid(X = 1:n, Y = 1:m, Z = 1:k) %>%
  mutate(probabilitate = runif(n * m * k))

# Normalizarea probabilitatilor astfel incat suma sa fie 1
rep_comuna$probabilitate <- rep_comuna$probabilitate /
sum(rep_comuna$probabilitate)
View(rep_comuna)
# Verificarea sumei probabilitatilor
#suma_probabilitatilor <- sum(rep_comuna$probabilitate)
#print("Suma probabilitatilor:")
#print(suma_probabilitatilor)

plot_ly(rep_comuna, x = ~X, y = ~Y, z = ~Z, type = 'scatter3d', mode = 'markers',
        marker = list(size = ~probabilitate*200, color = ~probabilitate,
colorscale = 'Reds')) %>%
  layout(title = 'Reprezentarea vizuala a repartitiei comune X, Y, Z',
```

```
         scene = list(xaxis = list(title = 'X'),
                      yaxis = list(title = 'Y'),
                      zaxis = list(title = 'Z')))

#Construim repartitiile marginale ale celor 3 variabile aleatoare discrete
rep_X <- aggregate(probabilitate ~ X, data = rep_comuna, FUN = sum)
rep_Y <- aggregate(probabilitate ~ Y, data = rep_comuna, FUN = sum)
rep_Z <- aggregate(probabilitate ~ Z, data = rep_comuna, FUN = sum)


print(rep_X)
View(rep_X)

print(rep_Y)
View(rep_Y)

print(rep_Z)
View(rep_Z)
```

```
> print(rep_X)
  X probabilitate
1 1     0.3552936
2 2     0.3187540
3 3     0.3259524
> View(rep_X)
>
> print(rep_Y)
  Y probabilitate
1 1     0.4388597
2 2     0.5611403
> View(rep_Y)
>
> print(rep_Z)
  Z probabilitate
1 1     0.5374278
2 2     0.4625722
```

Reprezentarea vizuala a repartitiei comune X, Y, Z



2.Exercițiul 2

a. Teorema lui Fubini

Pentru a verifica dacă peste o funcție dată f(x,y) cu domeniul [a,b]x[c,d] se poate aplica teorema lui Fubini, vom verifica dacă integrala funcției f(x, a) pe intervalul [a, b] și integrala funcției f(a, y) pe intervalul [c, d] sunt finite.

1. Funcția test_fubini verifică tocmai acest lucru și returnează TRUE sau FALSE în funcție de caz.
2. În momentul interogarii (if (test_fubini(f2, a, b, c, d))), dacă funcția test_fubini a returnat TRUE, vom calcula valoarea integralei duble, folosindu-ne de funcția predefinită integral2 din pachetul pracma.
3. Am testat programul pentru 2 funcții, f1 (TRUE) și f2 (FALSE).

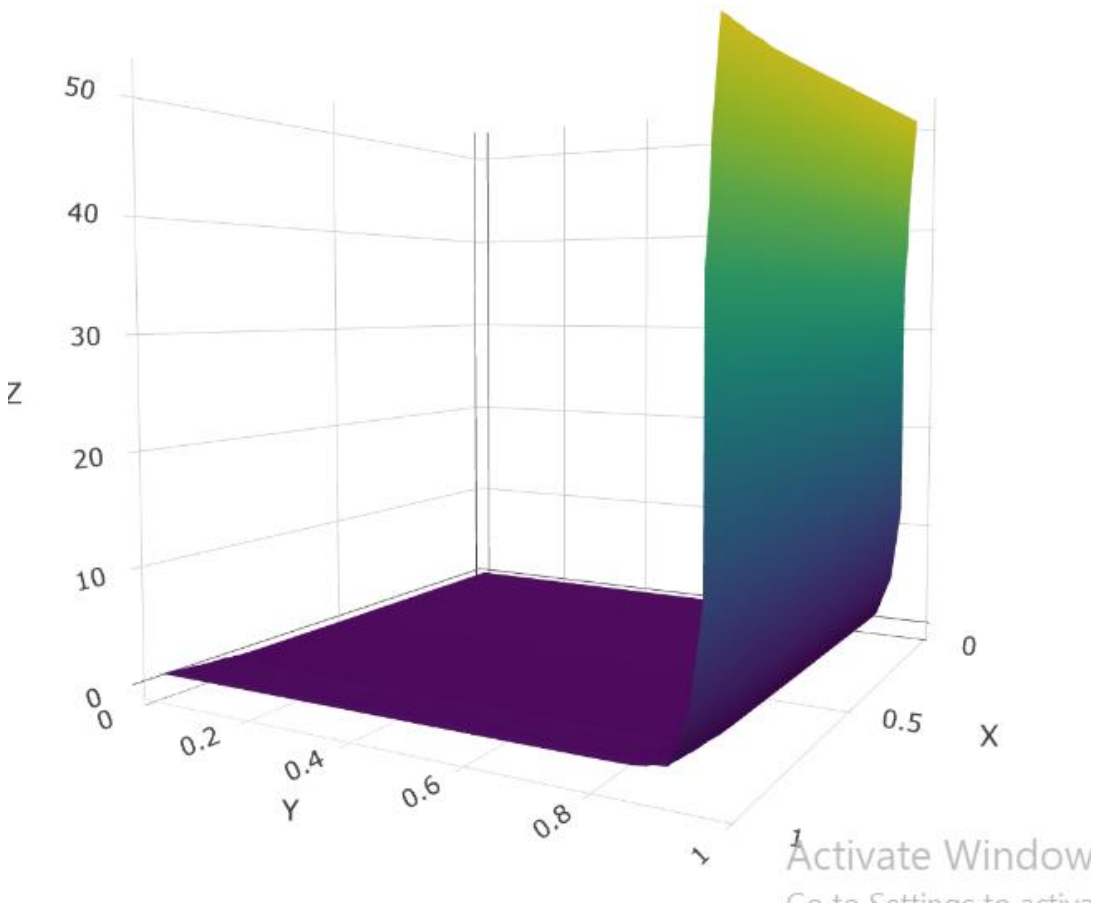
```r
# testam aplicabilitatea teoremei lui Fubini
test_fubini <- function(f, a, b, c, d) {
  if (!is.na(integrate(function(x) f(x, a), lower = a, upper = b)$value) && #daca
integrala primei functii e finita
      !is.na(integrate(function(y) f(a, y), lower = c, upper = d)$value)) { #daca
integrala functiei 2 e finita
    return(TRUE)  # teorema lui Fubini e aplicabila
  } else {
    return(FALSE) # teorema lu Fubini nu e aplicabila
  }
}


# limitele de integrare pentru x și y
a <- 0
b <- 1
c <- 0
d <- 1

# o functie buna
f1 <- function(x, y) {
  return(x^2+y^2)
}

# o functie (ne)buna
f2 <- function(x, y) {
  return(1/(x*y))
}


if (test_fubini(f2, a, b, c, d)) {
  cat("Teorema lui Fubini este aplicabilă pentru funcția dată.\n")

  library(pracma)
  I <- integral2(f2, 0, 1, 0, 1)
  I$Q

} else {
  cat("Teorema lui Fubini nu este aplicabilă pentru funcția dată.\n")
}
```

b. Reprezentarea geometrică

Pentru a face o interpretare geometrică a integralei duble pe un domeniu bidimensional, precum [0,1]x[0,1], putem folosi o reprezentare tridimensională a suprafeței definite de funcția dată pe acest domeniu. Astfel, mai întâi trebuie să generăm o serie de valori pentru parametrii funcției.

1. Vom inițializa variabila num_samples cu nr. de eșantioane aleatoare pe care le vom genera pt x si pt y.
2. Vom utiliza funcția predefinită runif() pentru a genera num_samples eșantioane aleatoare uniform distribuite între 0 și 1 pe axa x, respectiv pe axa y.
3. Vom utiliza funcția outer() pentru a aplica funcția bidimensională f(x, y) pe fiecare pereche de eșantioane aleatoare generate anterior.
4. Vom utiliza pachetul plotly pentru a crea un plot interactiv, în care x, y, z vor reprezenta axele dimensiunii tridimensionale.
5. Pentru a testa, am definit o functie f(x,y).

```
library(plotly)

# functia f
f <- function(x, y) {
  return(x^22 + y^8 + 76*x^55*x^5)
}

# nr de esantioane pt fiecare axa
num_samples <- 100

# generarea eșantioanelor aleatoare pentru axa x și y între 0 și 1
x <- sort(runif(num_samples, 0, 1))
y <- sort(runif(num_samples, 0, 1))

# calculul valorilor funcției f(x, y) pe baza eșantioanelor generate
z <- outer(x, y, f)

# crearea plotului
plot_ly(x = x, y = y, z = z, type = "surface") %>%
  layout(scene = list(
    xaxis = list(title = "X"),
    yaxis = list(title = "Y"),
    zaxis = list(title = "Z")
  ))
```



c. Verificare densitate de probabilitate

Pentru a verifica dacă o funcție dată este densitate de probabilitate, aceasta trebuie să îndeplinească 2 condiții:

1. Să fie pozitivă pe tot domeniul
2. Integrala dubla să fie egala cu 1

Astfel, construim un algoritm care verifica aceste 2 condiții.

1. Definim funcțiile de test (una care nu e pozitivă pe tot domeniul, una care e pozitivă, dar nu are integrala dublă egală cu 1 și una care e pozitivă și are și integrala dublă egala cu 1)
2. Creăm o secvență de la 0 la 1 cu un pas de 0.01 care o să reprezinte toate valorile pentru variabila x în [0,1]. Calculăm toate valorile posibile ale lui y din intervalul [0,1], utilizând funcția predefinită sapply(). Utilizăm all pentru a verifica dacă toate valorile calculate sunt >=0. Analog pentru o secventă care să reprezinte valorile lui y.

3. Dacă această primă condiție este îndeplinită, mergem mai departe și calculăm integrala dublă a funcției, folosind funcția predefinită integral2(). Dacă aceasta este egală cu 1, înseamnă că funcția noastră este densitate de probabilitate.

```
#functie care nu e pozitiva pe tot domeniul
f <- function(x, y) {
  return(-1)
}

#functie care e pozitiva pe tot domeniul si are integrala dubla = 1
f <- function(x, y) {
  return(4 * x * y)
}

#functie care e pozitiva pe tot domeniul, dar nu are integrala dubla = 1
f <- function(x, y) {
  return(x^2 + y^2)
}

# verificam daca f este pozitiva pe [0,1]x[0,1]
if(all(sapply(seq(0,1,0.01), function(x) all(f(x, seq(0,1,0.01)) >= 0))) &&
   all(sapply(seq(0,1,0.01), function(y) all(f(seq(0,1,0.01), y) >= 0)))) {
  print("Functia este pozitiva pe intervalul dat.")
} else {
  print("Functia nu este pozitiva pe intervalul dat.")
}


# calculam integrala dubla a functiei peste domeniul [0,1]x[0,1]
integral <- integral2(f, 0, 1, 0, 1)$Q

print(integral)

if(integral==1) {
  print("Functia este o densitate de probabilitate.")
} else {
  print("Functia nu este o densitate de probabilitate.")
}
```

d. Variabilă aleatoare continua

Vom crea o funcție care primește 2 parametri: o funcție care reprezintă densitatea de probabilitate dată de utilizator și un întreg care reprezintă dimensiunea introdusă de utilizator.

Dacă dimensiunea dată este 1, înseamnă că dorim să creăm un obiect de tip variabilă aleatoare unidimensionala, iar dacă dimensiunea este 2, înseamnă ca dorim să creăm un obiect de tip variabilă aleatoare bidimensională. Pentru orice altă valoare, funcția își oprește execuția.

Odată stabilit tipul variabilei aleatoare, creăm obiectele specifice.

Pentru variabilele unidimensionale, mai întâi generăm un set de valori (1000) cuprinse între -10 și 10. Aplicăm densitatea de probabilitate peste ele și utilizăm funcția predefinită sample() care alege aleatoriu valori din setul de valori ale lui x.

Pentru variabilele bidimensionale, utilizăm funcția rmvnorm din pachetul mvtnorm pentru a genera 1000 de variabile aleatoare bidimensionale dintr-o distribuție normală, specificând o medie de 0 pentru ambele variabile aleatoare și o matrice de covarianță 2x2 cu varianță de 1 pentru ambele variabile aleatoare și o covarianță de 0.5 între ele (pozitiv corelate).

În final, returnăm variabila aleatoare creată.

```
install.packages("mvtnorm")
library(mvtnorm)
```

```
creare_variabila_aleatoare <- function(densitate, dimensiune = 1) {
  # verific daca este vorba despre o v.a. unidim sau bidim
  if (is.function(densitate)) {
    if (dimensiune == 1) {
      # generam v.a. unidim folosind densitatea data
      valori_x <- seq(-10, 10, length.out = 1000)
      densitati <- densitate(valori_x)
      variabila_aleatoare <- sample(valori_x, size = 1000)
    } else if (dimensiune == 2) {
      # generam v.a. bidim folosind densitatea data
      variabila_aleatoare <- rmvnorm(n = 1000, mean = c(0, 0), sigma =
matrix(c(1, 0.5, 0.5, 1), nrow = 2))
    } else {
      stop("Dimensiunea specificată nu este validă.")
    }
  } else {
    stop("Densitatea de probabilitate nu este funcție.")
  }
```

```
  print(variabila_aleatoare)
  return(variabila_aleatoare)
}


# functia densitate de probabilitate pt o v.a. unidim
densitate_unidimensionala <- function(x) {
  return(1)
}

# crearea v.a. unidim folosind densitatea de probabilitate definită
variabila_aleatoare_unidimensionala <-
creare_variabila_aleatoare(densitate_unidimensionala, dimensiune = 1)

# functia densitate de probabilitate pt o v.a. bidim
densitate_bidimensionala <- function(x, y) {
  return(4 * x * y)
}

# crearea v.a. bidim folosind densitatea de probabilitate definită
variabile_aleatoare_bidimensionale <-
creare_variabila_aleatoare(densitate_bidimensionala, dimensiune = 2)
```