Proiect Statistică

Chirobocea Mihail-Bogdan 312
Costea Răzvan-George 311
Marin Mircea-Mihai 312
Popa Mihai-Cristian 311

Lider echipă: Chirobocea Mihail-Bogdan

I. Inegalitatea Berry-Essen

1. Am tansmis prin parametru pdf-ul / pmf-ul variabilei aleatoare X. Au fost folosite formulele pentru medie şi varianţă:

$$E(X) = \int_{-\infty}^{\infty} x f(x)\,dx \quad / \quad \sum_{i=0}^{n} x_i p_i$$

$$Var(X) = E(X^2) - E(X)^2$$

```r
#1

# Expected value for continuous random variable
E_c <- function(pdf){
  x1 <- runif(10^6)
  x_inf <- (1-x1)/x1
  f <- function(x) x * pdf(x) * 1/x1^2
  return(mean(f(x_inf)))
}

# Variance for continuous random variable
Var_c <- function(pdf){
  x1 <- runif(10^6)
  x_inf <- (1-x1)/x1
  f <- function(x) x**2 * pdf(x) * 1/x1^2
  return(mean(f(x_inf)) - E_c(pdf)**2)
}

# Expected value for discrete random variable
E_d <- function(pmf) {
  x <- seq(0, 10^2, 1)
  return(sum(x*pmf(x)))
}

# Variance for discrete random variable
Var_d <- function(pmf) {
  x <- seq(0, 10^2, 1)
  prob <- pmf(x)
  return(sum(x**2*prob) - sum(x*prob)**2)
}

```

În cazul discret, probabilităţile sunt calculate folosind funcţia de masa, i.e. pmf. În cazul continuu integrala este calculată folosind metoda Monte Carlo.

2. Functia primește ca parametru o variabila aleatoare X si returneaza rezultatul cerut.

```r
#2
mom_3 <- function(X){
  return(mean(abs(X - mean(X))**3))
}

```

3. Am scris o funcție care calculează marginea Berry-Essen care primește ca parametru variabila aleatoare X_1 și volumul eșantionului n. Apoi am construit un dataframe care contine primele 2 coloane, i.e. distribuția si volumul pentru care va fi calculată marginea. Pentru fiecare volum n am generat 10^6 numere din fiecare distribuție și am calculat marginea. Aceste rezultate sunt puse intr-un vector, iar acest vector este adaugat ca și coloană la dataframe.

```r
#3
BE_Margin <- function(X_1, n){
  sigma = sqrt(mean(X_1**2) - mean(X_1)**2)
  return(33/4 * (mom_3(X_1)) / (sqrt(n) * sigma**3))
}

names = c('Binomial', 'Geometric', 'Poisson', 'Uniform', 'Exponential', 'Gamma', 'Beta')
volumes = c(30, 100, 1000)

# Create a dataframe with Distribution and Volume columns
BE_Margins <- data.frame(
  'Distribution' = c(c(names, names), names),
  'Volume' = sort(c(c(c(c(c(c(volumes, volumes), volumes), volumes), volumes), volumes), volumes))
)

margins <- c()

for(n in volumes){
  # Discrete rv
  X_1_bin = rbinom(10^6, 3, 0.4)
  X_1_geom = rgeom(10^6, 0.3)
  X_1_pois = rpois(10^6, 10)
  X_1_unif = sample(1:20, 10^6, replace=T)

  # Continuous rv
  X_1_exp = rexp(10^6, 3)
  X_1_gamma = rgamma(10^6, 9, 0.5)
  X_1_beta = rbeta(10^6, 2, 2)

  # Calculate the B-E margins for every given distribution
  margins <- c(margins, BE_Margin(X_1_bin, n))
  margins <- c(margins, BE_Margin(X_1_geom, n))
  margins <- c(margins, BE_Margin(X_1_pois, n))
  margins <- c(margins, BE_Margin(X_1_unif, n))
  margins <- c(margins, BE_Margin(X_1_exp, n))
  margins <- c(margins, BE_Margin(X_1_gamma, n))
  margins <- c(margins, BE_Margin(X_1_beta, n))

}

# Append the Margin column to dataframe
BE_Margins$Margin = margins

# Print
BE_Margins
```

	Distribution	Volume	Margin
1	Binomial	30	2.2131096
2	Geometric	30	3.6616273
3	Poisson	30	2.4235634
4	Uniform	30	1.9543812
5	Exponential	30	3.6182861
6	Gamma	30	2.5472591
7	Beta	30	2.1057724
8	Binomial	100	1.2110884
9	Geometric	100	2.0002072
10	Poisson	100	1.3262601
11	Uniform	100	1.0703577
12	Exponential	100	2.0032390
13	Gamma	100	1.3992403
14	Beta	100	1.1530103
15	Binomial	1000	0.3831067
16	Geometric	1000	0.6334175
17	Poisson	1000	0.4192049
18	Uniform	1000	0.3385089
19	Exponential	1000	0.6304585
20	Gamma	1000	0.4419005
21	Beta	1000	0.3645935

În tabel se poate observa că marginea scade cu cât volumul eșantionului crește. Deci aproximarea realizată de TLC este mai exactă dacă volumul eșantionului este mare.

Cum $\mu<\infty \wedge \sigma<\infty \wedge E(|X_i|^3)<\infty$, atunci se poate observa că

$$\lim_{n\to\infty}\left(\frac{33}{4}\frac{E(|X_1-\mu|)^3}{\sqrt{(n)}\,\sigma^3}\right)=0$$

4. Pentru fiecare volum n a fost construită variabila aleatoare Z_n , calculând mai întâi media de selecție $\overline{X}$. Funcția de repartiție a fost calculată folosind funcția din R ecdf().

Întervalul ales este [0,1] care a fost construit in R folosind funcția seq(). Pentru fiecare punct din secvență se calculează diferența cerută care este stocată intr-un vector pentru a fi plotată.

Procesul descris mai sus este repetat pentru fiecare repartiție cerută.

```r
# Binomial
volumes <- c(30, 100, 1000)

for(n in volumes){
  X_bar <- c()

  # Init the sampling mean
  X_1 <- rbinom(10^6, 3, 0.4)
  X_bar <- c(X_bar, mean(X_1))

  for(i in 2:n){
    X_i <- rbinom(10^6, 3, 0.4)
    X_bar <- c(X_bar, mean(X_i))
  }

  # Init Z_n rv
  sigma = sqrt(mean(X_1**2) - mean(X_1)**2)
  Z_n <- (sqrt(n) * (X_bar - mean(X_1))) / sigma

  # Use ecdf() function to find the cdf of Z_n
  cdf <- ecdf(Z_n)

  # Define [0, 1] interval
  x <- seq(0, 1, 0.001)
  diff <- c()
  for(t in x){
    # Compute the difference
    diff <- c(diff, cdf(t) - pnorm(t))
  }

  # Plot
  plot(x, diff, main=sprintf('Binomial with %d volume', n), lwd=0.5)
  lines(x, diff, col="red")
}
```

```r
# Geometric
volumes <- c(30, 100, 1000)

for(n in volumes){
  X_bar <- c()

  # Init the sampling mean
  X_1 <- rgeom(10^6, 0.3)
  X_bar <- c(X_bar, mean(X_1))

  for(i in 2:n){
    X_i <- rgeom(10^6, 0.3)
    X_bar <- c(X_bar, mean(X_i))
  }

  # Init Z_n rv
  sigma = sqrt(mean(X_1**2) - mean(X_1)**2)
  Z_n <- (sqrt(n) * (X_bar - mean(X_1))) / sigma

  # Use ecdf() function to find the cdf of Z_n
  cdf <- ecdf(Z_n)

  # Define [0, 1] interval
  x <- seq(0, 1, 0.001)
  diff <- c()
  for(t in x){
    # Compute the difference
    diff <- c(diff, cdf(t) - pnorm(t))
  }

  # Plot
  plot(x, diff, main=sprintf('Geometric with %d volume', n), lwd=0.5)
  lines(x, diff, col="red")
}
```

```r
# Poisson
volumes <- c(30, 100, 1000)

for(n in volumes){
  X_bar <- c()

  # Init the sampling mean
  X_1 <- rpois(10^6, 10)
  X_bar <- c(X_bar, mean(X_1))

  for(i in 2:n){
    X_i <- rpois(10^6, 10)
    X_bar <- c(X_bar, mean(X_i))
  }

  # Init Z_n rv
  sigma = sqrt(mean(X_1**2) - mean(X_1)**2)
  Z_n <- (sqrt(n) * (X_bar - mean(X_1))) / sigma

  # Use ecdf() function to find the cdf of Z_n
  cdf <- ecdf(Z_n)

  # Define [0, 1] interval
  x <- seq(0, 1, 0.001)
  diff <- c()
  for(t in x){
    # Compute the difference
    diff <- c(diff, cdf(t) - pnorm(t))
  }

  # Plot
  plot(x, diff, main=sprintf('Poisson with %d volume', n), lwd=0.5)
  lines(x, diff, col="red")
}
```

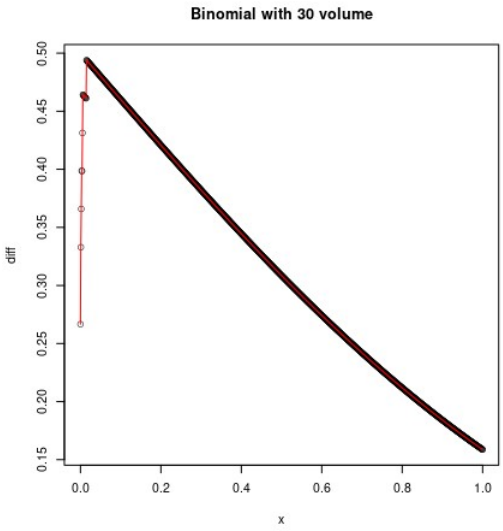
```r
# Discrete Uniform
volumes <- c(30, 100, 1000)

for(n in volumes){
  X_bar <- c()

  # Init the sampling mean
  X_1 <- sample(1:20, 10^6, replace = T)
  X_bar <- c(X_bar, mean(X_1))

  for(i in 2:n){
    X_i <- sample(1:20, 10^6, replace = T)
    X_bar <- c(X_bar, mean(X_i))
  }

  # Init Z_n rv
  sigma = sqrt(mean(X_1**2) - mean(X_1)**2)
  Z_n <- (sqrt(n) * (X_bar - mean(X_1))) / sigma

  # Use ecdf() function to find the cdf of Z_n
  cdf <- ecdf(Z_n)

  # Define [0, 1] interval
  x <- seq(0, 1, 0.001)
  diff <- c()
  for(t in x){
    # Compute the difference
    diff <- c(diff, cdf(t) - pnorm(t))
  }

  # Plot
  plot(x, diff, main=sprintf('Discrete Uniform with %d volume', n), lwd=0.5)
  lines(x, diff, col="red")
}
```

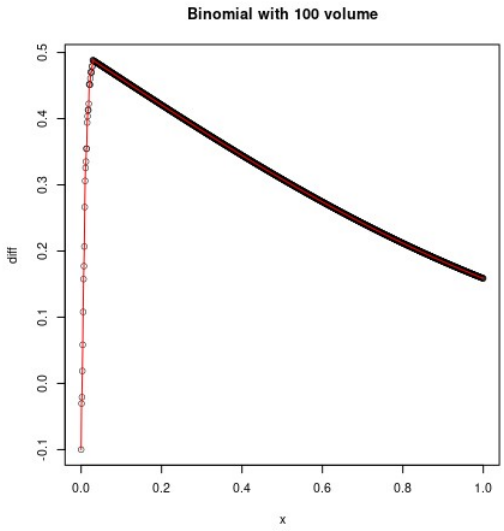
```r
# Exponential
volumes <- c(30, 100, 1000)

for(n in volumes){
  X_bar <- c()

  # Init the sampling mean
  X_1 <- rexp(10^6, 3)
  X_bar <- c(X_bar, mean(X_1))

  for(i in 2:n){
    X_i <- rexp(10^6, 3)
    X_bar <- c(X_bar, mean(X_i))
  }

  # Init Z_n rv
  sigma = sqrt(mean(X_1**2) - mean(X_1)**2)
  Z_n <- (sqrt(n) * (X_bar - mean(X_1))) / sigma

  # Use ecdf() function to find the cdf of Z_n
  cdf <- ecdf(Z_n)

  # Define [0, 1] interval
  x <- seq(0, 1, 0.001)
  diff <- c()
  for(t in x){
    # Compute the difference
    diff <- c(diff, cdf(t) - pnorm(t))
  }

  # Plot
  plot(x, diff, main=sprintf('Exponential Uniform with %d volume', n), lwd=0.5)
  lines(x, diff, col="red")
}
```

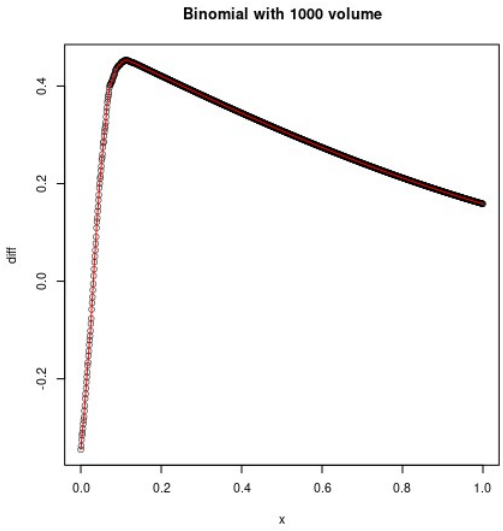
```r
# Gamma
volumes <- c(30, 100, 1000)

for(n in volumes){
  X_bar <- c()

  # Init the sampling mean
  X_1 <- rgamma(10^6, 9, 0.5)
  X_bar <- c(X_bar, mean(X_1))

  for(i in 2:n){
    X_i <- rgamma(10^6, 9, 0.5)
    X_bar <- c(X_bar, mean(X_i))
  }

  # Init Z_n rv
  sigma = sqrt(mean(X_1**2) - mean(X_1)**2)
  Z_n <- (sqrt(n) * (X_bar - mean(X_1))) / sigma

  # Use ecdf() function to find the cdf of Z_n
  cdf <- ecdf(Z_n)

  # Define [0, 1] interval
  x <- seq(0, 1, 0.001)
  diff <- c()
  for(t in x){
    # Compute the difference
    diff <- c(diff, cdf(t) - pnorm(t))
  }

  # Plot
  plot(x, diff, main=sprintf('Gamma Uniform with %d volume', n), lwd=0.5)
  lines(x, diff, col="red")
}
```

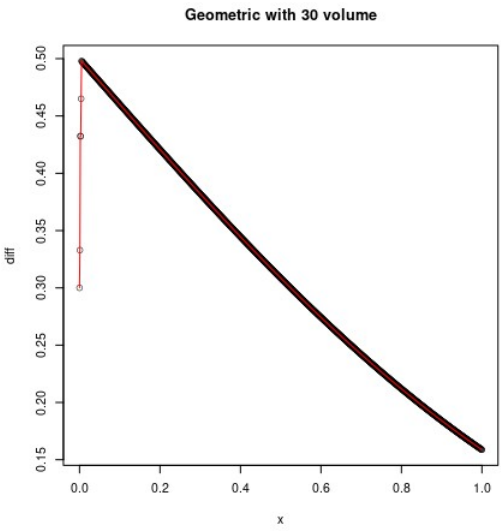
```r
# Beta
volumes <- c(30, 100, 1000)

for(n in volumes){
  X_bar <- c()

  # Init the sampling mean
  X_1 <- rbeta(10^6, 2, 2)
  X_bar <- c(X_bar, mean(X_1))

  for(i in 2:n){
    X_i <- rbeta(10^6, 2, 2)
    X_bar <- c(X_bar, mean(X_i))
  }

  # Init Z_n rv
  sigma = sqrt(mean(X_1**2) - mean(X_1)**2)
  Z_n <- (sqrt(n) * (X_bar - mean(X_1))) / sigma

  # Use ecdf() function to find the cdf of Z_n
  cdf <- ecdf(Z_n)

  # Define [0, 1] interval
  x <- seq(0, 1, 0.001)
  diff <- c()
  for(t in x){
    # Compute the difference
    diff <- c(diff, cdf(t) - pnorm(t))
  }

  # Plot
  plot(x, diff, main=sprintf('Beta Uniform with %d volume', n), lwd=0.5)
  lines(x, diff, col="red")
}
```

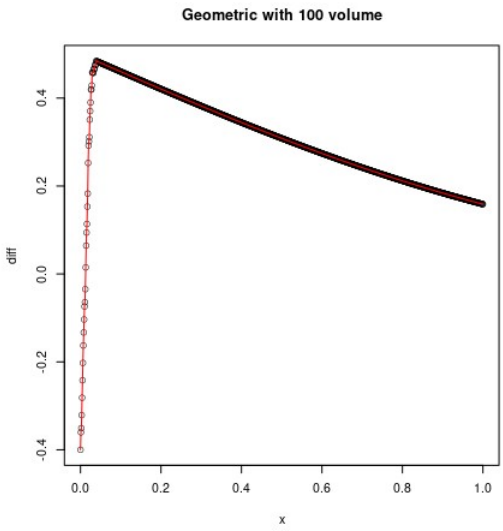
Mai jos se pot analiza graficele pentru fiecare repartiție:

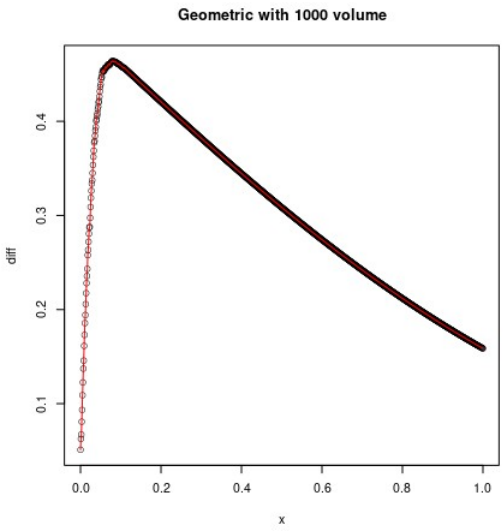


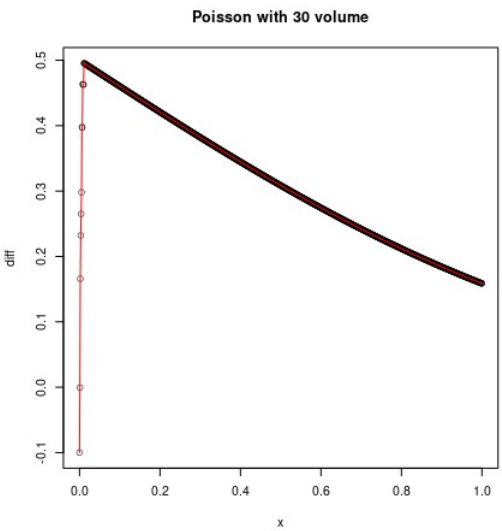


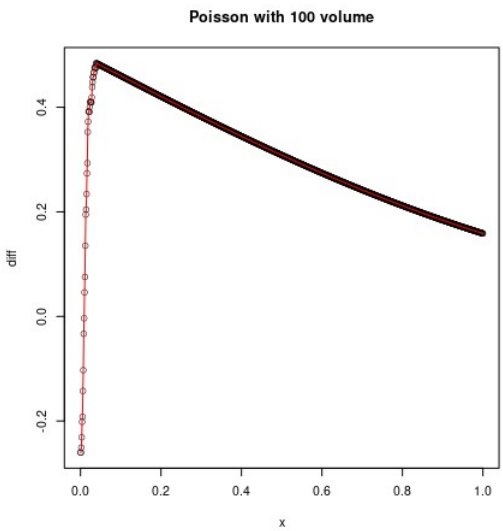


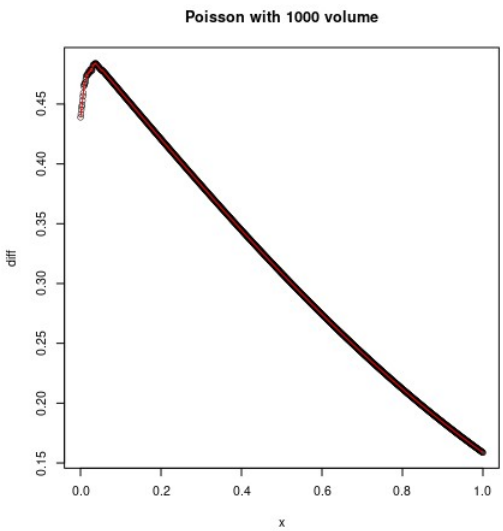


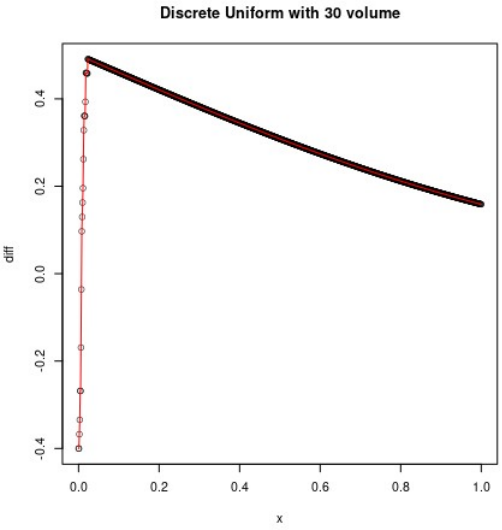


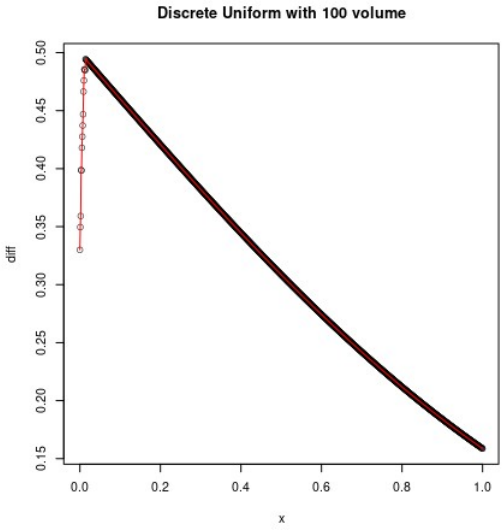


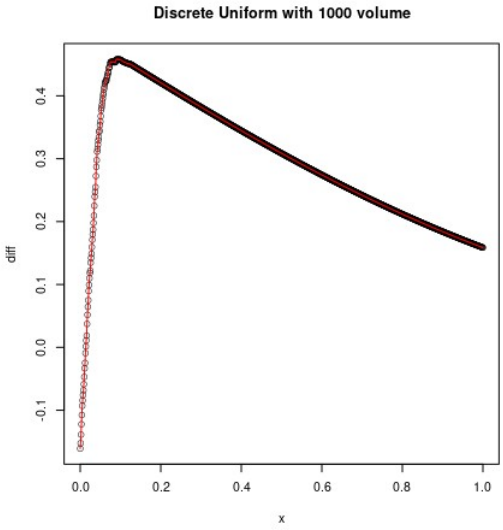


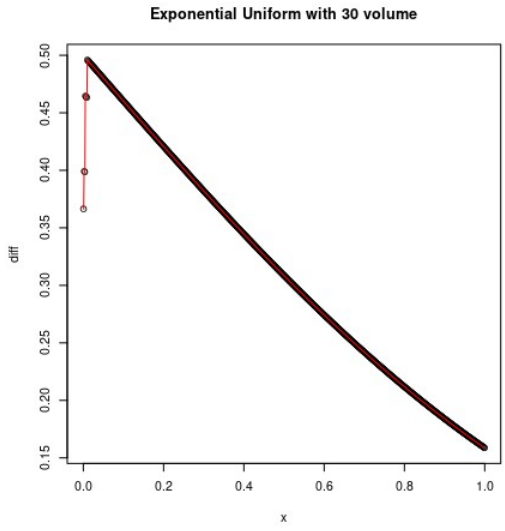


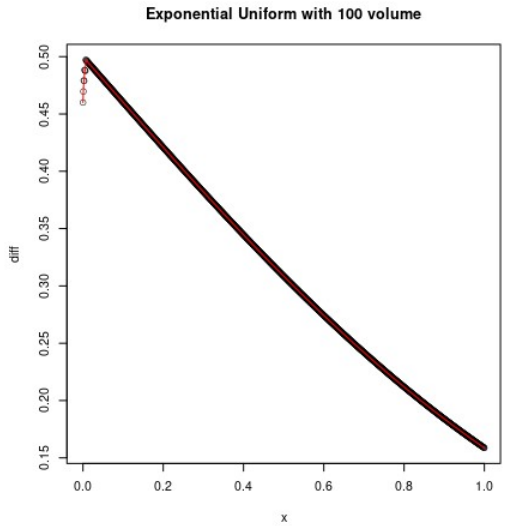


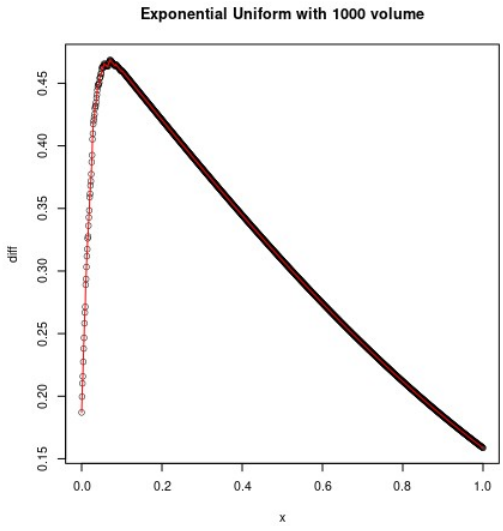

Discrete Uniform with 30 volume
diff
x

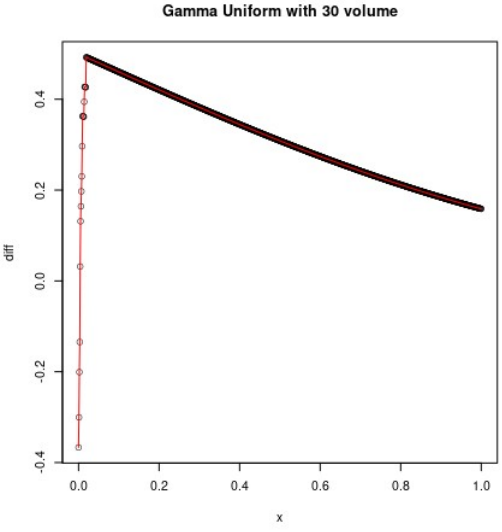

Discrete Uniform with 100 volume
diff
x

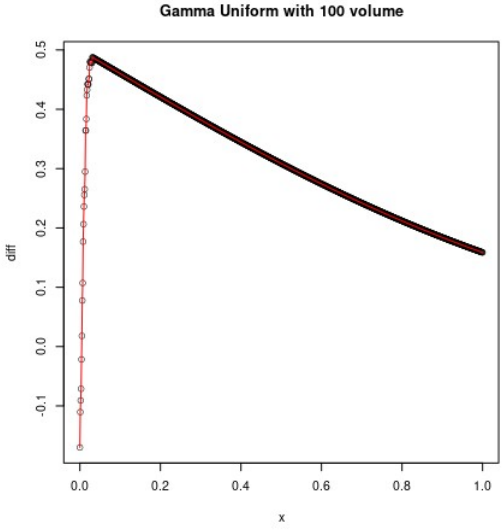

Discrete Uniform with 1000 volume
diff
x

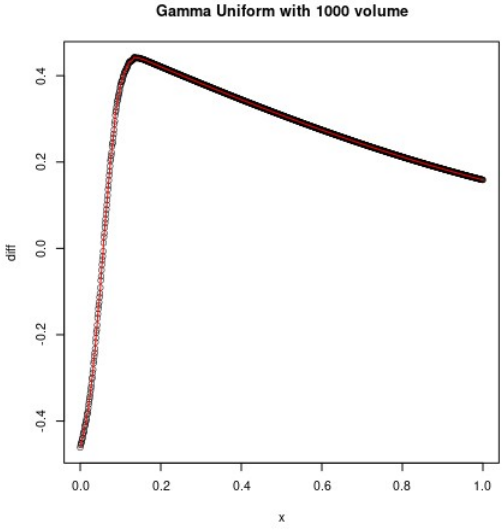

Exponential Uniform with 30 volume
diff
x

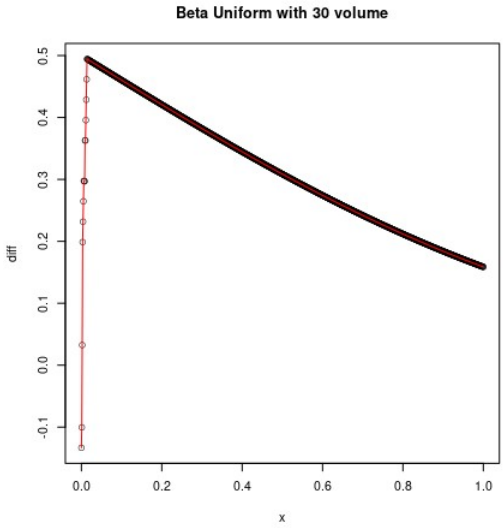

Exponential Uniform with 100 volume
diff
x

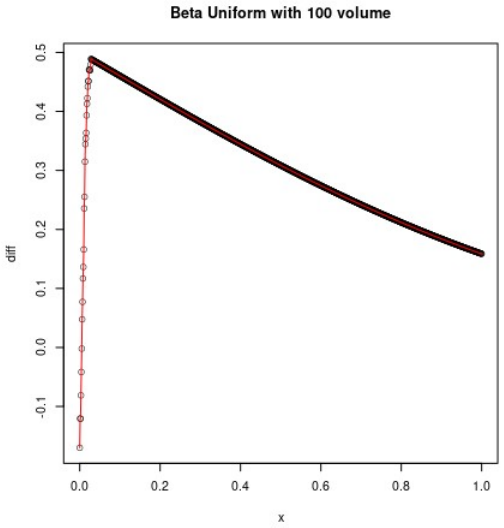

Exponential Uniform with 1000 volume
diff
x

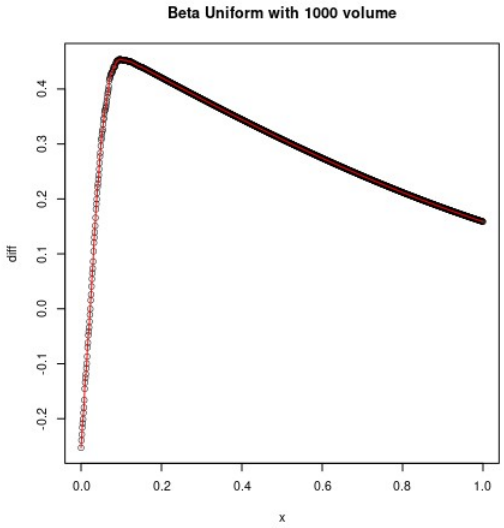

Gamma Uniform with 30 volume
diff
x


Gamma Uniform with 100 volume
diff
x


Gamma Uniform with 1000 volume
diff
x







5. Ştiind pdf-ul, respectiv pmf-ul, puntem calcula funcţia de repartiţie pentru a simula variabila aleatoare folosind metoda inversă. Am scris o funcţie pentru cazul discret şi una pentru cazul continuu. Fiecare dintre acestea foloseşte funcţiile scire la punctul 1.

```r
# Compute B-E margin for continous rv
Margin_c <- function(X_1, n, pdf){
  miu <- E_c(pdf)
  sigma <- sqrt(Var_c(pdf))
  return(33/4 * mean(abs(X_1 - miu)**3) / (sqrt(n) * sigma**3))
}

# Compute B-E margin for discrete rv
Margin_d <- function(X_1, n, pmf){
  miu <- E_d(pmf)
  sigma <- sqrt(Var_d(pmf))
  return(33/4 * mean(abs(X_1 - miu)**3) / (sqrt(n) * sigma**3))
}

pdf <- function(x){
  3*exp(-3*x)
}

pmf <- function(x){
  exp(-10)*(10^x)/(factorial(x))
}

generate_c <- function(){
  # we know the pdf/pmf so we can find cdf/cmf and apply inverse method for exp(lambda=3)
  u <- runif(10^6)
  x <- -1/3*log(u)
  return(x)
}

generate_d <- function(){
  # we know the pdf/pmf so we can find cdf/cmf and apply inverse method for pois(lambda=10)
  X <- c()
  for(i in 1:10^6){
    u <- runif(1)
    j <- 0
    p <- exp(-10)
    F_x <- p
    while(u >= F_x){
      p <- (10*p)/(j+1)
      F_x <- F_x + p
      j <- j + 1
    }
    X[i] <- j
  }
  return(X)
}

X_1 <- generate_c()
Margin_c(X_1, 1000, pdf)

X_1 <- generate_d()
Margin_d(X_1, 1000, pmf)
```

Rezultatele obţinute folosind media si varianţa de la punctul 1 sunt foarte apropiate faţă de cele de la punctul 3, unde media a fost calculată folosind funcţia din R mean(). Şi în acest caz, are loc concluzia dată la punctul 3.

Exercițiul 2

2.1.

Probabilitatea de acceptare a unei valori prin metoda respingerii este $\frac{1}{c}$, unde c este constanta care mărginește raportul $\frac{f(x)}{g(x)}$.

Pentru a putea ajunge la acel rezultat, ne întoarcem la algoritmul metodei respingerii:

1. Generez *U*
2. Generez *Y*
3.
   a. Dacă $U \leq \frac{f(Y)}{cg(Y)}$, atunci $X \leftarrow Y$
   b. Altfel, înapoi la pasul 1.

Pasul de acceptare-respingere, **3**, conține probabilitatea de acceptare :

$$P('acceptare') = P(U \leq \frac{f(Y)}{cg(Y)}) =$$

$$= \int_D P\left(U \leq \frac{f(Y)}{cg(Y)} \middle| Y = y\right) g(y)dy$$

$$= \int_D \frac{f(y)}{g(y)} g(y)dy = \frac{1}{c}\int_D f(y)dy = \frac{1}{c}$$

pentru f, g funcții de densitate ale unor variabile aleatoare, unde D este domeniul de definiție al lui f(domeniu pe care l-am restrâns și pe g, i.e. $g = g_D$).

2.2.

$E\left[1_{U \leq \frac{f}{g}}\right] = P\left(U \leq \frac{f(Y)}{g(Y)}\right) = \frac{1}{c}$, conform subpunctului 2.1.

Pentru funcția f dată, unde nu cunoaștem constanta de normalizare, considerăm

$$f(x) = \alpha p(x); p(x) = \sin^2 x \ e^{-x^2\sqrt{x}}$$

Atunci, $f(x) \leq cg(x) \Leftrightarrow p(x) \leq \frac{c}{\alpha} g(x)$. Notând $\frac{c}{\alpha} \coloneqq M$, $p(x) \leq Mg(x)$.

Atât pentru $f(x)$ cât și pentru $p(x)$ se poate aplica metoda respingerii, și din studiul celor două, putem găsi aproximări ale constantei de normalizare α.

2.4.

Pentru a rezolva acest subpunct, am folosit cod R, aplicând principiile de la 2.2.

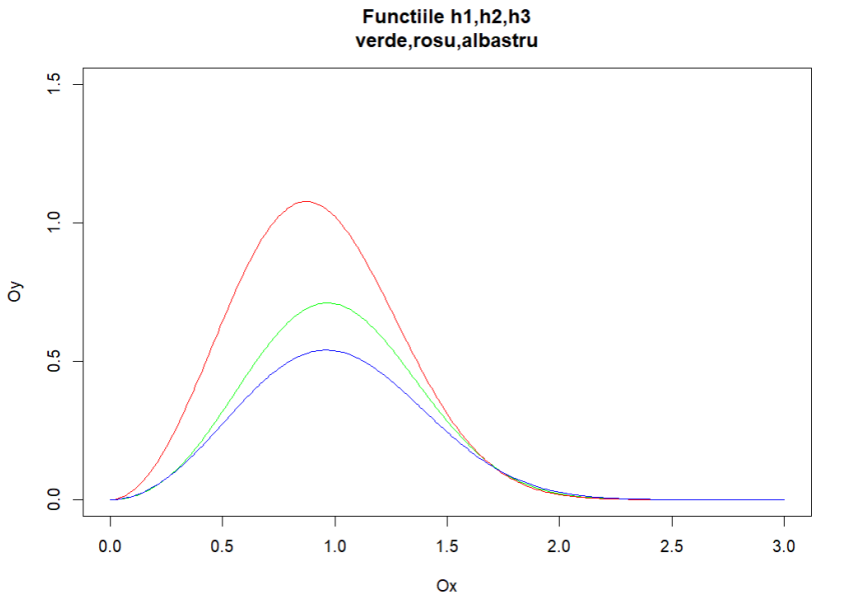
Începem cu definițiile din enunț

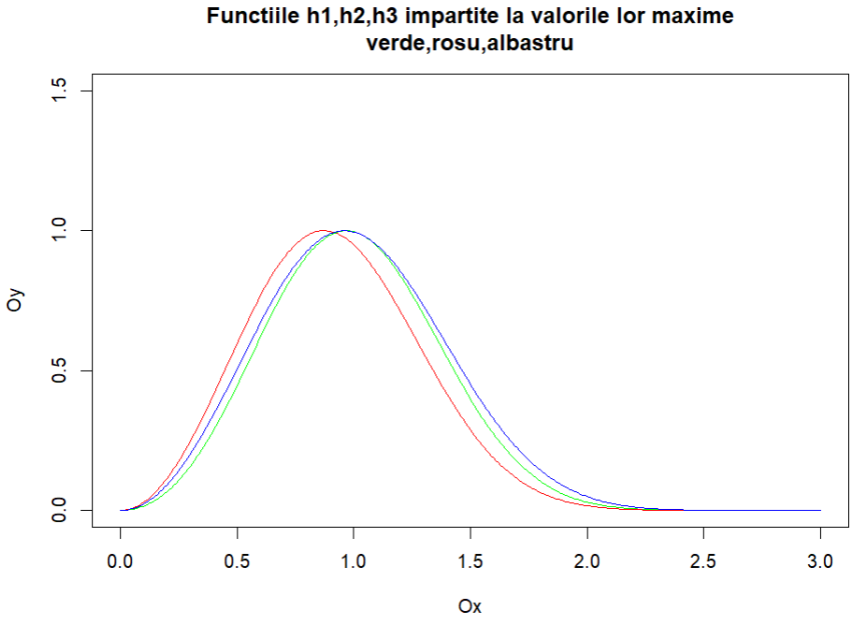
```r
#Exercitiul 2, subpunctul 4

#partea ne-constanta a lui f. in alte cuvinte, f proportional cu fb.
fb <- function(x) {
  (sin(x))^2*exp(-x^2*sqrt(x))
}

#g1,g2,g3 din enunt
g1 <- function(x) {
  (exp(-x))
}

g2 <- function(x) {
  (1/(pi)* (1/(1+x^2/4)))
}

g3 <- function(x) {
  (1/sqrt(pi/2)*exp(-x^2/2))
}
```

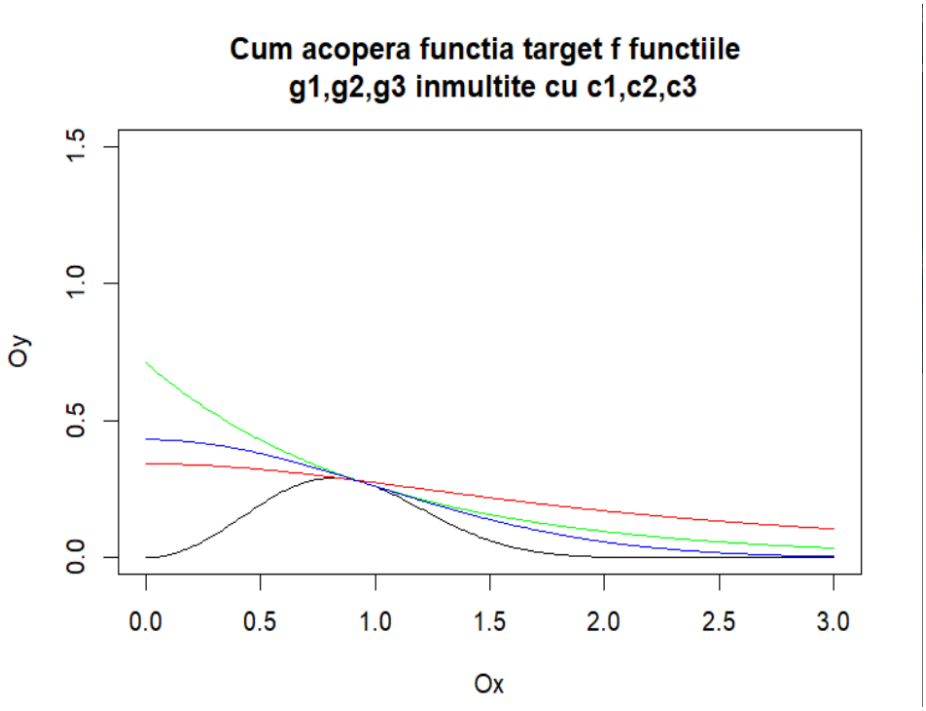
Aflăm c1, c2, c3 optimi(i.e. $c_i = \max_{x\in(0,\infty)} \frac{f(x)}{g(x)}$) folosind funcția optimize și luând a doua poziție din lista rezultată.

```r
h1 <- function(x){
  fb(x)/g1(x)}
c1 <- optimize(h1, lower=0,upper=10, maximum = T)
h2 <- function(x){
  fb(x)/g2(x)}
c2 <- optimize(h2, lower=0,upper=10, maximum = T)
h3 <- function(x){
  fb(x)/g3(x)}
c3 <- optimize(h3, lower=0,upper=10, maximum = T)

c1 <-as.numeric((c1)[2])
c2 <-as.numeric((c2)[2])
c3 <-as.numeric((c3)[2])
```

Câteva grafice:

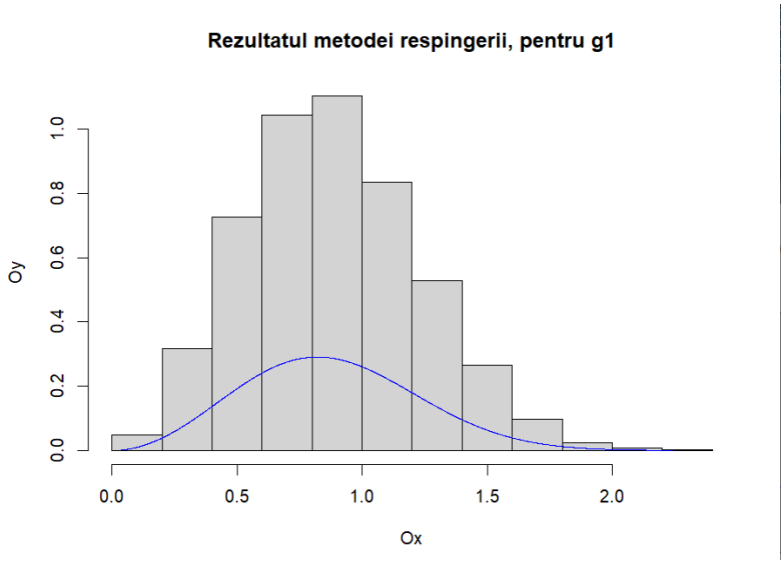




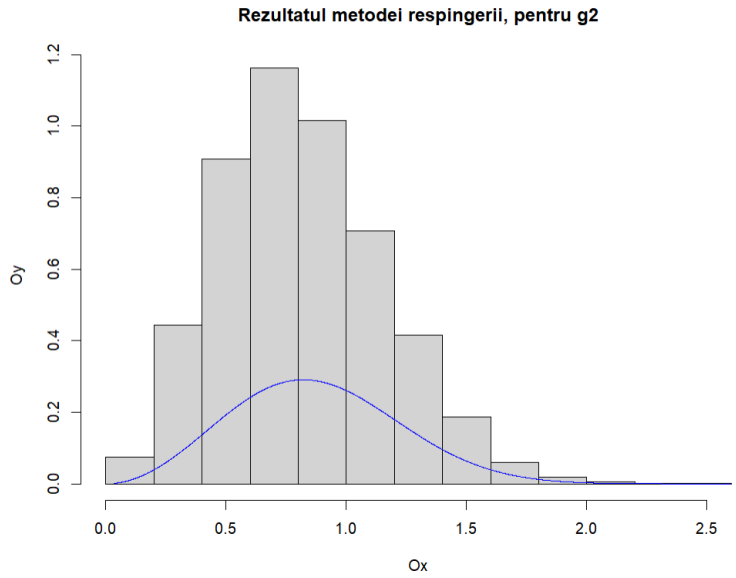


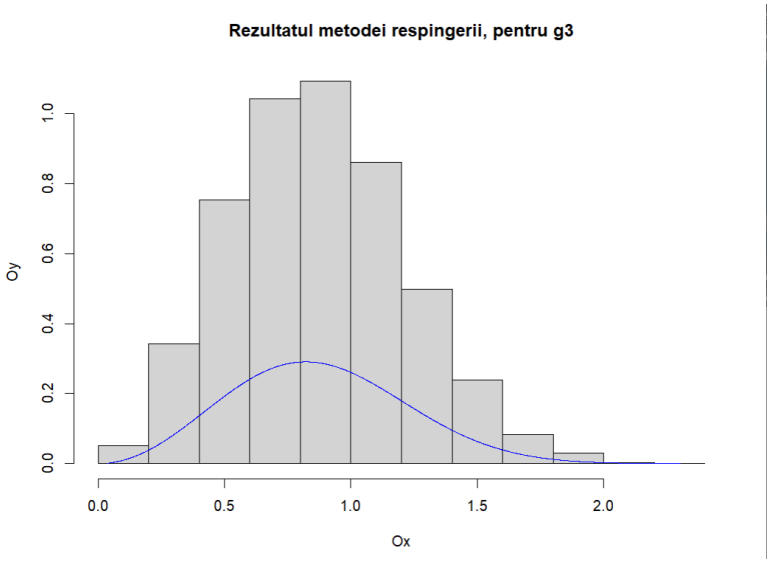
Apoi, aplicăm metoda respingerii:

```
t <- 10000


sf_1 <- function(x, n) {
  repeat {
    u <- runif(1, 0, 1)
    y <- rexp(1)
    if (u <= fb(y)/(c1*g1(y))) {
      return(y)
    }}}
X1 <- sapply(1:(t), sf_1)

x1 <- seq(min(X1), max(X1), 0.001)
hist(X1, freq = F,main='Rezultatul metodei respingerii, g1',xlab='Ox',ylab='Oy')

lines(x1, fb(x1), col = "blue")
```



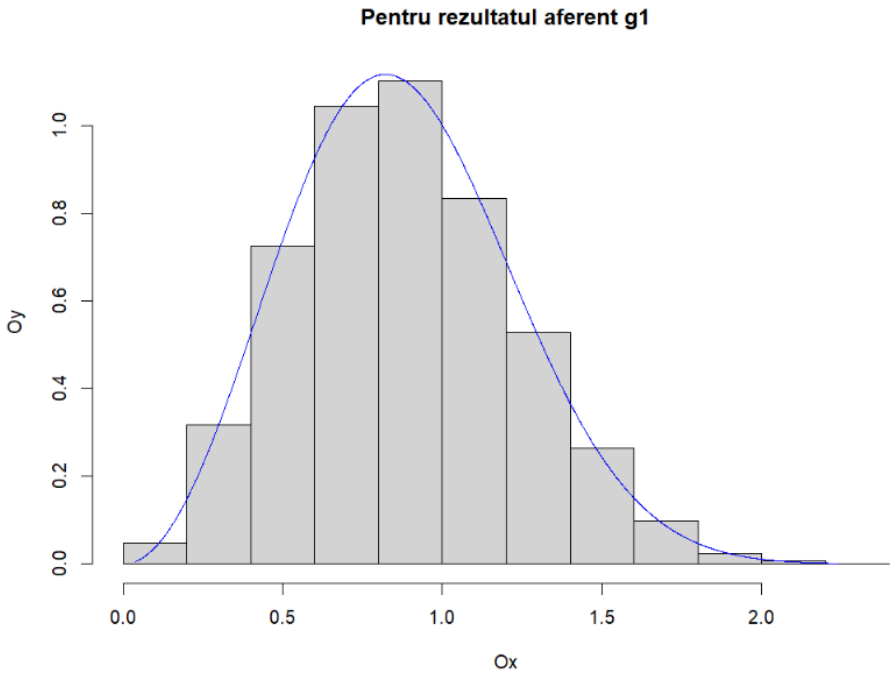
Pentru același cod dar cu g2, g3 în loc de g1:

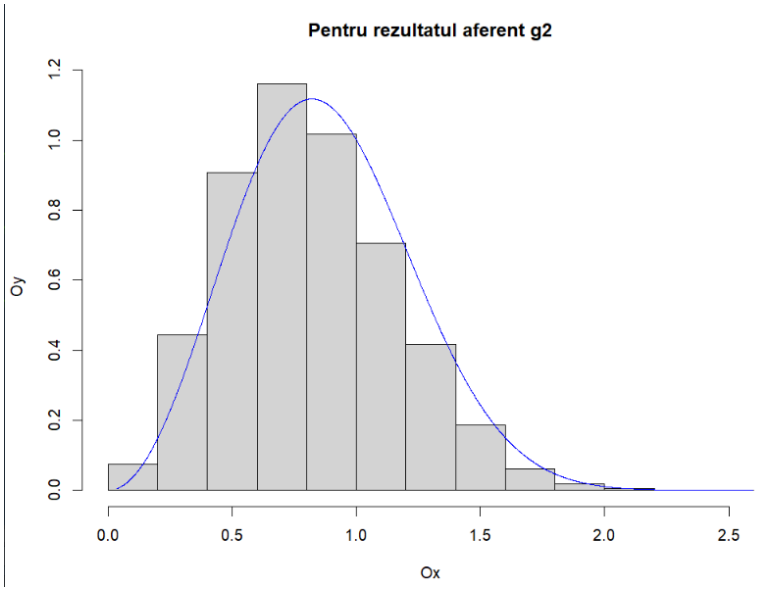


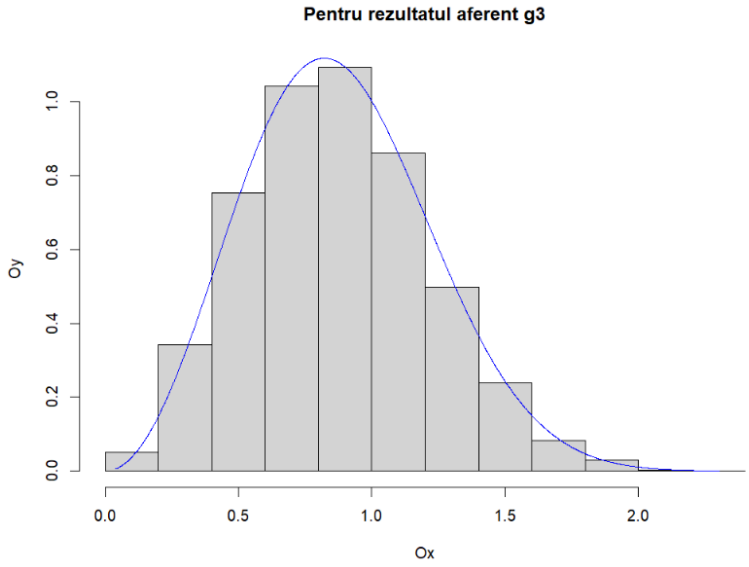


Se observă că funcția fb, în lipsa constantei de normalizare, nu se află deasupra histogramei. Astfel, o metodă de a deduce constanta de normalizare este să găsim o constantă care ar face graficul funcției să se potrivească cu histograma. În cazul nostru, acea constantă este aproximativ 3.844, care pare să fie în jur de $e\sqrt{2}$. Înmulțind funcția fb cu $e\sqrt{2}$,

```
f <- function(x) {
  exp(1)*2^0.5*(sin(x))^2*exp(-x^2*sqrt(x))
}

#Gasim ca
hist(X1, freq = F,main='Pentru rezultatul aferent g1',xlab='Ox',ylab='Oy')

lines(x1, f(x1), col = "blue")

hist(X2, freq = F,main='Pentru rezultatul aferent g2',xlab='Ox',ylab='Oy')

lines(x2, f(x2), col = "blue")

hist(X3, freq = F,main='Pentru rezultatul aferent g3',xlab='Ox',ylab='Oy')

lines(x3, f(x3), col = "blue")
```







Astfel, se potrivește bine pentru fiecare g.

III.

1) Pentru simularea lui T,vom avea nevoie sa simulam valori din distributiile:

$\text{Bin}(\alpha_i) \sim \begin{pmatrix} 0 & 1 \\ 1-\alpha_i & \alpha_i \end{pmatrix}$ si $\text{Exp}(\lambda_i)$

Folosim metoda inversa pentru simularea $\text{Bin}(\alpha_i)$

- Generam $U \sim \text{Unif}(0,1)$
- $X = \begin{cases} 0, & \text{daca } u < 1-\alpha_i \\ 1, & \text{daca } u \geq 1-\alpha_i \end{cases}$

Folosim metoda inversa pentru simularea $\text{Exp}(\lambda_i)$

- Generam $U \sim \text{Unif}(0,1)$
- $F(x) = \begin{cases} 1-e^{-\lambda_i x}, & x \geq 0 \\ 0, & \text{altfel} \end{cases}$
- $X = F^{-1}(U) \Rightarrow x = F^{-1}(u), \forall x \geq 0$
- $\Rightarrow u = F(x) \Leftrightarrow u = 1-e^{-\lambda_i x} \Leftrightarrow x = -\frac{1}{\lambda_i}\ln(1-u)$
- $X = -\frac{1}{\lambda_i}\ln(1-U)$

α_i, λ_i le vom genera uniform

Avand un examen cu un numar de n exercitii, algoritmul pentru simularea lui T este:

$T = 0$

pentru $i \in \overline{1,n}$:

generam $b := \text{Bin}(\alpha_i)$

daca $b = 0$:

STOP

altfel:

generam $t := \text{Exp}(\lambda_i)$

$T = T + t$

Algoritmul de mai sus genereaza o singura valoare pentru T,insa il putem scala in R pentru a genera 10^6 valori

```
b <- matrix(Bernoulli(p), nrow = n, ncol = max_n)
t <- matrix(Exponential(l), nrow = n, ncol = max_n)
```

b

Filter

	V1	V2	V3	V4
1	0	1	0	0
2	0	0	0	0
3	1	1	1	0
4	0	0	0	0
5	0	1	0	1

t

Filter

	V1	V2	V3	V4
1	11.733883442	14.767293019	89.233004647	3.407201e+01
2	1.465426997	0.301936768	3.793433059	4.654309e-01
3	3.780412018	5.936362983	0.279770850	3.286185e+00
4	3.348444445	1.102455008	0.006555691	8.412302e-03
5	9.012758122	22.699724217	5.249706733	4.623340e+00

n → numarul de sample-uri, i.e. 10^6

max_n → numarul de exercitii, i.e. n

p → un vector ce contine valorile $\alpha_1, \dots, \alpha_n$

l → un vector ce contine valorile $\lambda_1, \dots, \lambda_n$

b → o matrice care are pe fiecare coloanal cate 10^6 sample-uri din $\text{Bin}(\alpha_i),\ i \in \overline{1,n}$

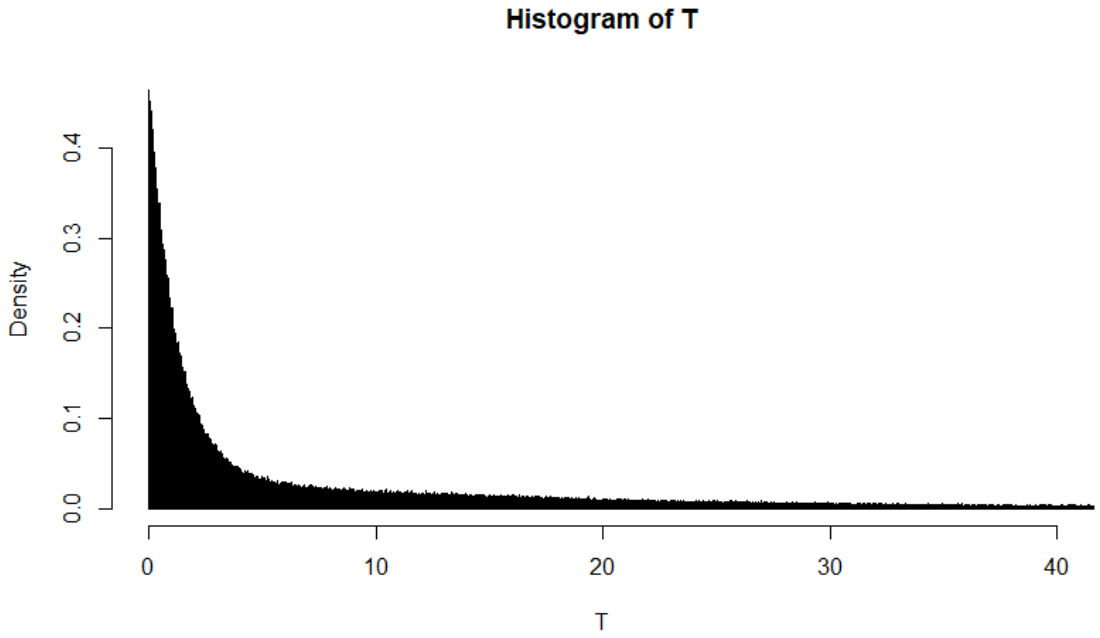
t → o matrice care are pe fiecare coloanal cate 10^6 sample-uri din $\text{Exp}(\lambda_i),\ i \in \overline{1,n}$

```
stop <- apply(b, MARGIN = 1, function(x) {
  # pastram indicii pozitiilor unde avem 0 in vectorul linie
  i <- which(x == 0)
  # daca nu exista 0, atunci elevul A rezolva tot examenul
  if (length(i) == 0){
    return(max_n)
  }
  # daca exista 0, atunci elevul A se opreste la primul 0
  return(i[1]-1)
})
```

Pentru fiecare sample din b cautam la al catelea exercitiu se opreste elevul din rezolvat.

```
t_stop <- cbind(t, stop)
# MARGIN=1  ->  functia va primia ca input fiecare linie din t_stop
T <- apply(t_stop, MARGIN = 1, function(x){
  # x[length(x)] = indicele de oprire
  # sumam toti timpi pana la indicele de oprire
  return(sum(x[1:x[length(x)]]))
})
```

Pentru fiecare sample din t facem suma primelor elemente, pana la exercitiul la care elevul se opreste din rezolvat.




```
Expected_value <- mean(T)
```

Expected_value	9.14371945041831

2) Pentru calculul exact al E(T) vom considera urmatoarele evenimente

H = evenimentul ca elevul sa termine examenul in timpul t

B = evenimentul ca elevul sa se opreasca din lucru la al n − lea exercitiu

Vrem sa aflam $P(H|B) = \frac{P(H \cap B)}{P(B)}$

$X_i \sim Exp(\lambda_i)$,

$$\sum_{i=1}^{n} X_i \sim Hypo(\lambda_1, \dots, \lambda_n) \Rightarrow P(H \cap B) = \sum_{i=1}^{n} \lambda_i e^{-x\lambda_i} \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{\lambda_j}{\lambda_j - \lambda_i}$$

$$P(B) = (1 - \alpha_n) \prod_{i=1}^{n} \alpha_i$$

$$E(T) = \int_{\mathbb{R}} P(H|B)\, d\lambda, \qquad \int_0^{\infty} e^{-x\lambda_i}\, dx = \frac{1}{\lambda_i}$$

$$E(T) = \frac{\sum_{i=1}^{n} \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{\lambda_j}{\lambda_j - \lambda_i}}{(1 - \alpha_n) \prod_{i=1}^{n} \alpha_i}$$

3) $P = \frac{\#\text{cazuri favorabile}}{\#\text{cazuri totale}} = \frac{\#\text{a finalizat activitate}}{10^6}$

In acest caz, finalziarea activitatii este echivalenta cu a nu se opri la niciun exercitiu.

```
p_finish <-sum(stop == max_n)/n
p_finish        0.133064
```

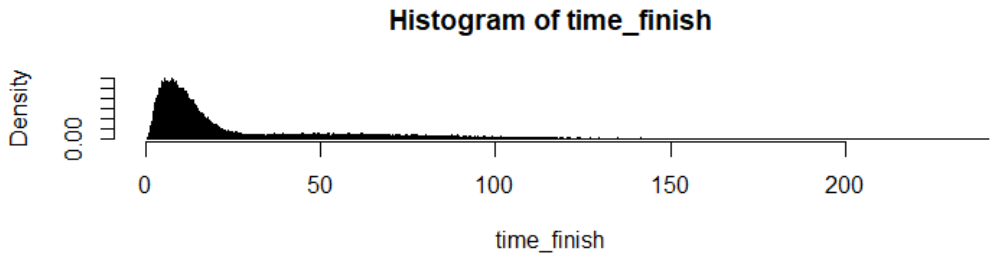
4) $P = \frac{\#\text{cazuri favorabile}}{\#\text{cazuri totale}} = \frac{\#\text{a finalizat activitate intr-un timp mai mic decat } \sigma}{10^6}$

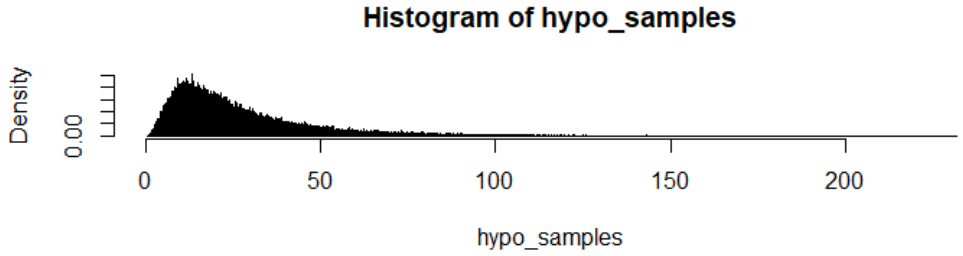
```
time <- runif(1, 1, 120)
p_finish_time <- sum(T[which(stop == max_n)] < time)/n
p_finish_time   0.10074
```

5) Vom selecta din T acele valori pentru care a fost finalziata activitatea

```
time_finish <- T[stop == max_n]
min_time <- min(time_finish)
max_time <- max(time_finish)

hypo_samples <- c()
for(i in 1:max_n){
  if(i==1){
    hypo_samples <- rexp(length(time_finish), l[i])
  } else{
    hypo_samples <- hypo_samples+rexp(length((time_finish)), l[i])
  }
}
max_time <- max(hypo_samples)

# plotam histogramele una sub celalta
par(mfrow=c(2,1))
hist(time_finish, xlim = c(min_time, max_time), breaks = 10^4, freq = FALSE)
hist(hypo_samples, xlim = c(min_time, max_time), breaks = 10^4, freq = FALSE)
```

Mai departe am generat o distributie hypoexponentiala cu aceasi parametri si acelasi numar de sample-uri.





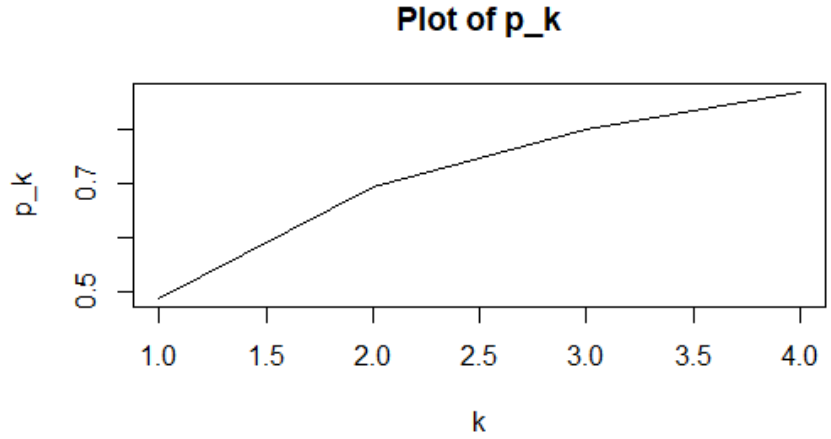
Observatie:

Finalizarea activitatii presupune rezolvarea tuturor exercitiilor, astfel ca pentru calculul lui T va trebui sa sumam toate variabilele aleatoare din distributiile exponentiala de parametri λ_i, acest lucru nu este nimic altceva decat distrbutia $\text{Hypo}(\lambda_1, \dots, \lambda_n)$

6) $P_k = \frac{\#\text{cazuri favorabile}}{\#\text{cazuri totale}} = \frac{\#\text{s-a oprit din lucru pana la exc k}}{10^6}$

```
p_k <- c()
for(k in 1:max_n){
  p_k <- c(p_k, sum(stop < k)/n)
}
plot(1:max_n, p_k, type="l", xlab="k", ylab="p_k", main="Plot of p_k")
```

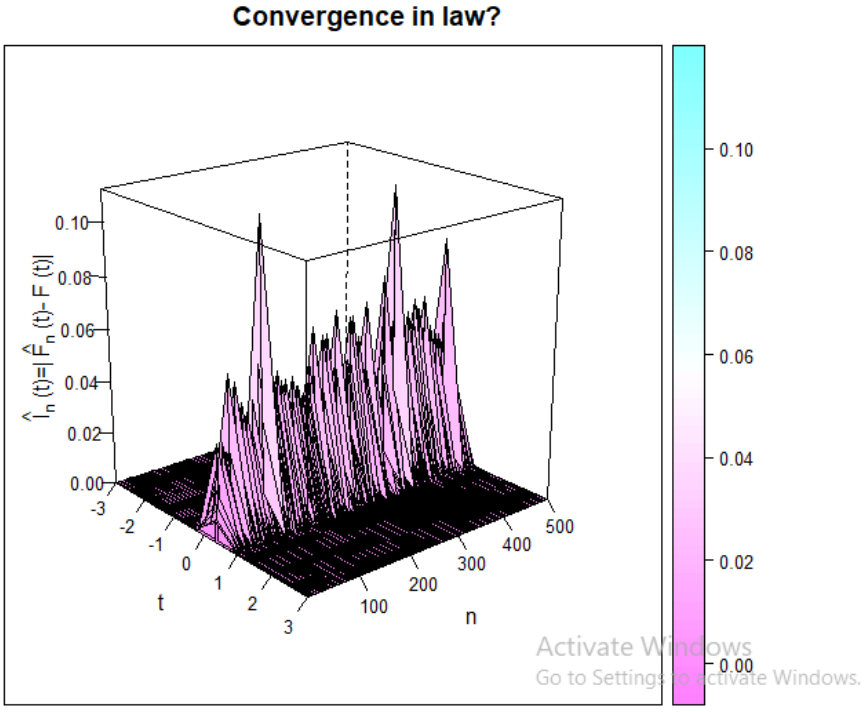
```
p_k                num [1:4] 0.49 0.692 0.8 0.867
```



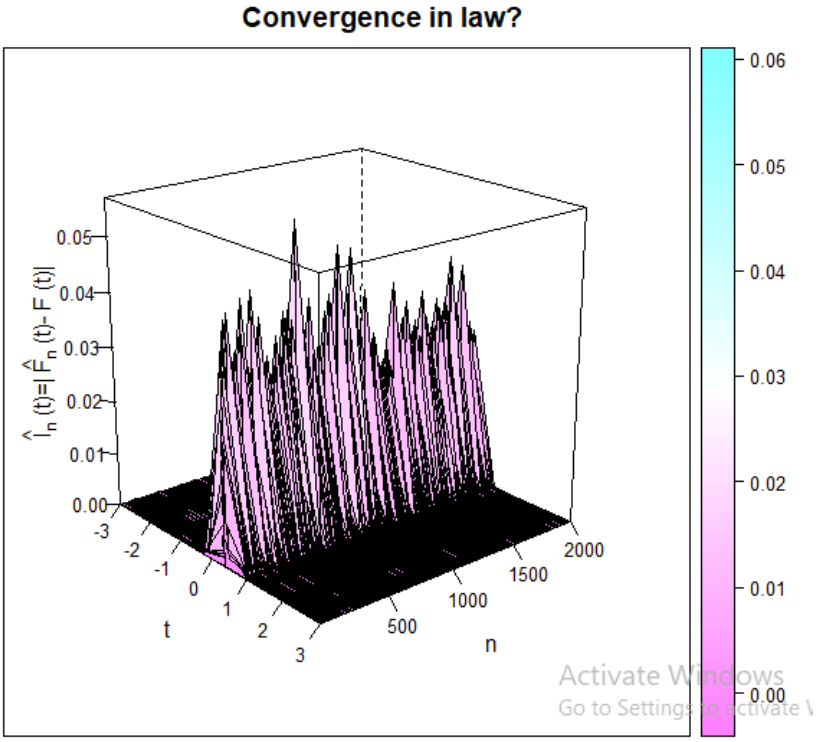
Exercitiul 4:

1) Fie X1,X2,….Xn v.a. i.i.d Xi~Beta(1 □ , 1 □) şi X~Binomial(1, 1 2).
(i) Verificaţi dacă Xn □ →X.

```
# Generam sirul Xn
genXn <- function(n)
{
  res <- rbeta(n, 1/n, 1/n)
  return(res)
}
# Folosim functia check.convergence in modul "L" pentru a verifica convergenta in distributie la
# X pe care il introducem prin functia de masa si prin cea de repartitie

check.convergence(500, 250, genXn, mode="L",
                  density = F,
                  densfunc = function(x){dbinom(x, 1, 1/2)},
                  probfunc = function(x){pbinom(x, 1, 1/2)})

check.convergence(2000, 1000, genXn, mode="L",
                  density = F,
                  densfunc = function(x){dbinom(x, 1, 1/2)},
                  probfunc = function(x){pbinom(x, 1, 1/2)})
```



Acesta este graficul pentru nmax = 500 si M = 250.



Acesta este graficul obtinut pentru nmax = 2000 si M = 1000.

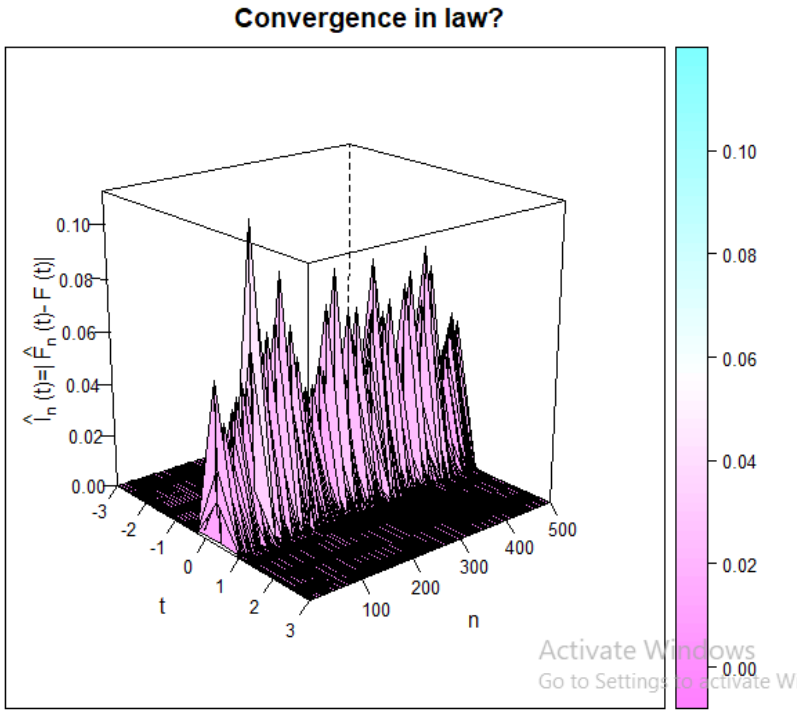
Putem observa ca de la un grafic la altul eroarea a scazut semnificativ, deci pentru un nmax suficient de mare si un M suficient de mare am obtine graficul erorii foarte aproape de planul orizontal zero.

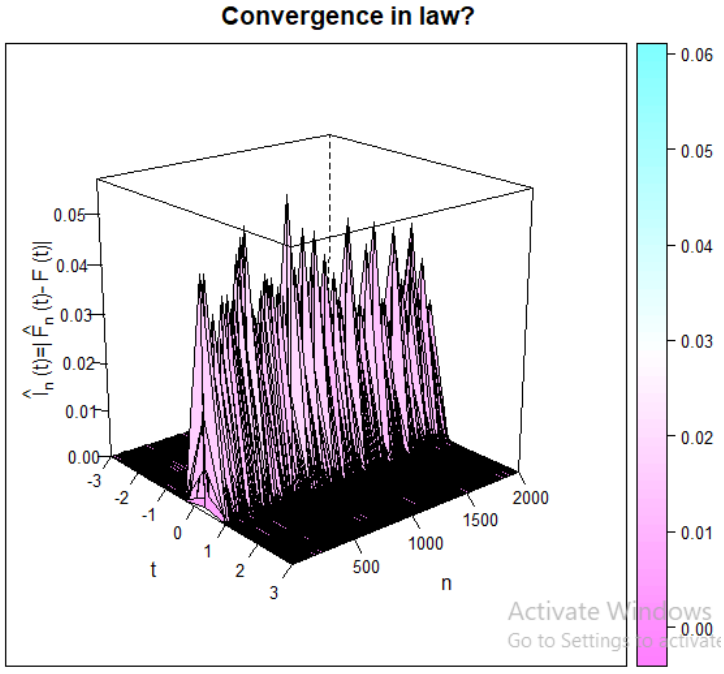
Prin urmare putem trage concluzia ca Xn converge in distributie la X.

(ii) Dar în cazul în care Xi~Beta(□/□ , □/□), cu a>0,b>0?

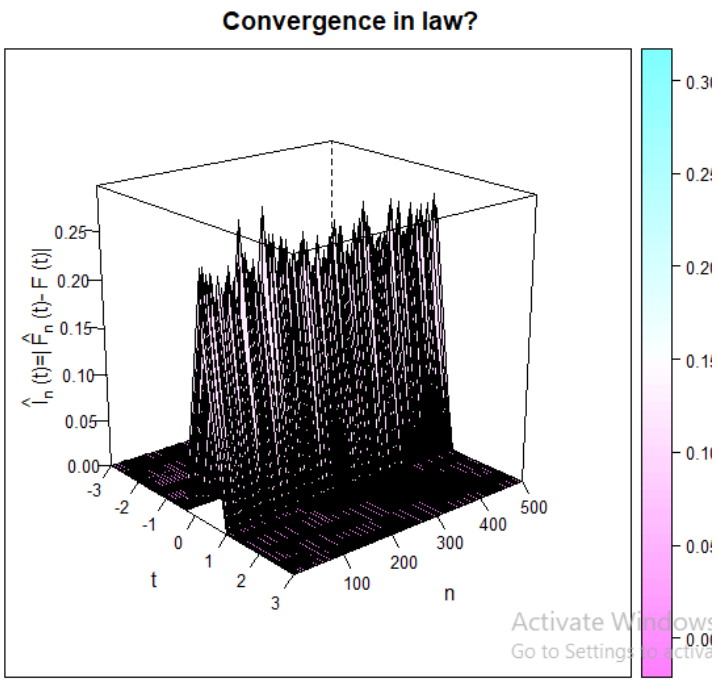
```
# Generam sirul de variabile aleatoare, de aceasta data ca o functie de n, a si b
genXn <- function(n, a, b)
{
  res <- rbeta(n, a/n, b/n)
  return(res)
}
# Aici trebuie sa completam cu argumentul argsXn unde adaugam a si b carora le dam niste valori
default

check.convergence(500, 250, genXn, argsXn=list(a = 5, b = 5), mode="L",
                  density = F,
                  densfunc = function(x){dbinom(x, 1, 1/2)},
                  probfunc = function(x){pbinom(x, 1, 1/2)})
check.convergence(2000, 1000, genXn, argsXn=list(a = 5, b = 5), mode="L",
                  density = F,
                  densfunc = function(x){dbinom(x, 1, 1/2)},
```

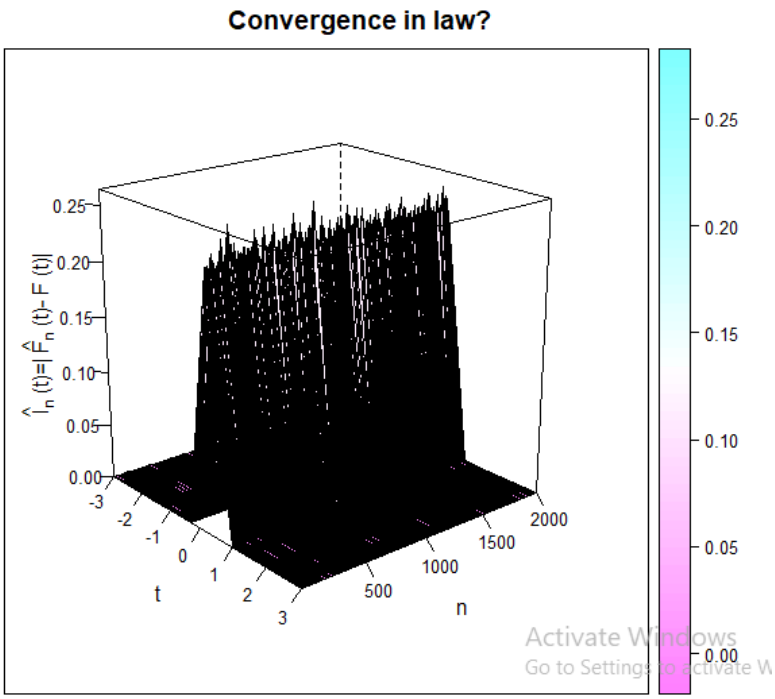
```
        probfunc = function(x){pbinom(x, 1, 1/2)})


check.convergence(500, 250, genXn, argsXn=list(a = 2, b = 5), mode="L",
        density = F,
        densfunc = function(x){dbinom(x, 1, 1/2)},
        probfunc = function(x){pbinom(x, 1, 1/2)})
check.convergence(2000, 1000, genXn, argsXn=list(a = 2, b = 5), mode="L",
        density = F,
        densfunc = function(x){dbinom(x, 1, 1/2)},
        probfunc = function(x){pbinom(x, 1, 1/2)})


check.convergence(500, 250, genXn, argsXn=list(a = 5, b = 2), mode="L",
        density = F,
        densfunc = function(x){dbinom(x, 1, 1/2)},
        probfunc = function(x){pbinom(x, 1, 1/2)})
check.convergence(2000, 1000, genXn, argsXn=list(a = 5, b = 2), mode="L",
        density = F,
        densfunc = function(x){dbinom(x, 1, 1/2)},
        probfunc = function(x){pbinom(x, 1, 1/2)})
```



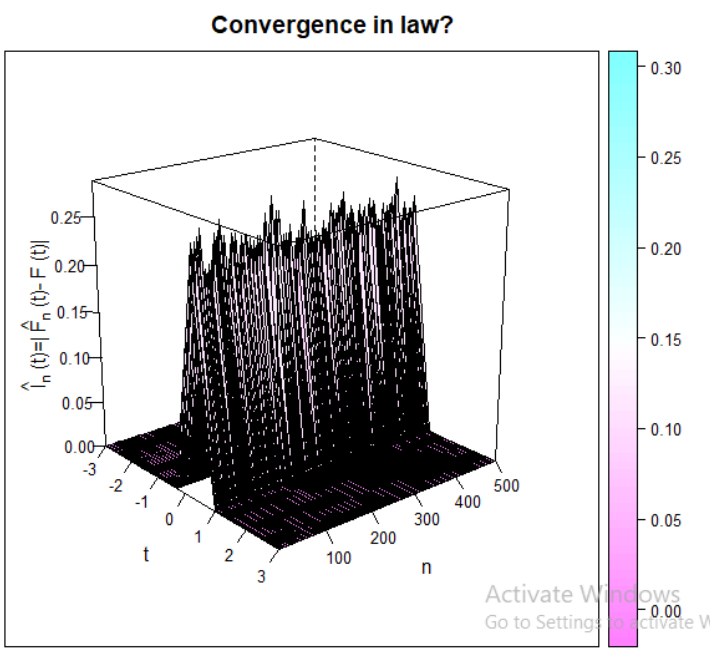


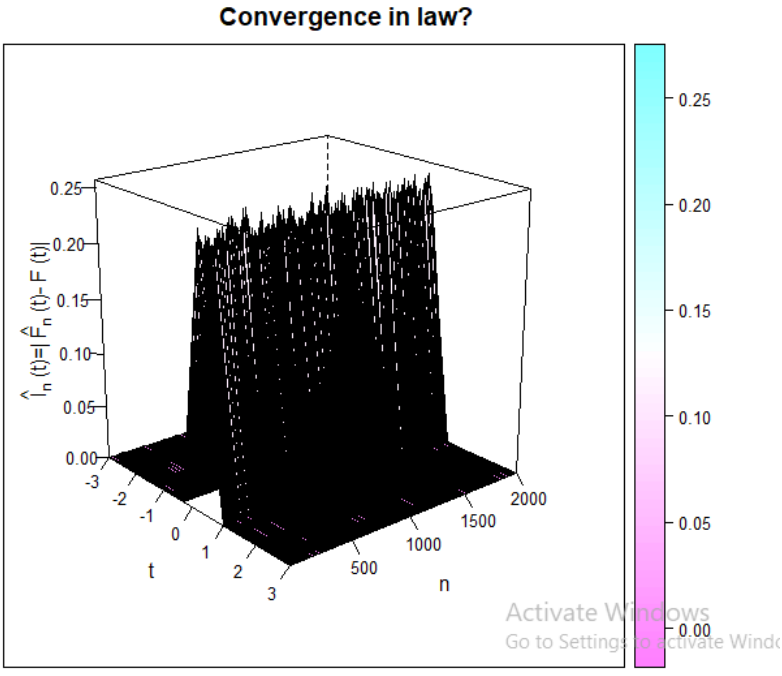
Pentru a si b egale (a = b = 5) obtinem ceva analog cu punctul (i). In stanga avem graficul pentru nmax =500 si M = 250, iar in dreapta graficul pentru nmax = 2000 si M = 1000. In acest caz Xn converge in distributie la X.





Pentru a = 2 si b = 5, eroarea ramane constanta, pentru (500, 250) si (2000, 1000) de unde tragem concluzia ca pentru a<b si Xi~Beta(□/□ , □/□), cu a>0,b>0, Xn nu converge in distributie la X~Binomial(1, 1 2).

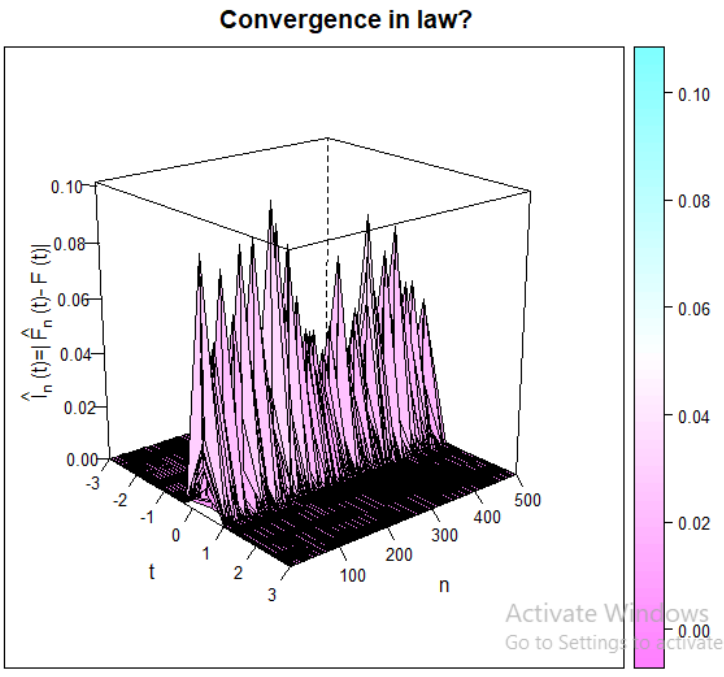


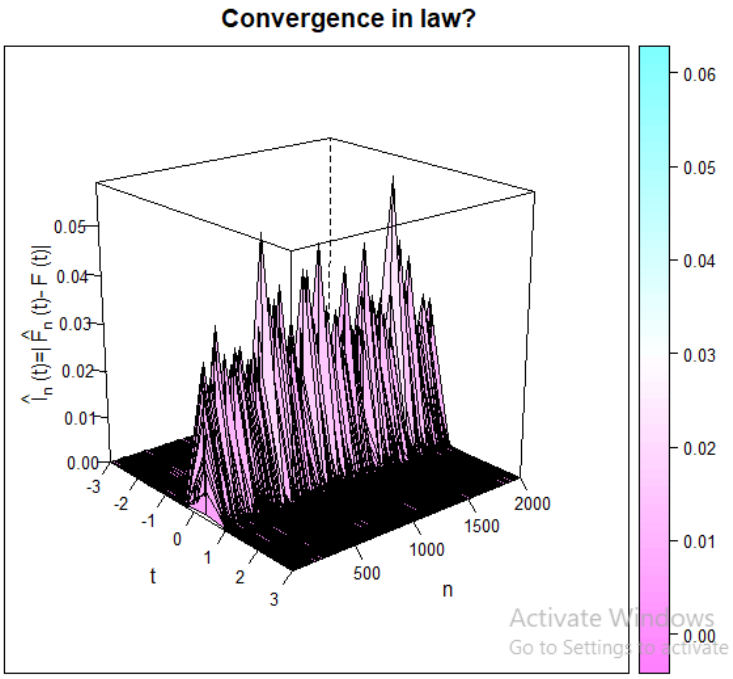


Pentru a = 5 si b = 2, obtinem acelasi lucru ca pentru a= 2 si b =5 si anume ca pentru a>b si Xi~Beta(□/□ , □/□), cu a>0,b>0, Xn nu converge in distributie la X~Binomial(1, 1 2).

2) Fie X1,X2,….Xn v.a. i.i.d uniform distribuite pe mulţimea de valori {1/□ , 2/□ … 1} şi X~Unif(0,1). (i) Verificaţi dacă Xn □ →X.

```
# Functia care ne genereaza valori cu aceeasi probabilitate
rUnifDisc <- function(n, discreteUnifValues) sample(discreteUnifValues, n, replace=T)
# Generam Xn pe care il vom folosi in check.convergence, uniform discret pe {1/n, 2/n, … , 1}
genXn <- function(n)
{
  discreteUnifValues <- seq(from = 1/n, to = 1, by = 1/n)
  res <- rUnifDisc(n, discreteUnifValues)
  return(res)
}
# Verificam convergenta in distributie la X uniform continua pe (0, 1)

check.convergence(nmax = 500, M = 250, genXn, mode="L",
                  density = F,
                  densfunc = function(x){dunif(x, 0, 1)},
                  probfunc = function(x){punif(x, 0, 1)})

check.convergence(nmax = 2000, M = 1000, genXn, mode="L",
                  density = F,
                  densfunc = function(x){dunif(x, 0, 1)},
                  probfunc = function(x){punif(x, 0, 1)})
```

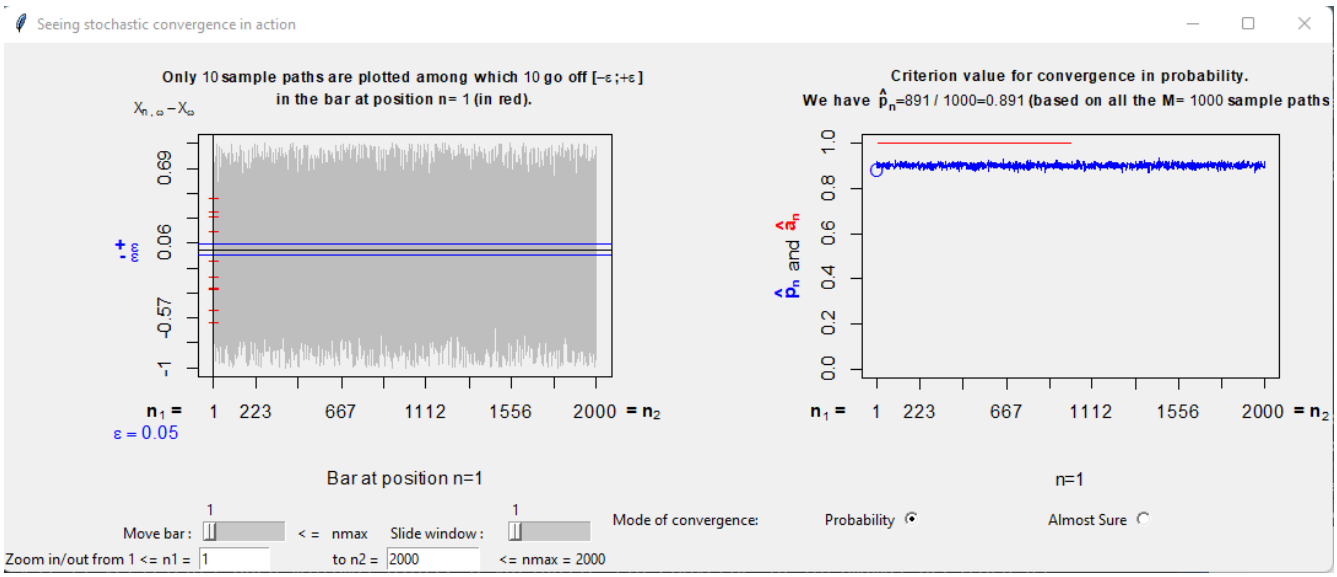




In graficul din stanga, am verificat convergenta pentru nmax = 500 si M = 250, iar in graficul din dreapta pentru nmax = 2000 si M = 1000. Observam ca diferenta dintre Fn(t) si F(t) a scazut odata cu cresterea lui n si M, asadar Xn converge in distributie la X~Unif(0,1).

(ii) Dar Xn □ →X?

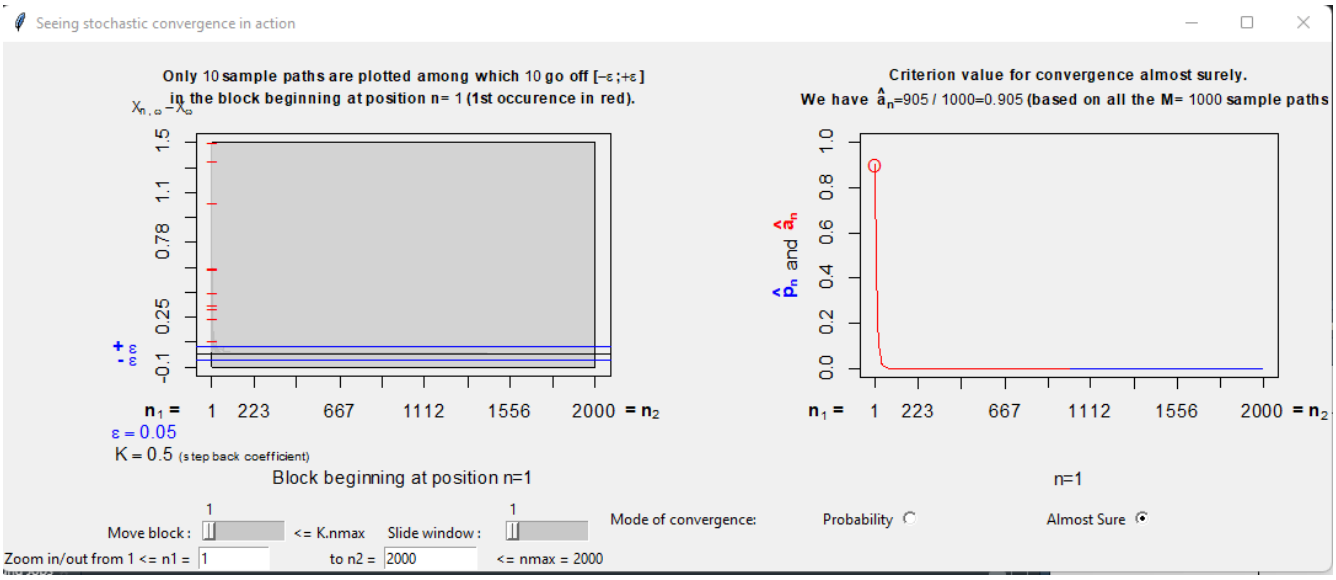
```
# Functia care ne genereaza valori cu aceeasi probabilitate pe intervalul discretizat anterior
rUnifDisc <- function(n, discreteUnifValues) sample(discreteUnifValues, n, replace=T)
# Generam un nou sir Xn, care este obtinut din Xn generat ca in subpunctul 2i)
# din care  scadem un X cu distributie uniforma pe (0, 1)
genXn <- function(n)
{ X <- runif(1)
  discreteUnifValues <- seq(from = 1/n, to = 1, by = 1/n)
  res <- rUnifDisc(n, discreteUnifValues) - X
  return(res)
}
check.convergence(nmax = 2000, M = 1000, genXn, mode="p")
```



A avea convergenta in probabilitate de la Xn la X este echivalent cu a avea convergenta in probabilitate de la Xn-X la 0. Prin urmare generatorul nostru este sub forma Xn-X, iar pentru a avea convergenta in probabilitate ar trebui sa observam in graficul din dreapta o descrestere spre 0 a lui pn. Pentru ca pn nu scade spre 0 putem concluziona ca Xn nu converge in probabilitate la X.

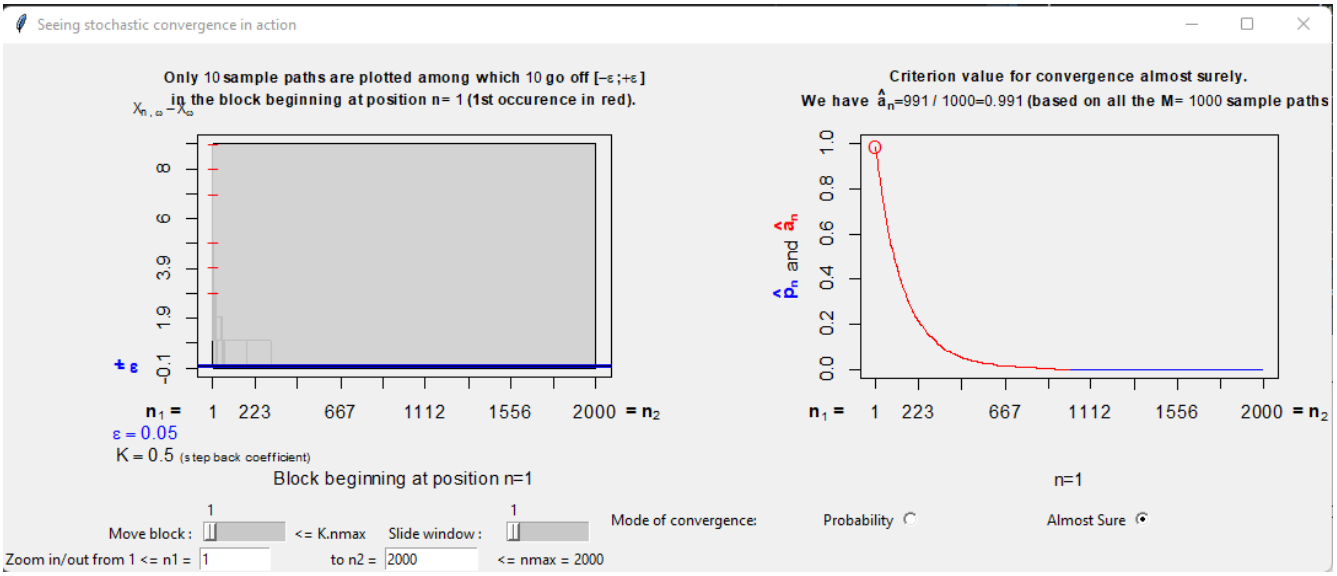
3) Fie X1,X2,….Xn v.a. i.i.d . Notăm cu m şi respectiv M infimumul şi respectiv supremumul mulţimii valorilor pe care le poate lua X1. (i.e. P(m≤X≤M)=1, P(X10 şi P(X1>b)>0 pentru orice a>m şi respectiv b
(i) Verificaţi că min{X1 , X2 …Xn} □.□. → m

```
# Luam Xi~Exp(2) si folosim cummin pentru a obtine repartitia min{X1 , X2 …Xn}
genXn <- function(n)
{
  xMin <- cummin(rexp(n, 2))
# Aleg ca infimum minimul repartitiei xMin obtinute
  infimum <- min(xMin)
# Verific daca xMin converge aproape sigur la infimum
  return(xMin - infimum)
}

check.convergence(nmax = 2000, M = 1000, genXn, mode = "as")
```



Observam ca pentru nmax = 2000 si M = 1000, in graficul din dreapta, an descreste spre 0, deci putem concluziona ca min{X1 , X2 …Xn} □.□. → m, pentru Xi~Exp(2). Verificam identic si cazul Xi~Pois(5).

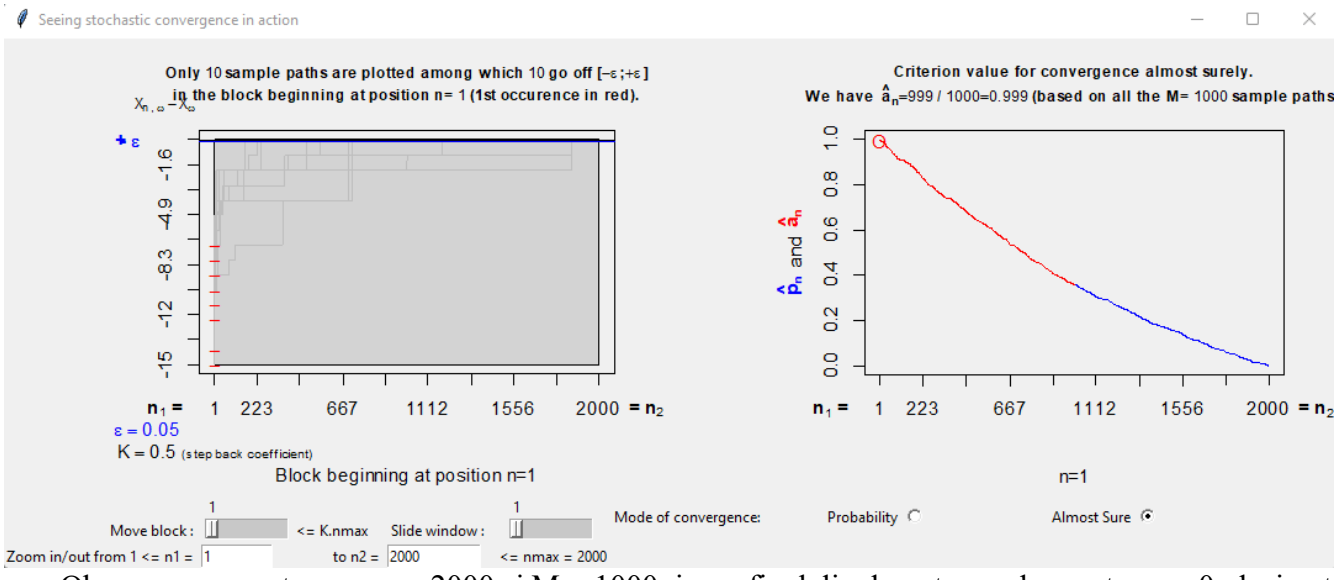
```
genXn <- function(n)
{
  xMin <- cummin(rpois(n, 5))
  infimum <- min(xMin)
  return(xMin - infimum)
}

check.convergence(nmax = 2000, M = 500, genXn, mode = "as")
```



Observam ca pentru nmax = 2000 si M = 1000, in graficul din dreapta, an descreste spre 0, deci putem concluziona ca min{X1 , X2 …Xn} □.□. → m, pentru Xi~Pois(5).

(ii) Verificati că max{X1 , X2 …Xn} □.□. → M.

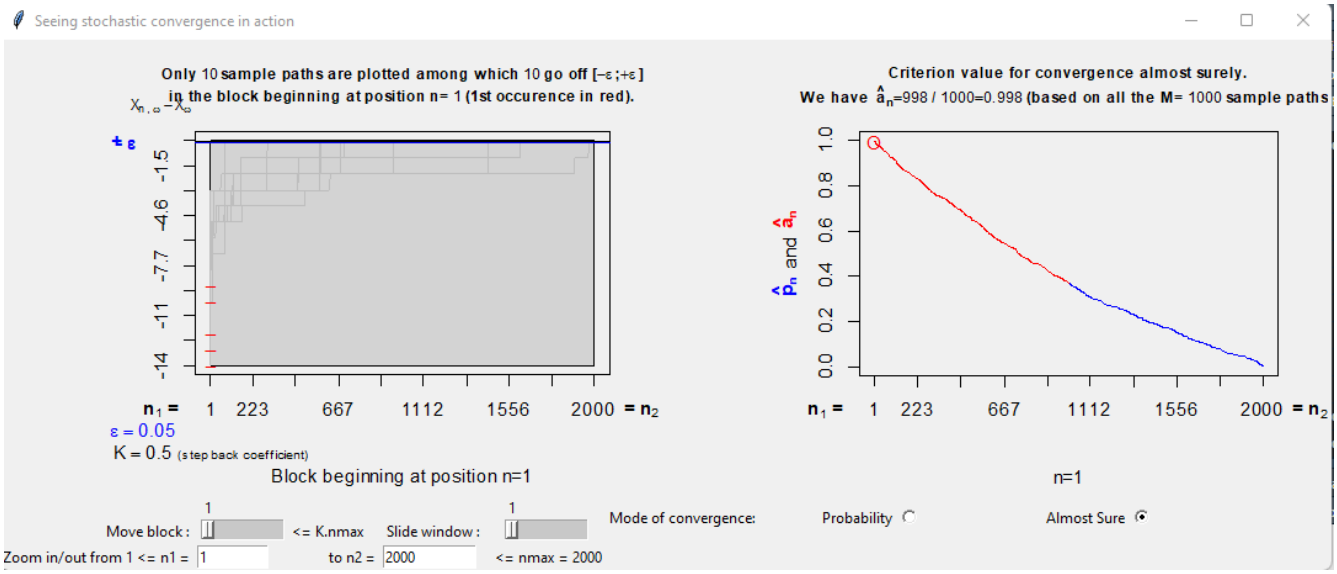
```
# Luam Xn exponentiale de 2,
genXn <- function(n)
{ # xMax va fi max{X1,X2,...Xn}
  xMax <- cummax(rexp(n, 2))
  # supremum luam valoarea maxima din xMax
  supremum <- max(xMax)
  # verificam convergenta aproape sigura a lui xMax la supremum
  return(xMax - supremum)
}
check.convergence(nmax = 2000, M = 1000, genXn, mode = "as")
```



Observam ca pentru nmax = 2000 si M = 1000, in graficul din dreapta, an descreste spre 0, deci putem

concluziona ca max{X1 , X2 …Xn} □.□. → M(supremumul, nu argunentul din check.convergence), pentru Xi~Exp(2).

#Verificam si pentru Xi~Pois(5).
genXn <- function(n)
{
  xMax <- cummax(rpois(n, 5))
  supremum <- max(xMax)
  return(xMax - supremum)
}
check.convergence(nmax = 2000, M = 1000, genXn, mode = "as")



Observam ca pentru nmax = 2000 si M = 1000, in graficul din dreapta, an descreste spre 0, deci putem concluziona ca max{X1 , X2 …Xn} □.□. → M(supremumul, nu argunentul din check.convergence), pentru Xi~Pois(5).