Laborator 3

Elemente de probabilități și simulare în R

Obiectivul acestui laborator este de a prezenta succint câteva funcții utile teoriei probabilităților din programul R, care este structura lor și cum le putem aplica. De asemenea, tot în acest laborator vom prezenta și câteva probleme de simulare.

1 Familia de funcții `apply`

Pe lângă buclele `for` și `while`, în R există și un set de funcții care permit scrierea și rularea într-o manieră mai compactă a codului dar și aplicarea de funcții unor grupuri de date.

- `lapply()`: Evaluează o funcție pentru fiecare element al unei liste
- `sapply()`: La fel ca `lapply` numai că încearcă să simplifice rezultatul
- `apply()`: Aplică o funcție după fiecare dimensiune a unui `array`
- `tapply()`: Aplică o funcție pe submulțimi ale unui vector
- `mapply()`: Varianta multivariată a funcției `lapply`
- `split`: Împarte un vector în grupuri definite de o variabilă de tip factor.

1.1 `lapply()`

Funcția `lapply()` efectuează următoarele operații:

1. buclează după o listă, iterând după fiecare element din acea listă
2. aplică o *funcție* fiecărui element al listei (o funcție pe care o specificăm)
3. întoarce ca rezultat tot o listă (prefixul `l` vine de la listă).

Această funcție primește următoarele trei argument: (1) o listă `X`; (2) o funcție `FUN`; (3) alte argumente via `...`. Dacă `X` nu este o listă atunci aceasta va fi transformată într-una folosind comanda `as.list()`.

Considerăm următorul exemplu în care vrem să aplicăm funcția `mean()` tuturor elementelor unei liste

```
set.seed(222)
x <- list(a = 1:5, b = rnorm(10), c = rnorm(20, 1), d = rnorm(100, 5))
lapply(x, mean)
$a
[1] 3

$b
[1] 0.1996044

$c
[1] 0.7881026

$d
[1] 5.064188
```

Putem să folosim funcția `lapply()` pentru a evalua o funcție în moduri repetate. Mai jos avem un exemplu în care folosim funcția `runif()` (permite generarea observațiilor uniform repartizate) de patru ori, de fiecare

dată generăm un număr diferit de valori aleatoare. Mai mult, argumentele $min = 0$ și $max = 3$ sunt atribuite, prin intermediul argumentului `...`, funcției `runif`.

```
x <- 1:4
lapply(x, runif, min = 0, max = 3)
[[1]]
[1] 0.03443616

[[2]]
[1] 1.267361 1.365441

[[3]]
[1] 1.8084700 2.1902665 0.4139585

[[4]]
[1] 1.5924650 0.7355067 2.1483841 1.6082945
```

1.2 `sapply()`

Funcția `sapply()` are un comportament similar cu `lapply()` prin faptul că funcția `sapply()` apelează intern `lapply()` pentru valorile de input, după care evaluează:

- dacă rezultatul este o listă în care fiecare element este de lungime 1, atunci întoarce un vector
- dacă rezultatul este o listă în care fiecare element este un vector de aceeași lungime (>1), se întoarce o matrice
- în caz contrar se întoarce o listă.

Considerăm exemplul de mai sus

```
set.seed(222)
x <- list(a = 1:4, b = rnorm(10), c = rnorm(20, 1), d = rnorm(100, 5))
sapply(x, mean)
        a         b         c         d
2.5000000 0.1996044 0.7881026 5.0641876
```

1.3 `split()`

Funcția `split()` primește ca argument un vector sau o listă (sau un data.frame) și împarte datele în grupuri determinate de o variabilă de tip factor (sau o listă de factor).

Argumentele aceste funcții sunt

```
str(split)
function (x, f, drop = FALSE, ...)
```

unde

- `x` este un vector, o listă sau un data.frame
- `f` este un factor sau o listă de factori

Considerăm următorul exemplu în care generăm un vector de date și îl împărțim după o variabilă de tip factor creată cu ajutorul funcției `gl()` (*generate levels*).

```
x <- c(rnorm(10), runif(10), rnorm(10, 1))
f <- gl(3, 10)
split(x, f)
```

```
$`1`
 [1] -2.27414224 -0.11266780  0.61308167  0.07733545  0.57137727
 [6]  0.11672493 -0.95685256 -1.90008460 -1.48972089  0.55925676

$`2`
 [1] 0.91159086 0.03291829 0.78368939 0.11852882 0.64443831 0.78790988
 [7] 0.82451477 0.05642366 0.65075027 0.95426854

$`3`
 [1]  2.6666242  2.6634334  1.8106280 -0.7837308  1.6575684  0.1546575
 [7]  0.4930056 -0.9031544  2.4042311  1.4106863
```

Putem folosi funcția `split` și în conjuncție cu funcția `lapply` (atunci când vrem să aplicăm o funcție `FUN` pe grupuri de date).

```
lapply(split(x, f), mean)
$`1`
[1] -0.4795692

$`2`
[1] 0.5765033

$`3`
[1] 1.157395
```

1.4 `tapply()`

Funcția `tapply()` este folosită pentru aplicarea unei funcții `FUN` pe submulțimile unui vector și poate fi văzută ca o combinație între `split()` și `sapply()`, dar doar pentru vectori.

```
str(tapply)
function (X, INDEX, FUN = NULL, ..., default = NA, simplify = TRUE)
```

Argumentele acestei funcții sunt date de următorul tabel:

Table 1: Argumentele functiei tapply

Argument	Descriere
`X`	un vector
`INDEX`	este o variabilă de tip factor sau o listă de factori
`FUN`	o funcție ce urmează să fie aplicată
`...`	argumente ce vor fi atribuite funcției `FUN`
`simplify`	dacă vrem să simplificăm rezultatul

Următorul exemplu calculează media după fiecare grupă determinată de o variabilă de tip factor a unui vector numeric.

```
x <- c(rnorm(10), runif(10), rnorm(10, 1))
f <- gl(3, 10)
f
 [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3
Levels: 1 2 3
tapply(x, f, mean)
```

```
           1            2            3
-0.0007774025  0.3736457792  0.5789436983
```

Putem să aplicăm și funcții care întorc mai mult de un rezultat. În această situație rezultatul nu poate fi simplificat:

```
tapply(x, f, range)
$`1`
[1] -2.1904113  0.9249901

$`2`
[1] 0.004445296 0.998309704

$`3`
[1] -0.3379675  1.9327099
```

1.5 `apply()`

Funcția `apply()` este folosită cu precădere pentru a aplica o funcție liniilor și coloanelor unei matrice (care este un `array` bidimensional). Cu toate acestea poate fi folosită pe tablouri multidimensionale (`array`) în general. Folosirea funcției `apply()` nu este mai rapidă decât scrierea unei bucle `for`, dar este mai compactă.

```
str(apply)
function (X, MARGIN, FUN, ...)
```

Argumentele funcției `apply()` sunt

- `X` un tablou multidimensional
- `MARGIN` este un vector numeric care indică dimensiunea sau dimensiunile după care se va aplica funcția
- `FUN` este o funcție ce urmează să fie aplicată
- `...` alte argumente penru funcția`FUN`

Considerăm următorul exemplu în care calculăm media pe coloane într-o matrice

```
x <- matrix(rnorm(200), 20, 10)
apply(x, 2, mean)  ## media fiecarei coloane
 [1]  3.745002e-02  1.857656e-01 -2.413659e-01 -2.093141e-01 -2.562272e-01
 [6]  8.986712e-05  7.444137e-02 -7.460941e-03  6.275282e-02  9.801550e-02
```

precum și media după fiecare linie

```
apply(x, 1, sum)   ## media fiecarei linii
 [1]  2.76179139  2.53107681  0.87923177  1.80480589  0.98225832
 [6] -3.06148753 -1.40358820 -0.65969812 -1.63717046 -0.29330726
[11] -2.41486442 -3.15698523  2.27126822 -3.88290287 -3.15595194
[16]  5.41211963  2.32985530 -3.05330574 -0.02110926 -1.34909559
```

2 Repartiții și elemente aleatoare în R

R pune la dispoziție majoritatea repartițiilor uzuale. Tabelul de mai jos prezintă numele și parametrii acestora:

Table 2: Numele si parametrii repartitiilor uzuale in R

Repartiția	Nume	Parametrii	Valori prestabilite
Beta	`beta`	`shape1`, `shape2`	
Binomial	`binom`	`size`, `prob`	
Cauchy	`cauchy`	`location`, `scale`	`location = 0`, `scale = 1`
Chi-Squared	`chisq`	`df`	
Exponential	`exp`	`rate` (=1/mean)	`rate = 1`
Fisher	`f`	`df1`, `df2`	
Gamma	`gamma`	`shape`, `rate` (=1/scale)	`rate = 1`
Hypergeometric	`hyper`	`m`, `n`, `k`	
Log-Normal	`lnorm`	`mean`, `sd`	`mean = 0`, `sd = 1`
Logistic	`logis`	`location`, `scale`	`location = 0`, `scale = 1`
Normal	`norm`	`mean`, `sd`	`mean = 0`, `sd = 1`
Poisson	`pois`	`lambda`	
Student	`t`	`df`	
Uniform	`unif`	`min`, `max`	`min = 0`, `max = 1`
Weibull	`weibull`	`shape`	

Pentru fiecare repartiție, există patru comenzi în R prefixate cu literele `d`, `p`, `q` și `r` și urmate de numele repartiției (coloana a 2-a). De exemplu `dnorm`, `pnorm`, `qnorm` și `rnorm` sunt comenzile corespunzătoare repartiției normale pe când `dunif`, `punif`, `qunif` și `runif` sunt cele corespunzătoare repartiției uniforme.

- `dname`: calculează densitatea atunci când vorbim de o variabilă continue sau funcția de masă atunci când avem o repartiție discretă ($\mathbb{P}(X = k)$)
- `pname`: calculează funcția de repartiție, i.e. $F(x) = \mathbb{P}(X \leq x)$
- `qname`: reprezintă funcția cuantilă, cu alte cuvinte valoarea pentru care funcția de repartiție are o anumită probabilitate; în cazul continuu, dacă `pname(x) = p` atunci `qname(p) = x` iar în cazul discret întoarce cel mai mic întreg u pentru care $\mathbb{P}(X \leq u) \geq p$.
- `rname`: generează observații independente din repartiția dată

Avem următoarele exemple:

```
qnorm(0.975)
[1] 1.959964
pnorm(1.96)
[1] 0.9750021
rnorm(5)
[1] 0.4304737 0.8405027 1.9550682 1.6208507 2.1059503

x = seq(-1, 1, 0.25)
dnorm(x)
[1] 0.2419707 0.3011374 0.3520653 0.3866681 0.3989423 0.3866681 0.3520653
[8] 0.3011374 0.2419707
rnorm(3, 5, 0.5)
[1] 5.327249 4.728878 5.773167

dunif(x)
[1] 0 0 0 0 1 1 1 1 1
runif(3)
[1] 0.6353840 0.8470974 0.0672359
```

3 Exerciții propuse

3.1 Aruncarea cu banul

În acest exemplu vrem să simulăm aruncarea unei monede (echilibrate) folosind funcția `sample()`. Această funcție permite extragerea, cu sau fără întoarcere (`replace = TRUE` sau `replace = FALSE` - aceasta este valoarea prestabilită), a unui eșantion de volum dat (`size`) dintr-o mulțime de elemente `x`.

Spre exemplu dacă vrem să simulăm 10 aruncări cu banul atunci apelăm:

```
sample(c("H", "T"), 10, replace = TRUE)
 [1] "T" "T" "T" "T" "T" "T" "T" "H" "H" "T"
```

Pentru a estima probabilitatea de apariției a stemei (`H`) repetăm aruncarea cu banul de 10000 de ori și calculăm raportul dintre numărul de apariții ale evenimentului $A = \{H\}$ și numărul total de aruncări:

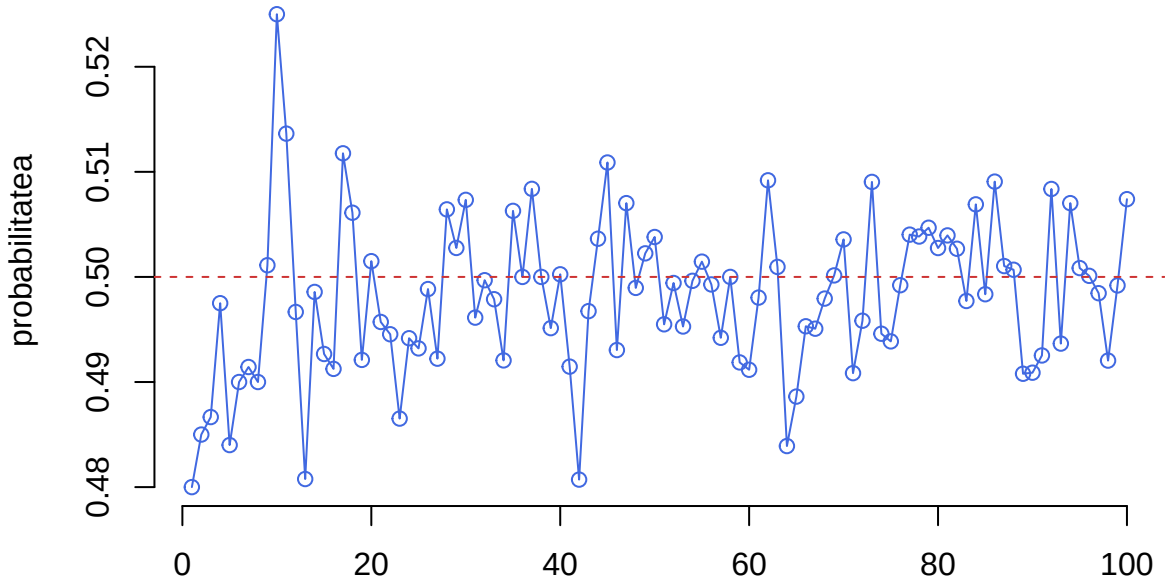
```
# atunci cand moneda este echilibrata
a = sample(c("H","T"), 10000, replace = TRUE)
p = sum(a == "H")/length(a)
p
[1] 0.5073
```

și pentru cazul în care moneda nu este echilibrată

```
a = sample(c("H","T"), 10000, replace = TRUE, prob = c(0.2, 0.8))
p = sum(a == "H")/length(a)
p
[1] 0.2012
```

Putem vedea cum evoluează această probabilitatea în funcție de numărul de repetări

```
y = rep(0,100)

for (i in 1:100){
  a = sample(c("H","T"), i*100, replace = TRUE)
  y[i] = sum(a == "H")/length(a)
}

plot(1:100, y, type = "o", col = "royalblue", bty = "n",
     xlab ="", ylab = "probabilitatea")
abline(h = 0.5, lty = 2, col = "brown3")
```



3.2 Jocul de loto

Construiți în R o funcție care să simuleze jocul de loto 6/49. Acest joc consistă din extragerea aleatoare a 6 numere dintr-o urnă cu 49 de numere posibile, fără întoarcere. Fiecare extragere se face de manieră uniformă din numerele rămase în urnă (la a i-a extragere fiecare bilă din urnă are aceeași șansă să fie extrasă). De exemplu putem avea următorul rezultat: $10, 27, 3, 45, 12, 24$.

Notă: Funcția `sample()` poate face această operație, ceea ce se cere este de a crea voi o funcție care să implementeze jocul fără a folosi funcția *sample*. Binențeles că puteți folosi funcții precum: `runif` , `floor`, `choose`, etc.

Începem prin a construi o funcție care ne permite generarea unei variabile aleatoare uniform repartizate pe mulțimea $\{1, 2, \ldots, n\}$ (această funcție este cea care simulează procesul de extragere de la fiecare pas):

```
myintunif = function(n){
  # dunctia care genereaza un numar uniform intre 1 si n
  r = n*runif(1)
  u = floor(r)+1
  return(u)
}
```

Funcția care realizează extragerea fără întoarcere a k numere aleatoare din n, este:

```
myrandsample=function(n,k){
  #
```

```
  x = 1:n
  q = rep(0,k)

  for(i in 1:k){
    l = length(x)
    u = myintunif(l)
    q[i] = x[u]
    x = x[x!=q[i]]
  }
  return(q)
}
```

Pentru a vedea ce face această funcție putem scrie:

```
n = 49
k = 6

myrandsample(n,k)
[1]  3 16 12 48 23 32
```

3.3 Generarea variabilelor aleatoare discrete

În acest exercițiu ne propunem să definim o funcție `rand_sample(n,x,p)` care permite generarea a n observații dintr-o mulțime x (vector numeric sau de caractere) cu probabilitatea p pe x (un vector de aceeași lungime ca x).

Funcția se poate construi sub forma următoare:

```
rand_sample = function(n,x,p){
  # n - numarul de observatii
  # x - multimea de valori
  # p - vectorul de probabilitati

  out = c()

  ind = 1:length(x)
  cs = cumsum(p)

  if (length(x)!=length(p)){
    return(print('Cei doi vectori ar trebui sa fie de aceeasi lungime !'))
  }

  for (i in 1:n){
    r = runif(1)

    m = min(ind[r<=cs])
    out = c(out,x[m])
  }

  return(out)
}
```

Pentru a testa această funcție să considerăm două exemple:

1. în acest caz: $n = 10$, $x = [1, 2, 3]$ și $p = [0.2, 0.3, 0.5]$

```
rand_sample(10,c(1,2,3),c(0.2,0.3,0.5))
 [1] 2 3 3 1 3 3 1 2 3 1
```

2. în acest caz: $n = 15$, $x = [a, b, c, d]$ și $p = [0.15, 0.35, 0.15, 0.45]$

```
rand_sample(15,c('a','b','c','d'),c(0.15,0.35,0.15,0.45))
 [1] "b" "d" "d" "d" "a" "b" "d" "a" "d" "c" "d" "d" "b" "b" "d"
```

O funcție un pic mai generală este:

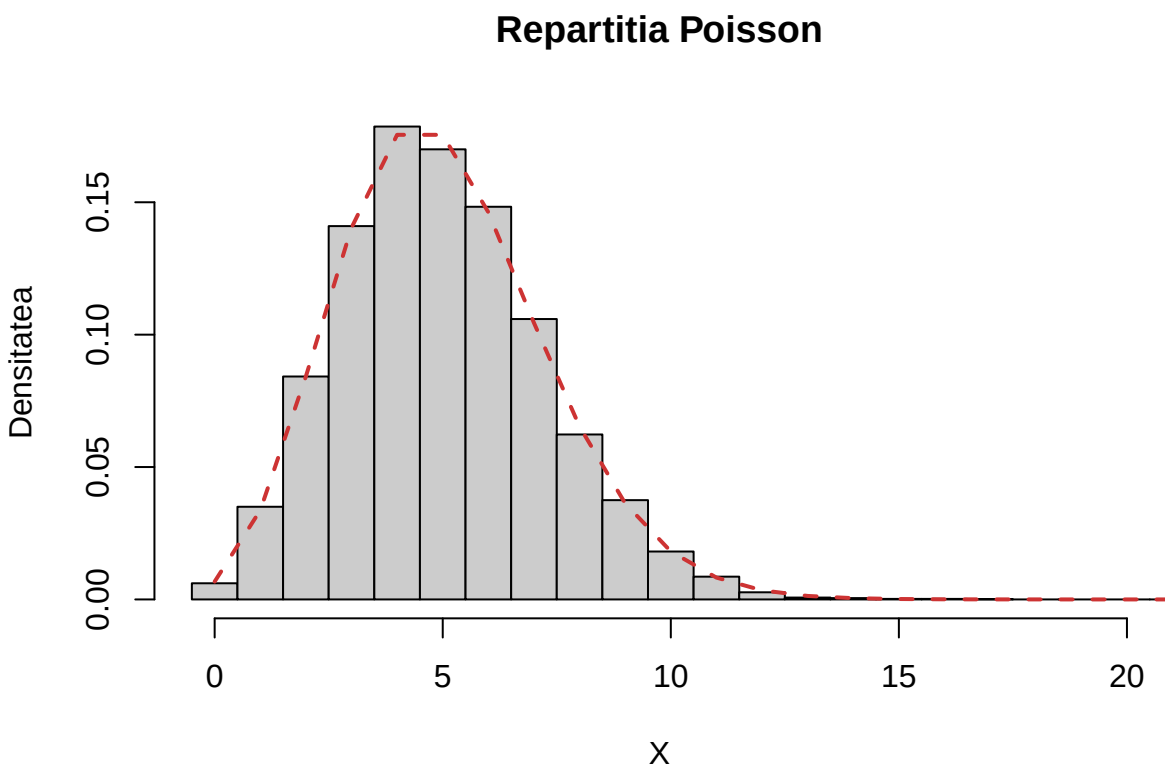
```
GenerateDiscrete = function(n = 1, x, p, err = 1e-15){
  # n numarul de observatii
  # x multimea de valori
  # p vectorul de probabilitati

  lp = length(p)
  lx = length(x)

  # verificarea conditiilor de aplicare
  if(abs(sum(p)-1)>err | sum(p>=0)!=lp){

    stop("Suma probabilitatilor nu este egala cu 1!")

  }else if(lx!=lp){

    stop("x si p trebuie sa aiba aceeasi marime!")

  }else{
    out = rep(0, n)

    indOrderProb = order(p, decreasing = TRUE) # index
    pOrdered = p[indOrderProb] # rearanjam valorile probabilitatilor
    xOrdered = x[indOrderProb] # rearanjam valorile lui x

    pOrderedCS = cumsum(pOrdered)

    for (i in 1:n){
      u = runif(1)

      k = min(which(u<=pOrderedCS))
      out[i] = xOrdered[k]
    }
  }

  return(out)
}
```

și pentru a o putea testa să considerăm cazul repartițiilor Poisson și Geometrică:

a) Poisson

```
# Poisson
hist(GenerateDiscrete(10000, x = 0:50,
                      p = dpois(0:50, 5)),
     probability = TRUE,
```

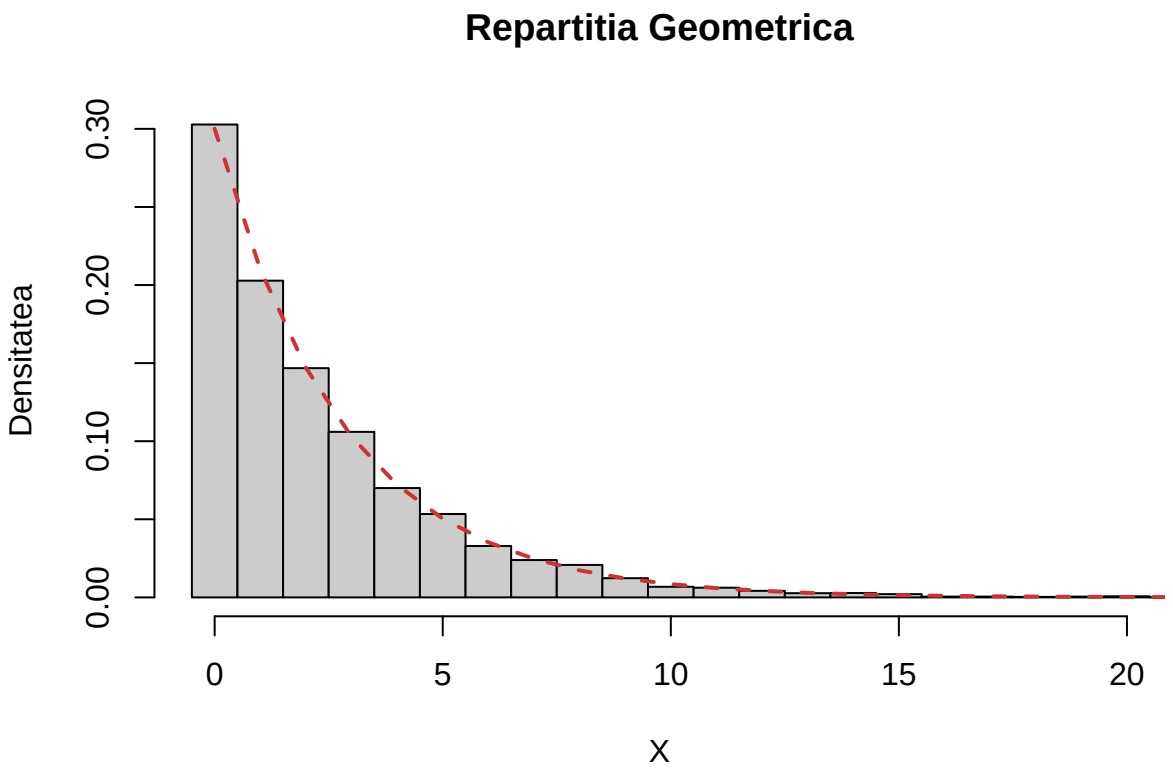
```
     breaks = seq(-0.5,49.5, by = 1),
     xlim = c(-0.5, 20),
     col = "grey80",
     main = "Repartitia Poisson",
     xlab = "X",
     ylab = "Densitatea")

lines(0:50,
      dpois(0:50, 5),
      type = "l",
      col = "brown3", lty = 2, lwd = 2)
```



b) Geometrică

```
# Geometric
hist(GenerateDiscrete(10000, x = 0:100,
                      p = dgeom(0:100, 0.3)),
     probability = TRUE,
     breaks = seq(-0.5,99.5, by = 1),
     xlim = c(-0.5, 20),
     col = "grey80",
     main = "Repartitia Geometrica",
     xlab = "X",
     ylab = "Densitatea")

lines(0:100,
```

```
      dgeom(0:100, 0.3),
      type = "l",
      col = "brown3", lty = 2, lwd = 2)
```



3.4 Generarea unei variabile aleatoare folosind metoda inversă

Scrieți un program care să folosească metoda transformării inverse pentru a genera n observații din densitatea

$$f(x) = \begin{cases} \frac{1}{x^2}, & x \geq 1 \\ 0, & \text{altfel} \end{cases}$$

Testați programul trasând o histogramă a 10000 de observații aleatoare împreună cu densitatea teoretică f.

Primul pas este să determinăm funcția de repartiție F corespunzătoare acestei densități. Pentru $x < 1$ avem că $f(x) = 0$ deci $F(x) = 0$ iar pentru $x \geq 1$ avem

$$F(x) = \int_1^x \frac{1}{t^2}\, dt = 1 - \frac{1}{x}.$$

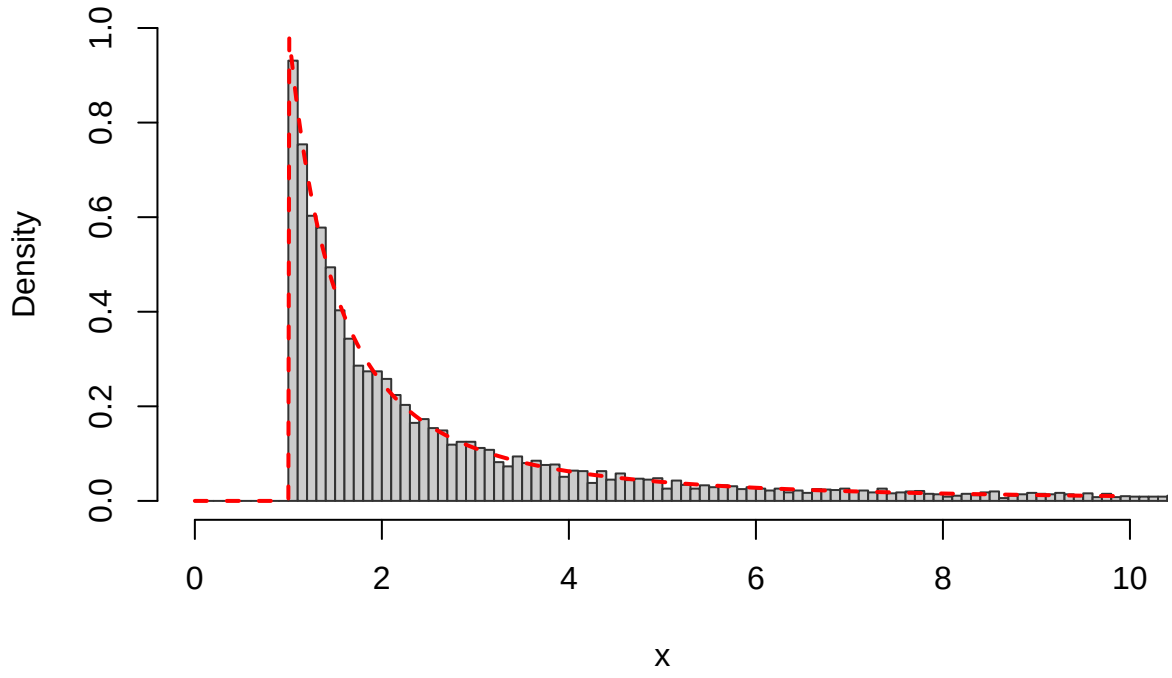
Cum F este continuă putem să determinăm F^{-1} rezolvând ecuația $F(x) = u$. Un calcul direct conduce la $F^{-1}(u) = \frac{1}{1-u}$ iar conform rezultatului văzut la curs concluzionăm că $X = \frac{1}{1-U}$ cu $U \sim \mathcal{U}([0,1])$.

Astfel putem simula un eșantion de talie n din populația f construind funcția

```
GenerateSampleX = function(n){
  u = runif(n)
  return(1/(1-u))
}
```

Pentru a testa comparăm valorile simulate cu densitatea teoretică

```
# simulate
x = GenerateSampleX(10000)
hist(x, freq=FALSE, breaks=seq(0, max(x)+1, 0.1),
     xlim=c(0,10), ylim=c(0,1),
     main=NULL, col="gray80", border="gray20")

# densitatea teoretica
y <- seq(0, 10, 0.01)
f <- ifelse(y <= 1, 0, 1/y^2)
lines(y, f, col = "red", lty = 2, lwd = 2)
```



3.5 Generarea unei repartiții normale

Plecând cu o propunere de tip $Exp(\lambda)$ vrem să generăm, cu ajutorul metodei acceptării-respingerii, un eșantion din următoarea densitate (jumătate de normală):

$$f(x) = \begin{cases} \frac{2}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}, & \text{dacă } x \geq 0 \\ 0, & \text{altfel} \end{cases}$$

Fie g densitatea repartiției exponențiale de parametru λ,

$$g(x) = \begin{cases} \lambda e^{-\lambda x}, & \text{dacă } x \geq 0 \\ 0, & \text{altfel} \end{cases}$$

Pentru a aplica algoritmul de acceptare-respingere trebuie să găsim valoarea lui $c > 0$ pentru care $f(x) \leq cg(x)$ pentru toate valorile $x \in \mathbb{R}$. Pentru $x \geq 0$ avem

$$\frac{f(x)}{g(x)} = \frac{2}{\lambda\sqrt{2\pi}} e^{-\frac{x^2}{2} + \lambda x}$$

și cum funcția $-\frac{x^2}{2} + \lambda x$ își atinge valoarea maximă în punctul $x = \lambda$ rezultă că

$$\frac{f(x)}{g(x)} \leq c^*, \ \forall x \geq 0$$

unde

$$c^* = \sqrt{\frac{2}{\pi\lambda^2}} e^{\lambda^2/2}.$$

Astfel algoritmul devine:

- pentru $n = 1, 2, \ldots$
- generează $X_n \sim Exp(\lambda)$
- generează $U_n \sim \mathcal{U}[0, 1]$
- dacă $U_n \leq \exp\left(-\frac{1}{2}(X_n - \lambda)^2\right)$ atunci
- intoarceți X_n

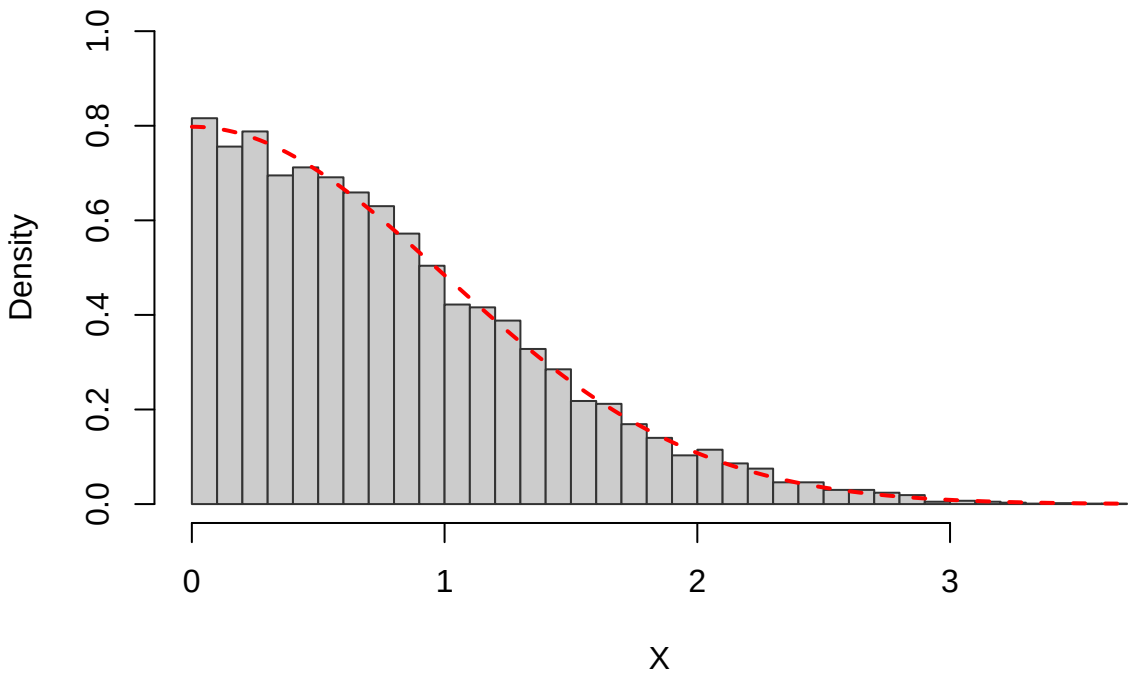
Avem funcția:

```
# generarea puntelor din densitatea f

f <- function(x) {
  return((x> 0) * 2 * dnorm(x,0,1))
}

g <- function(x) { return(dexp(x,1)) }

c <- sqrt(2 * exp(1) / pi)

rhalfnormal <- function(n) {
  res <- numeric(length=n)
  i <- 0
  while (i<n) {
    U <- runif(1, 0, 1)
    X <- rexp(1, 1)
```

```
    if (c * g(X) * U <= f(X)) {
      i <- i+1
      res[i] <- X;
    }
  }
  return(res)
}
```

Testăm

```
X <- rhalfnormal(10000)

hist(X,
     breaks=50,
     prob=TRUE,
     ylim=c(0,1),
     main=NULL,
     col="gray80",
     border="gray20")

curve(f, min(X), max(X), n=500, col = "red", lty = 2, lwd = 2, add=TRUE)
```



Modificați codul de la exercițiul precedent pentru a simula un eșantion dintr-o normală standard.

Cum f (din problema 1) este densitatea unei normale standard $X \sim \mathcal{N}(0,1)$ condiționată la $X > 0$ și cum

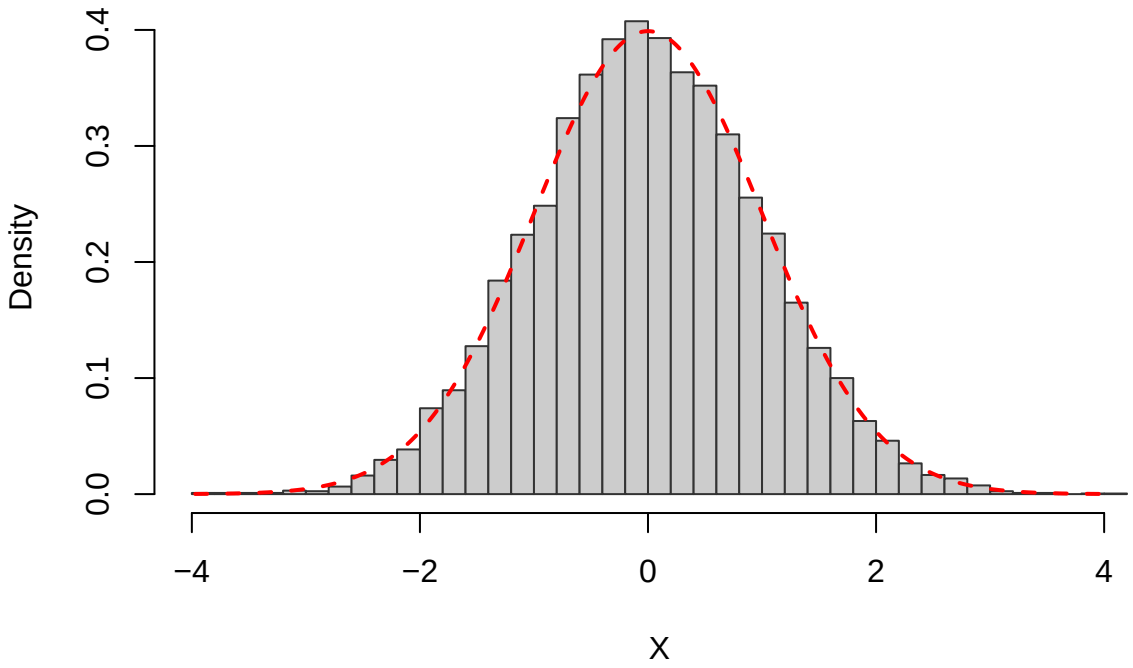
densitatea normală este simetrică față de medie (0 în acest caz) algoritmul se modifică acceptând x_n și $-X_n$ cu probabilitatea de 0.5.

Astfel avem funcția:

```
f2 <- function(x) {
  return(dnorm(x,0,1))
}

normal1 <- function(n) {
  res <- numeric(length=n)
  i <- 0
  while (i<n) {
    U <- runif(1, 0, 1)
    X <- rexp(1, 1)
    if (c * g(X) * U <= f(X)) {
      i <- i+1

      res[i] <- ifelse(runif(1) <= 0.5, X, -X);
    }
  }
  return(res)
}
```

si testul

```
X <- normal1(10000)

hist(X, breaks=50,
     prob=TRUE,
     main=NULL,
     col="gray80", border="gray20")

curve(f2, min(X), max(X), n=500,col = "red", lty = 2, lwd = 2, add=TRUE)
```



3.6 Simularea unei uniforme pe disc

Considerăm pătratul $C = [0, L]^2$ și discul D de centru $(\frac{L}{2}, \frac{L}{2})$ și rază $\frac{L}{2}$. Considerăm șirul de v.a. $(Y_n)_{n\geq 1}$ pe $\mathbb{R}^2$ i.i.d. repartizate uniform pe pătratul C.

1. Aproximați valoarea lui π prin ajutorul numărului de puncte Y_n care cad în interiorul discului D (Metoda respingerii)
2. Simulați n puncte uniforme pe disc.

1. Definim v.a. $X_n = \mathbf{1}_{\{Y_n \in D\}}$, $n \geq 1$, care formează un șir de v.a. i.i.d. de lege $\mathcal{B}(\mathbb{P}(Y_n \in D))$, deoarece $(Y_n)_{n\geq 1}$ este un șir de v.a. i.i.d. repartizate uniform pe C, $\mathcal{U}(C)$. Din *Legea Numerelor Mari* avem că

$$\frac{1}{n}\sum_{i=1}^{n} X_i \overset{a.s.}{\to} \mathbb{E}[X_1] = \mathbb{P}(Y_1 \in D),$$

prin urmare trebuie să calculăm probabilitatea $\mathbb{P}(Y_1 \in D)$. Știm că densitatea v.a. Y_1 este dată de $f_{Y_1}(x, y) = \frac{1}{\mathcal{A}(C)}\mathbf{1}_C(x, y)$ de unde

$$\begin{aligned}\mathbb{P}(Y_1 \in D) &= \iint_D f_{Y_1}(x, y)\, dxdy = \iint \mathbf{1}_D(x, y)\mathbf{1}_C(x, y)\, dxdy \\ &= \frac{1}{\mathcal{A}(C)} \iint \mathbf{1}_D(x, y)\, dxdy = \frac{\mathcal{A}(D)}{\mathcal{A}(C)} = \frac{\pi\frac{L^2}{4}}{L^2} = \frac{\pi}{4}.\end{aligned}$$

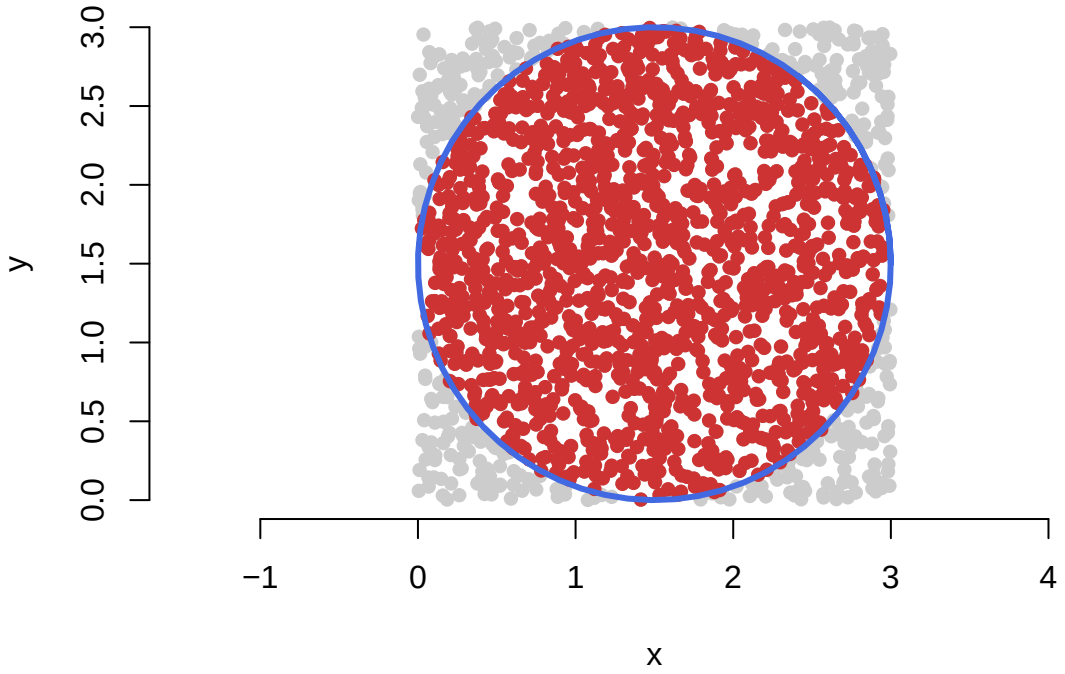
Astfel, putem estima valoarea lui π prin $\frac{4}{n}\sum_{i=1}^{n} X_i$ pentru valori mari ale lui n.

```
# Estimam valoarea lui pi

L = 3 # lungimea laturii patratului
R = L/2 # raza cercului inscris

n = 2000 # numarul de puncte din patratul C
# generam puncte uniforme in C
x = L*runif(n)
y = L*runif(n)

# metoda respingerii (rejectiei)
l = (x-R)^2+(y-R)^2 # distanta dintre centrul cercului si punct
ind = l<=(R)^2 # indicii pentru care distanta este mai mica sau egala cu R

xc = x[ind] # coordonatele punctelor din interiorul cercului
yc = y[ind]

estimate_pi = 4*sum(ind)/n # estimarea lui pi
err = abs(estimate_pi-pi) # eroarea absoluta
```

Aplicând acest procedeu obținem că valoarea estimată a lui π prin generarea a $n = 2000$ puncte este 3.16 iar eroarea absoluta este 0.01841.

2. Una dintre metodele prin care putem simula puncte uniform repartizate pe suprafața discului D este *Metoda respingerii*. Această metodă consistă în generarea de v.a. Y_n repartizate uniform pe suprafața pătratului C, urmând ca apoi să testăm dacă Y_n aparține discului D (deoarece $D \subset C$). Dacă da, atunci le păstrăm dacă nu atunci mai generăm. Următoarea figură ilustrează această metodă:

```
# figura
theta = seq(0, 2*pi+1, by = 0.1)
xd = R+R*cos(theta)
yd = R+R*sin(theta)

plot(x, y,
     col = "grey80", pch = 16,
     asp = 1,
     xlim = c(0,3), ylim = c(0,3),
     bty = "n")

points(xc, yc, col = "brown3", pch = 16)
lines(xd, yd, col = "royalblue", lwd = 3)
```



Vom da mai jos o altă metodă de simulare a punctelor distribuite uniform pe discul D de rază L. O primă idee ar fi să generăm cuplul de v.a. (X_1, Y_1) așa încât $X_1, Y_1 \sim \mathcal{U}([0, L])$ și ele să fie independente (ceea ce nu este adevărat în realitate). Vom vedea (printr-o ilustrație grafică) că această abordare este greșită (punctele sunt concentrate în centrul cercului).

O altă abordare este următoarea. Căutăm să simulăm un cuplu de v.a. (X, Y) care este uniform distribuit pe suprafața discului D, i.e. densitatea cuplului este dată de $f_{(X,Y)}(x, y) = \frac{1}{\pi L^2}\mathbf{1}_D(x, y)$. Considerăm schimbarea de variablile în coordonate polare: $x = r\cos(\theta)$ și $y = r\sin(\theta)$. Obiectivul este de a găsi densitatea variabilelor R și Θ.

Fie $g(x, y) = \left(\sqrt{x^2 + y^2}, \arctan(y/x)\right) = (r, \theta)$, transformarea pentru care avem $(R, \Theta) = g(X, Y)$. Știm că inversa acestei transformări este $g^{-1}(r, \theta) = (r\cos(\theta), r\sin(\theta))$, prin urmare

$$\begin{aligned} f_{(R,\Theta)}(r, \theta) &= f_{(X,Y)}\left(g^{-1}(r, \theta)\right) |\det(J_{g^{-1}}(r, \theta))| \\ &= \frac{1}{\pi L^2}\mathbf{1}_D(r\cos(\theta), r\sin(\theta)) \begin{vmatrix} \cos(\theta) & \sin(\theta) \\ r\sin(\theta) & -r\cos(\theta) \end{vmatrix} \\ &= \frac{1}{\pi L^2}\mathbf{1}_{[0,L]}(r)\mathbf{1}_{[0,2\pi]}(\theta) r. \end{aligned}$$

Observăm că densitatea (marginală) v.a. Θ este

$$\begin{aligned} f_\Theta(\theta) &= \int f_{(R,\Theta)}(r, \theta)\, dr = \mathbf{1}_{[0,2\pi]}(\theta) \int \frac{r}{\pi L^2}\mathbf{1}_{[0,L]}(r)\, d\theta \\ &= \frac{1}{\pi L^2}\mathbf{1}_{[0,2\pi]}(\theta)\frac{L^2}{2} = \frac{1}{2\pi}\mathbf{1}_{[0,2\pi]}(\theta), \end{aligned}$$

iar densitatea v.a. R este

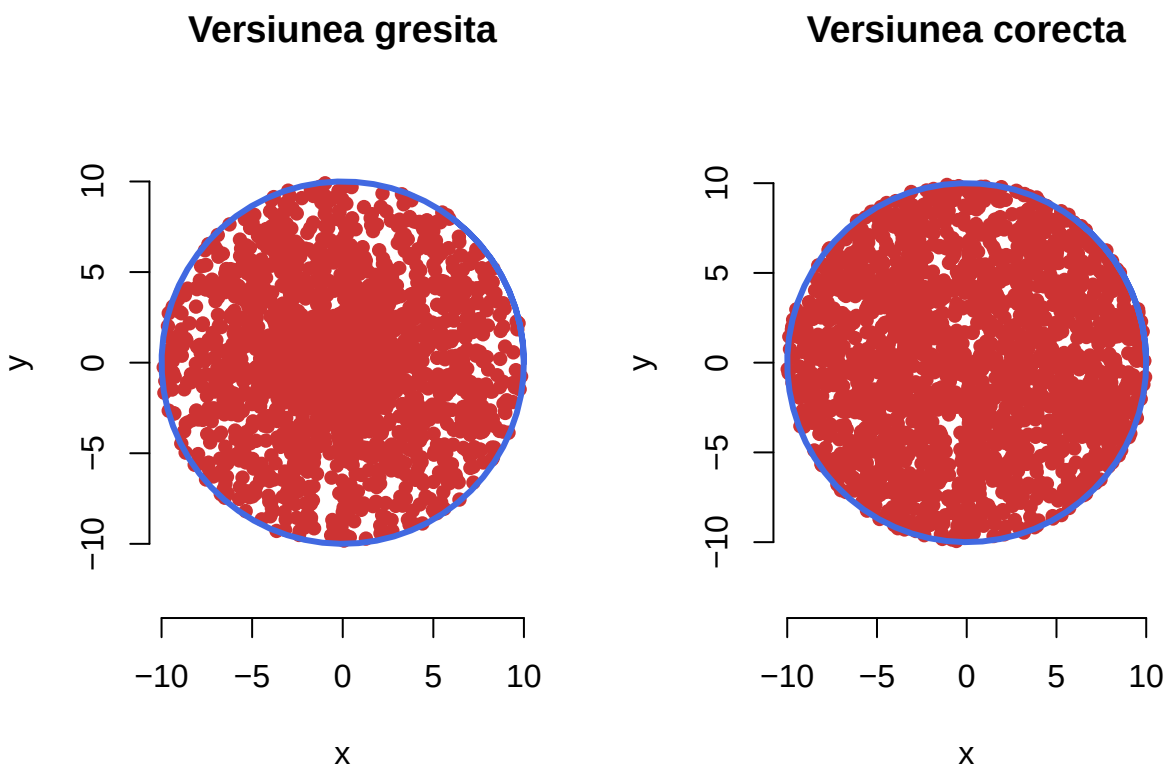
$$f_R(r) = \int f_{(R,\Theta)}(r,\theta)\,d\theta = \frac{r}{\pi L^2}\mathbf{1}_{[0,L]}(r)\int_0^{2\pi} d\theta$$
$$= \frac{r}{\pi L^2}\mathbf{1}_{[0,L]}(r)2\pi = \frac{2r}{L^2}\mathbf{1}_{[0,L]}(r).$$

Din expresiile de mai sus putem observa că Θ este o v.a. repartizată uniform pe $[0,2\pi]$ și putem verifica ușor că legea v.a. R este aceeași cu cea a v.a. $L\sqrt{U}$ unde $U \sim \mathcal{U}([0,1])$.

Astfel pentru simularea unui punct (X,Y) uniform pe D este suficient să simulăm o v.a. Θ uniform pe $[0,2\pi]$ și o v.a. U uniformă pe $[0,1]$ și să luăm $X = L\sqrt{U}\cos(\Theta)$ și $Y = L\sqrt{U}\sin(\Theta)$.

Următorul cod ne ilustrează cele două proceduri prezentate:

```
rm(list=ls())

n = 2000;# numarul de puncte

R = 10;# raza cercului

theta = 2*pi*runif(n);# theta este uniforma pe [0,2*pi]

# versiunea gresita - r este uniforme pe [0,R]
r1 = R*runif(n);

x1 = r1*cos(theta);# coordonate polare
y1 = r1*sin(theta);

# versiunea corecta
r2 = R*sqrt(runif(n));

x2 = r2*cos(theta);# coordonate polare
y2 = r2*sin(theta);

# schimbarea de variabila in coordonate polare: cercul
theta2 = seq(0,2*pi+1,by=0.1)
xc = R*cos(theta2);
yc = R*sin(theta2);

# graficul

par(mfrow = c(1,2))

plot(x1, y1,
     ylim = c(-11, 11),
     col = "brown3", pch = 16,
     main = "Versiunea gresita", xlab = "x", ylab = "y", asp = 1, bty = "n")
lines(xc, yc, lwd = 3, col = "royalblue")

plot(x2, y2,
     ylim = c(-11, 11),
     col = "brown3", pch = 16,
     main = "Versiunea corecta", xlab = "x", ylab = "y", asp = 1, bty = "n")
lines(xc, yc, lwd = 3, col = "royalblue")
```


Versiunea gresita
Versiunea corecta
y
x
−10
−5
0
5
10