

# Simularea variabilelor aleatoare

Generarea unor valori numerice în conformitate cu o anumită repartiție (simularea unei variabile aleatoare) este o prelucrare frecvent utilizată atât în Statistică (de exemplu în tehnicile de eșantionare) cât și în modelarea stohastică și în implementarea algoritmilor aleatori (de tip Monte-Carlo).

Există două modalități principale de simulare a variabilelor aleatoare, și anume:

- prin utilizarea unor dispozitive fizice cum ar fi zarul, ruleta și detectorii de radiație nucleară;
- prin implementarea pe calculator a unor algoritmi de generare a unor valori numerice.

Generatoarele bazate pe dispozitive fizice au câteva dezavantaje cum ar fi: rezultatele sunt dificil de reprodus, iar funcționarea dispozitivelor poate fi afectată de evenimente externe. Din acest motiv, după apariția și dezvoltarea calculatoarelor, interesul s-a orientat înspre metodele de generare care se bazează pe capacitățile aritmetice ale acestora.

Metodele de simulare a variabilelor aleatoare cu ajutorul calculatorului nu sunt perfecte întrucât calculatorul în sine este un dispozitiv determinist, iar majoritatea metodelor se bazează pe secvențe deterministe de calcul. Din acest motiv, valorile generate cu ajutorul calculatorului sunt în realitate "pseudo-aleatoare". În continuare vom omite calificativul "pseudo" doar pentru simplitate.

Din punct de vedere al modului de proiectare, generatoarele pot fi clasificate în:

- generatoare de numere aleatoare uniform repartizate;
- generatoare de numere aleatoare neuniform repartizate.

De cele mai multe ori generatoarele din a doua categorie se construiesc pe baza celor din prima categorie.

În continuare vom prezenta algoritmi de generare a numerelor uniform repartizate și metode generale pentru simularea variabilelor neuniform repartizate, discrete sau continue.

## 1 Simularea variabilelor aleatoare uniform repartizate

### 1.1 Cazul variabilelor discrete

O variabilă aleatoare discretă,  $X$ , ce ia valori în mulțimea  $\{1, \dots, n\}$  este uniform repartizată dacă  $P(X = i) = 1/n$  pentru  $i = \overline{1, n}$ . Simularea unei astfel de variabile se bazează pe utilizarea unei funcții  $g : \mathcal{I}^k \rightarrow \mathcal{I}$ , unde  $\mathcal{I}$  este o submulțime a numerelor întregi care pot fi reprezentate în calculator. Pentru a genera o secvență de numere se pornește de la câteva valori inițiale  $x_1, \dots, x_k$  care formează *sămânța* generatorului (*seed*) și se folosește o relație de recurență:

$$x_n = g(x_{n-1}, \dots, x_{n-k}), \quad n > k \quad (1)$$

Întrucât mulțimea valorilor numerice ce pot fi reprezentate în calculator este finită rezultă că șirul definit de relația de recurență (1) este periodic. Pentru ca generatorul implementat pe baza acestei relații să fie acceptabil trebuie să îndeplinească cel puțin două condiții:

- Perioada lui să fie mare în raport cu numărul de valori generate.
- Valorile generate să nu fie secvențial corelate. Aceasta proprietate trebuie înțeleasă în modul următor: considerând secvențe de câte  $p$  valori succesiv generate se obține o ”umplere” uniformă a spațiului  $p$ -dimensional. În cazul prezenței corelației secvențiale, aceste ”puncte” din spațiul  $p$ -dimensional se grupează într-un număr redus de hiperplane.

De remarcat că importanța calității unui generator depinde de tipul aplicației unde este folosit (ea este mai puțin importantă în aplicațiile care utilizează numerele aleatoare doar pentru ilustrări grafice de efect și devine foarte importantă de exemplu în aproximarea integralelor prin metoda Monte-Carlo).

Satisfacerea condițiilor de mai sus poate fi asigurată printr-o bună alegere a funcției  $g$ . Cele mai frecvent utilizate metode (pe care se bazează și generatoarele incluse în limbajele de programare) sunt cele congruențiale. Acestea sunt caracterizate printr-o relație de recurență de forma:

$$x_n = f(x_{n-1}, \dots, x_{n-k}) \bmod m$$

unde  $f : \mathcal{I}^k \rightarrow \mathcal{I}$  este o funcție iar  $k$  și  $m$  sunt valori care definesc generatorul.

Dintre metodele congruențiale, cele mai utilizate sunt cele pentru care  $f$  este o funcție liniară:

$$x_n = (a_1 x_{n-1} + \dots + a_k x_{n-k} + c) \bmod m$$

cu  $a_1, \dots, a_k, c$  și  $m$  valori întregi care caracterizează generatorul. De regulă  $a_i, c \in \{0, \dots, m-1\}$ . Evident relația de recurență se completează cu un set de valori inițiale:  $x_1, \dots, x_k$ .

Se observă că aceste metode generează valori cuprinse între 0 și  $m-1$ , iar perioada poate fi cel mult  $m$ . Din acest motiv  $m$  se alege cât mai mare, fiind de regulă valoarea întreagă maximă care poate fi reprezentată în calculator. Mai dificilă este însă problema alegerii parametrilor  $a_1, \dots, a_k, c$ . Pentru aceștia nu există ”rețete” riguroase, ci mai degrabă există valori particulare care au fost determinate în urma a numeroase încercări și testări ale eficienței.

Cel mai adesea se utilizează generatori de ordin 1, a căror formă este  $x_{n+1} = (ax_n + c) \bmod m$ . În cazul particular în care  $c = 0$ , generatorul se numește congruențial multiplicativ iar dacă  $c \neq 0$  e numit congruențial mixt. Dacă  $a$  și  $m$  sunt alese adecvat, atunci un generator multiplicativ este la fel de bun ca unul mixt.

Câteva exemple de valori adecvate pentru parametrii unui generator de ordin 1 ( $k = 0$ ) sunt:

1.  $a = 16807, c = 0, m = 2^{31} - 1$ ;
2.  $a = 14^{29}, c = 0, m = 2^{31} - 1$ ;
3.  $a = 24298, c = 99991, m = 199017$ ;
4.  $a = 1993538837, c = 7261067085, m = 2^{35}$ .

Secvența de valori generate depinde de valoarea inițială ( $x_0$ ). Aceasta este de regulă stabilită de către utilizator prin proceduri specifice fiecărui limbaj de programare (de exemplu, procedura `randomize` din Pascal sau funcția `srand` din C). Dacă se dorește ca la fiecare relansare a programului să fie generată altă secvență de valori este indicat ca  $x_0$  să fie stabilită pornind de la ceasul intern al calculatorului.

Dacă  $m = 2$  atunci se obține un generator de biți conducând la o altă variantă de generare a numerelor întregi. Pentru o reprezentare pe 32 de biți, fiecare valoare întreagă,  $x_n$ , poate fi exprimată ca:

$$x_n = b_n^{(1)} b_n^{(2)} \dots b_n^{(32)}$$

iar fiecare bit  $b_n^{(i)}$  se generează cu regula:

$$b_n^{(i)} = (b_{n-p}^{(i)} + b_{n-(p-q)}^{(i)}) \bmod 2$$

în care  $p > q > 0$ , iar  $b_0^{(i)}, \dots, b_p^{(i)}$  sunt valori binare care asigură inițializarea generatorului.

Pentru ca generatorul să aibă proprietăți statistice bune, parametrii  $p$  și  $q$  se aleg astfel încât polinomul  $X^p + X^q + 1$  să fie prim în  $Z_2[X]$ . Un exemplu de alegere a parametrilor este  $p = 98$  și  $q = 27$ . În acest caz perioada generatorului este  $2^p - 1$ .

## 1.2 Cazul variabilelor continui

Pentru a simula o variabilă aleatoare uniform repartizată în intervalul  $[0, 1]$  este suficient să se genereze valori întregi uniform repartizate pe mulțimea  $\{0, \dots, m-1\}$  și să se împartă la  $m-1$ . Dacă se dorește simularea unei variabile uniform repartizate pe  $(0, 1)$  este suficient să se genereze valori în  $1, \dots, m-1$  și să se împartă la  $m$ . De observat că un generator mixt nu produce niciodată valoarea 0.

Pe de altă parte, se consideră că un bun generator de valori uniform distribuite în intervalul  $[0, 1]$  nu trebuie să furnizeze niciodată valoarea 0 sau valoarea 1.

Pentru a simula o variabilă,  $V$ , uniform repartizată în intervalul  $(a, b)$  este suficient să se simuleze o variabilă,  $U$ , uniform repartizată în  $(0, 1)$ , după care să se facă transformarea  $V = (b-a)U + a$ .

Majoritatea metodelor de simulare a variabilelor neuniforme se bazează pe un generator de valori uniform repartizate în  $(0, 1)$ . În cele ce urmează vom considera o metodă de generare a valorilor uniform repartizate în  $(0, 1)$  ca fiind implementată de funcția `Random`. Este ușor de observat că o expresie de forma `1-Random` simulează deasemenea o variabilă uniform repartizată pe  $(0, 1)$ .

## 1.3 Detalii de implementare

De remarcat că unele dintre generatoarele liniar congruențiale de ordin 1 (exemplele 1, 2 și 4) nu pot fi implementate în mod direct în limbaje de nivel înalt întrucât valorile intermediare obținute prin calcule depășesc valorile maxime ce pot fi reprezentate ca întregi pe 32 de biți. În aceste condiții, majoritatea generatoarelor sunt implementate în limbaje de asamblare ce utilizează regiștrii de 64 de biți sau sunt utilizați algoritmi care produc doar valori intermediare ce nu depășesc  $2^{31} - 1$ .

În cazul simulării variabilelor continue, pentru a diminua efectul erorilor de rotunjire de la operația de împărțire este indicat să se folosească reprezentarea în dublă precizie.

Întrucât generatorii liniar congruențiali produc valori în care cifrele de rang scăzut (cele mai puțin semnificative) sunt mai puțin aleatoare decât cele de rang înalt este indicat ca atunci când se dorește simularea unei variabile uniforme pe  $\{1, \dots, s\}$  să se folosească câțul împărțirii și nu restul. De exemplu, în C se va folosi o expresie de forma `1+(int)(s*rand()/(RAND_MAX+1.0))` și nu una de forma `1+(rand()%s)`.

## 2 Metoda inversării funcției de repartiție

Fie  $X$  o variabilă aleatoare (discretă sau continuă) având funcția de repartiție  $F_X : \mathbb{R} \rightarrow [0, 1]$ . Se definește inversa  $F_X^{-1} : [0, 1] \rightarrow \mathbb{R}$  în modul următor:

$$F_X^{-1}(u) = \inf\{x \in \mathbb{R} | F_X(x) \geq u\}, \quad \forall u \in [0, 1].$$

Metoda inversării funcției de repartiție se bazează pe următorul rezultat:

*Fie  $F$  o funcție de repartiție. Dacă  $U$  este o variabilă aleatoare uniform repartizată în  $[0, 1]$ , atunci funcția de repartiție a variabilei aleatoare  $X = F^{-1}(U)$  este  $F$ .*

Într-adevăr, dacă notăm cu  $F_X$  funcția de repartiție a variabilei  $X = F^{-1}(U)$  atunci are loc:

$$F_X(x) = P(\{X < x\}) = P(\{F^{-1}(U) < x\}) = P(\{U < F(x)\}) = P(\{0 < U < F(x)\}) = F(x)$$

deci funcția de repartiție a lui  $X$  este chiar  $F$ .

Astfel forma generală a algoritmului de simulare a unei variabile având funcția de repartiție  $F_X$  este:

```
u:=Random
x:=InvF(u)          /* InvF este inversa functiei F */
Return x
```

Pentru repartiții concrete se obțin variante particulare ale algoritmilor, care diferă între ele prin modul de implementare a inversei funcției de repartiție.

## 2.1 Cazul repartițiilor discrete

Fie  $X$  o variabilă aleatoare discretă care ia valori în mulțimea  $\{x_1, x_2, \dots, x_n\}$ . Să presupunem că  $x_1 < x_2 < \dots < x_n$ , iar repartiția lui  $X$  este:

$$\begin{pmatrix} x_1 & \dots & x_i & \dots & x_n \\ p_1 & \dots & p_i & \dots & p_n \end{pmatrix}$$

Funcția de repartiție asociată va fi:

$$F_X(x) = \begin{cases} F_1 = 0, & x \leq x_1 \\ F_2 = p_1, & x_1 < x \leq x_2 \\ \dots \\ F_i = \sum_{k=1}^{i-1} p_k, & x_{i-1} < x \leq x_i \\ \dots \\ F_n = \sum_{k=1}^{n-1} p_k, & x_{n-1} < x \leq x_n \\ F_{n+1} = 1 & x > x_n \end{cases}$$

În acest caz valorile inversei funcției de repartiție pot fi calculate în modul următor:

$$F_X^{-1}(u) = x_i, \quad \text{dacă } F_X(x_{i-1}) < u \leq F_X(x_i), \quad \text{pentru } i = \overline{1, n}$$

cu  $x_0 = -\infty$  și  $F_X(x_0) = 0$ .

Dacă descriem funcția de repartiție prin tabelul:

$$\begin{pmatrix} x_0 = -\infty & x_1 & \dots & x_{i-1} & x_i & x_{i+1} & \dots & x_n \\ F_0 = 0 & F_1 & \dots & F_{i-1} & F_i & F_{i+1} & \dots & F_n \end{pmatrix}$$

atunci algoritmul de simulare constă în generarea unei valori,  $u$ , uniform repartizate în  $(0, 1)$  și în găsirea indicelui  $i$  pentru care  $F_{i-1} < u \leq F_i$ . Principala prelucrare a acestui algoritm este astfel căutarea intervalului din lista valorilor funcției de repartiție care conține valoarea  $u$ .

Structura algoritmului este:

```

i:=1
u:=Random
While (u>Fi) and (i<n) Do i:=i+1
Return xi

```

*Exemple.*

1. *Repartiția Bernoulli.* Fie  $X : \Omega \rightarrow \{0, 1\}$  o variabilă binară caracterizată de  $P(\{X = 0\}) = p$  și  $P(\{X = 1\}) = q = 1 - p$ . Funcția de repartiție este:

$$F_X(x) = \begin{cases} 0, & x \leq 0 \\ p, & 0 < x \leq 1 \\ 1, & x > 1 \end{cases}$$

În acest caz particular problema căutării este mult simplificată și algoritmul devine:

```

u:=Random
If u<=p then x:=0 else x:=1
Return x

```

2. *Repartiția binomială.* Fie  $X : \Omega \rightarrow \{0, 1, \dots, n\}$  o variabilă având funcția de repartiție:

$$F_X(x) = \begin{cases} 0, & x \leq 0 \\ P_0 + \dots + P_k, & k < x \leq k + 1, \quad k = \overline{0, n-1} \\ 1, & x > n \end{cases}$$

unde  $P_k = P(X = k) = C_n^k p^k (1-p)^{n-k}$ . Evident că  $X$  poate fi simulată prin construirea tabelului asociat funcției de repartiție și prin aplicarea metodei clasice de căutare. Pentru valori mari ale lui  $n$  metoda nu este eficientă. O altă variantă este de a simula "extragerile cu revenire" și de a număra de câte ori de produce evenimentul de probabilitate  $p$ . Pentru aceasta se consideră  $N$  ca fiind cel mai mic număr natural pentru care  $n1 = Np$  și  $n2 = Np$  sunt și ele numere naturale. Se construiește un tabel cu  $n1 + n2$  elemente având următoarele valori:  $t_1 = \dots = t_{n1} = 1$ ,  $t_{n1+1} = \dots = t_{n1+n2} = 0$ . Algoritmul pentru generarea unei valori  $x$  este:

```

x:=0
For k:=1 to n do
  {u:=Random
  i:=INT(N*u)+1
  x:=x+t[i]
  }
Return x

```

3. *Repartiția Poisson.* Fie  $X : \Omega \rightarrow \{0, 1, 2, \dots\}$  o variabilă aleatoare caracterizată prin

$$P(X = k) = \frac{\lambda^k}{k!} e^{-\lambda}, \quad \text{cu } \lambda > 0.$$

Funcția de repartiție este:

$$F_X(n) = \sum_{k=0}^{n-1} P(X = k), \quad n \geq 1.$$

În acest caz este ineficient să se genereze un tabel al funcției de repartiție (întrucât tabelul este infinit și nu se cunoaște a priori câte elemente trebuie generate), fiind mai indicat ca evaluarea acesteia să se facă pe măsură ce se efectuează căutarea intervalului care conține valoarea  $u$ . Această variantă a algoritmului are forma:

```
u:=Random
i:=0; p:=exp(-lambda); f:=p
While (u>f) do {i:=i+1; p:=p*lambda/i; f:=f+p;}
Return i
```

## 2.2 Cazul repartițiilor continue

În cazul variabilelor continue există două situații:

- se cunoaște expresia analitică a inversei funcției de repartiție;
- se folosește un tabel cu valori ale funcției de repartiție.

În primul caz se folosește forma generală a algoritmului, iar în al doilea se aplică algoritmul specific repartițiilor discrete, însă după ce s-a determinat intervalul  $(F_{i-1}, F_i]$  care conține pe  $u$  se calculează:

$$x^* = x_{i-1} + (x_i - x_{i-1}) \frac{u - F_{i-1}}{F_i - F_{i-1}}$$

aceasta fiind valoarea care se returnează și nu  $x_i$ . În locul ultimei relații, care asigură o interpolare liniară a valorilor din tabelul funcției de repartiție, se poate aplica o formulă corespunzătoare unui alt tip de interpolare.

### Exemple.

1. *Repartiția exponențială.* Fie  $X : \Omega \rightarrow [0, \infty)$  o variabilă aleatoare având funcția de repartiție:

$$F_X(x) = 1 - e^{-\lambda x}, \quad \lambda > 0.$$

Inversa acestei funcții este  $F_X^{-1}(u) = -\frac{1}{\lambda} \ln(1 - u)$ . Întrucât dacă  $u$  este uniform repartizată în  $(0, 1)$  atunci și  $1 - u$  este uniform repartizată în  $(0, 1)$ , rezultă că algoritmul de simulare are forma:

```
u:=Random
Return -ln(u)/lambda
```

2. *Repartiția Weibull.* Fie  $X : \Omega \rightarrow [0, \infty)$  o variabilă aleatoare având funcția de repartiție:

$$F_X(x) = 1 - e^{-ax^b}, \quad a, b > 0.$$

Inversa lui  $F_X$  este  $F_X^{-1}(u) = \left(-\frac{1}{a} \ln(1 - u)\right)^{1/b}$  astfel că algoritmul de simulare este:

```
u:=Random
Return (-ln(u)/a)^(1/b)
```

*Observație.* Metoda inversării funcției de repartiție poate fi extinsă și în cazul variabilelor multi-dimensionale. Pe o astfel de extindere se bazează metoda Box-Muller de simulare a variabilelor normale. Această metodă va fi prezentată în secțiunea dedicată variabilelor cu repartiție normală.

### 3 Metoda respingerii

Fie  $X$  o variabilă aleatoare pentru care se cunoaște un algoritm de simulare și fie  $E$  un eveniment de probabilitate nenulă. Să presupunem că repartiția variabilei  $Y$  are proprietatea:

$$F_Y(y) = P(\{Y < y\}) = P(\{X < y\}|E). \quad (2)$$

Astfel o valoare generată în conformitate cu repartiția lui  $X$  poate fi considerată o realizare a variabilei  $Y$  dacă este satisfăcut evenimentul  $E$ . Realizările lui  $X$  care nu satisfac evenimentul  $E$  sunt *respinse*. Pornind de la această idee se poate formula un algoritm general de simulare a lui  $Y$ :

```
Repeat
  X:=GenerareX
Until E
Return X
```

Principalele probleme care trebuie rezolvate pentru a aplica metoda respingerii pentru simularea unei variabile  $Y$  sunt:

- găsirea variabilei  $X$ ,
- stabilirea evenimentului  $E$ ,

astfel încât să fie satisfăcută relația (2). În plus este de dorit ca în aplicarea algoritmului, numărul mediu de respingeri ale valorilor generate să nu fie prea mare. Numărul mediu de respingeri este cu atât mai mic cu cât  $P(E)$  e mai apropiat de 1.

Un caz particular, destul de frecvent întâlnit, în care metoda respingerii poate fi aplicată cu succes este cel al simulării repartițiilor uniforme multidimensionale pe domenii oarecare pornind de la repartiții uniforme pe domenii paralelipipedice.

Presupunem că  $Y$ , variabila care trebuie simulată, are repartiția uniformă pe domeniul  $D \subset R^n$ . Fie  $D' = [a_1, b_1] \times [a_2, b_2] \times \dots \times [a_n, b_n]$  cel mai mic domeniu paralelipedic care conține domeniul  $D$ , iar  $X$  o variabilă uniform repartizată pe  $D'$ . Considerând evenimentul  $E = \{X \in D\}$ , variabila  $Y$  poate fi simulată în modul următor:

```
Repeat
  /* generarea componentelor vectorului u din D' */
  For i:=1 to n Do ui=ai+(bi-ai)*Random
Until (u1, u2, ..., un) in D
```

Se observă că probabilitatea evenimentului  $E$  este  $\text{vol}(D)/\text{vol}(D')$ . Algoritmul va fi cu atât mai eficient cu cât raportul ariilor este mai apropiat de 1. Pe de altă parte, complexitatea algoritmului depinde de forma domeniului  $D$  întrucât aceasta influențează complexitatea condiției de apartenență.

*Exemple.*

1. *Generarea unor puncte uniform repartizate în interiorul cercului unitate:*

```
Repeat
  x:=2*Random-1; y:=2*Random-1;
Until (x*x+y*y<1)
```

2. *Generarea unor puncte uniform repartizate în interiorul unui tor centrat în origine cu raza mare egală cu 4 și cea mică egală cu 2:*

```
Repeat
x:=-4+8*Random; y:=-4+8*Random; z:=-1+2*Random;
Until (z*z+Sqr(Sqrt(x*x+y*y)-3)<1)
```

Să considerăm problema mai generală a simulării variabilei  $Y$  având densitatea  $f_Y$  pornind de la variabila  $X$  cu densitatea  $f_X$ . Presupunem că există  $c \geq 1$  astfel încât

$$f_Y(x) \leq cf_X(x), \quad \forall x \in \mathbb{R}$$

Intuitiv metoda respingerii constă în acest caz în generarea de puncte uniform repartizate în domeniul  $D'$  delimitat de axa  $Ox$  și graficul funcției  $cf_X$  și în acceptarea valorii abscisei doar dacă punctul aparține și domeniului  $D$  determinat de axa  $Ox$  și graficul funcției  $f_Y$ .

Astfel algoritmul de generare este:

```
Repeat
x:=GenerareX;
u:=Random;
Until c*f_X(x)*u<=f_Y(x);
```

Se observă că evenimentul  $E$  este  $\{cf_X(X)U \leq f_Y(X)\}$  unde  $X$  este variabilă aleatoare cu densitatea  $f_X$  iar  $U$  este o variabilă uniform repartizată în  $(0, 1)$ .

Arătăm că  $P(E) = 1/c$ . Într-adevăr:

$$\begin{aligned} P(E) &= P(\{cf_X(X)U \leq f_Y(X)\}) = \int_{\{(x,u)|cf_X(x)u \leq f_Y(x)\}} f_X(x)\mathbf{I}_{(0,1)}(u)dxdu = \\ &= \int_{\{x|f_X(x)>0\}} f_X(x)dx \int_0^{f_Y(x)/(cf_X(x))} \mathbf{I}_{(0,1)}(u)du = \int_{\{x|f_X(x)>0\}} \frac{f_Y(x)}{c}dx = \frac{1}{c}. \end{aligned}$$

Deci algoritmul este cu atât mai eficient cu cât  $c$  este mai apropiat de 1. Astfel este indicat ca  $c$  să se aleagă ca fiind cea mai mică valoare care verifică  $f_Y(x) \leq cf_X(x)$ , de exemplu  $c = \max_x f_Y(x)/f_X(x)$ .

Pentru a justifica utilizarea metodei trebuie să arătăm că pentru orice mulțime boreliană,  $B$ , are loc  $P(\{X \in B\}|E) = \int_B f_Y(x)dx$ . Într-adevăr:

$$\begin{aligned} P(\{X \in B\}|E) &= \frac{P(\{X \in B\} \cap E)}{P(E)} = c \int_{\{x \in B\} \cap \{cf_X(x) \leq f_Y(x)\}} f_X(x)\mathbf{I}_{(0,1)}(u)dxdu = \\ &= c \int_{\{x \in B\} \cap \{u \leq f_Y(x)/(cf_X(x))\}} f_X(x)\mathbf{I}_{(0,1)}(u)dxdu = c \int_{\{x \in B\}} f_X(x) \int_0^{f_Y(x)/(cf_X(x))} \mathbf{I}_{(0,1)}(u)du = \int_B f_Y(x)dx. \end{aligned}$$

*Exemple.*

1. *Simularea repartiției exponențiale trunchiată la  $(0, 1)$ .* Fie  $Y$  variabila având densitatea de repartiție:

$$f_Y(x) = \begin{cases} \frac{e^{-x}}{1-e^{-1}}, & \text{dacă } x \in (0, 1) \\ 0 & x \notin (0, 1) \end{cases}$$

Fie  $X$  variabilă cu densitatea de repartiție  $f_X(x) = e^{-x}$  pentru  $x \geq 0$  și egală cu 0 în rest. Atunci

$$c = \max_{x \geq 0} \frac{f_Y(x)}{f_X(x)} = \frac{e}{e-1}.$$

Folosind pentru simularea variabilei  $X$  (cu repartiție exponențială) algoritmul prezentat în secțiunea anterioară, variabila  $Y$  poate fi simulată prin:

```
Repeat
  x:=-ln(Random)
  u:=Random
Until u<1/e
```

2. *Simularea repartiției seminormale.* Fie  $Y$  o variabilă cu densitatea de repartiție:

$$f_Y(x) = \sqrt{\frac{2}{\pi}} e^{-\frac{x^2}{2}}, \quad \text{dacă } x > 0$$

Considerăm  $X$  o variabilă cu repartiția exponențială:

$$f_X(x) = e^{-x}, \quad \text{dacă } x > 0.$$

Se observă că:

$$c = \max_x \frac{f_Y(x)}{f_X(x)} = \sqrt{\frac{2e}{\pi}}$$

deci algoritmul poate fi descris astfel:

```
Repeat
  x:=-ln(Random)
  u:=Random
Until ln(u)<=-(x-1)*(x-1)/2
```

3. *Simularea abscisei unui punct uniform repartizat în discul unitate.* Variabila de simulat,  $Y$ , are densitatea

$$F_Y(x) = \frac{2}{\pi} \sqrt{1-x^2} \mathbf{I}_{[-1,1]}(x), \quad x \in R$$

Considerând  $X$  ca fiind o variabilă uniform repartizată în  $[-1, 1]$  ( $f_X(x) = 1/2 \mathbf{I}_{[-1,1]}$ ) se obține  $c = 4/\pi$ . Algoritmul poate fi scris în forma:

```
Repeat
  x:=2*Random-1
  u:=Random
Until (u*u<=1-x*x)
```

## 4 Metoda descompunerii

Se aplică atunci când funcția de repartiție a variabilei de simulat poate fi scrisă ca o combinație convexă a unor funcții de repartiție asociate unor variabile pentru care se cunosc metode de simulare.

Fie  $F_1, F_2, \dots, F_n$  funcții de repartiție și fie  $(p_1, p_2, \dots, p_n)$  o repartiție de probabilitate ( $0 \leq p_i \leq 1, \sum_{i=1}^n p_i = 1$ ).

Considerăm funcția de repartiție  $F(x) = \sum_{i=1}^n p_i F_i(x)$ . Pentru a simula o variabilă aleatoare în conformitate cu repartiția dată de  $F$  se aplică algoritmul:

*Pas1.* Se generează o valoare  $i \in \{1, 2, \dots, n\}$  în conformitate cu repartiția  $p$  (aplicând, de exemplu, metoda inversării funcției de repartiție).

*Pas2.* Se returnează o valoare generată prin metoda de simulare a variabilelor cu repartiția corespunzătoare lui  $F_i$

*Observație.* Metoda se aplică în același mod în cazul în care se cunosc funcțiile de densitate ale variabilelor  $X_i$ .

*Exemple.*

1. *Simularea repartiției normale.* Fie  $X$  o variabilă cu repartiția normală standard. Atunci  $|X|$  are repartiția seminormală. Ținând cont de faptul că repartiția normală este simetrică putem scrie:

$$F_X(x) = \frac{1}{2}F_{-|X|}(x) + \frac{1}{2}F_{|X|}(x),$$

astfel că algoritmul de simulare a lui  $X$  este:

```
u:=Random
x:="generare variabila cu repartitia seminormala"
If u<0.5 then Return -x else Return x
```

2. *Simularea variabilelor cu repartiție uniformă pe domenii decompozabile.* Fie  $X$  o variabilă uniform repartizată pe  $D$ , un domeniu mărginit. Presupunem că  $D = D_1 \cup D_2 \cup \dots \cup D_n$  și  $D_i \cap D_j = \emptyset$  pentru orice  $i \neq j$ . Pentru a aplica metoda descompunerii, se calculează  $p_i = \text{aria}D_i/\text{aria}D$  și se aplică algoritmul:

*Pas 1.* Se generează  $i$  în conformitate cu  $(p_1, \dots, p_n)$

*Pas 2.* Se generează o valoare uniform repartizată în  $D_i$  (utilizând de exemplu metoda respingerii).

## 5 Simularea variabilelor aleatoare normal repartizate

### 5.1 Metoda bazată pe teorema limită centrală

Conform teoremei limită centrală, dacă  $X_1, X_2, \dots, X_n$  sunt variabile aleatoare i.i.d. (independente și identic repartizate) cu  $M(X_i) = \mu$  iar  $D^2(X_i) = \sigma^2$  atunci șirul de variabile aleatoare  $Z_n = \sqrt{n}(\bar{X} - \mu)/\sigma$  tinde în distribuție (când  $n$  tinde la infinit) către  $Z \sim N(0, 1)$ .  $\bar{X}$  reprezintă  $(X_1 + \dots + X_n)/n$ .

Considerând  $X_1 \sim \mathcal{U}(0, 1)$  rezultă că  $\mu = 1/2, \sigma^2 = 1/12$ , deci

$$Z_n = \frac{(X_1 + \dots + X_n) - n/2}{\sqrt{n}/\sqrt{12}}.$$

Se observă că luând  $n = 12$  expresia lui  $Z_n$  devine foarte simplă ( $Z_{12} = X_1 + \dots + X_{12} - 6$ ) astfel că un algoritm care simulează repartiția normală este:

```
z:=-6;
For i:=1 to 12 do
  z:=z+Random
Return z
```

*Observație.* Această metodă este costisitoare din punct de vedere computațional întrucât pentru generarea unei valori  $z$  necesită 12 apeluri ale funcției `Random`.

## 5.2 Metoda Box-Muller

Metoda Box-Muller se bazează pe următorul rezultat teoretic:

*Propoziție.* Fie  $X \sim N(0, 1)$  și  $Y \sim N(0, 1)$  variabile independente și fie  $R$  și  $\theta$  două variabile aleatoare ce au proprietatea că  $X = R\cos(\theta)$  și  $Y = R\sin(\theta)$ . Atunci  $R$  și  $\theta$  sunt independente,  $R$  are repartiția Rayleigh ( $f_R(r) = r \exp(-r^2/2)$  pentru  $r > 0$  și 0 în caz contrar) iar  $\theta$  are repartiție uniformă pe  $[0, 2\pi)$ .

Din această propoziție rezultă că simulând o variabilă  $R$  cu repartiția Rayleigh și una  $\theta \sim \mathcal{U}_{(0,1)}$  putem genera două valori în conformitate cu repartiția normală standard.

Simularea lui  $\theta$  este evidentă, iar simularea lui  $R$  se poate face prin metoda inversării funcției de repartiție, căci:

$$F_R(r) = 1 - \exp(-r^2/2), r > 0 \text{ iar } F_R^{-1}(s) = \sqrt{-2 \ln(1 - s)}.$$

Astfel algoritmul de generare poate fi scris în modul următor:

```
u:=Random; v:=Random
r:=sqrt(-2*ln(u))
z1:=r*cos(2*pi*v)
z2:=r*sin(2*pi*v)
Return z1,z2
```

O variantă a acestui algoritm care evită folosirea funcțiilor trigonometrice și care se bazează pe generarea de puncte uniforme distribuite în interiorul cercului unitate este:

```
Repeat
  u:=2*Random-1; v:=2*Random-1; s=u^2+v^2
Until 0<s<1
r:=sqrt(-2*ln(s)/s)
z1:=r*u; z2:=r*v
Return z1, z2
```

*Observație.* La fiecare apel al oricăreia dintre variantele algoritmului se obțin două valori în conformitate cu repartiția normală standard.

## 6 Simularea variabilelor de tip $\chi^2$ , Student și Fisher

Simularea acestor variabile se bazează pe legăturile ce există între ele și repartiția normală standard. Presupunem că avem la dispoziție o funcție `normala` care generează valori în conformitate cu repartiția normală standard.

Pentru simularea unei variabile  $X \sim \chi^2(n)$  se poate folosi algoritmul:

```
x:=0
For i:=1 to n do x:=x+normala(0,1)
Return x
```

Pentru simularea unei variabile  $X \sim t(n)$  se poate folosi algoritmul:

```
z:=normala(0,1)
y:=0
For i:=1 to n do y:=y+normala(0,1)
x:=z*sqrt(n)/sqrt(y)
Return x
```

Pentru simularea unei variabile  $X \sim F(n_1, n_2)$  se poate folosi algoritmul:

```
y:=0
For i:=1 to n1 do y:=y+normala(0,1)
z:=0
For i:=1 to n2 do z:=z+normala(0,1)
x:=(y/n1)/(z/n2)
Return x
```

*Observație.* Algoritmii folosesc din fiecare apel al funcției `normala` ambele valori generate.

## Referințe

- [1] M.T. Boswell, S.D. Gore, G.P. Patil, C. Taillie, *The Art of Computer Generation of Random Variables*, in *Handbook of Statistics*, vol. 9 (ed. C.R. Rao), Elsevier Science Publ., 1993.
- [2] S.M. Ermakov, *Metoda Monte Carlo și probleme înrudite*, Ed. Tehnică, 1976.
- [3] R.E. Crandall, *Projects in Scientific Computation*, Springer Verlag, 1994.
- [4] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, *Numerical Recipes in C. The Art of Scientific Computing*, Cambridge University Press, 1992.
- [5] B. Ycart, *Simulation des modeles markoviens*, curs DESS d'Ingénierie Mathématique, Univ. J. Fourier, Grenoble, 1997.