Material suplimentar-curs 8.: Histograma observaţiilor asupra unei variabile aleatoare. Generatori de numere pseudo-aleatoare. Generatorul liniar congruenţial

0.1 Histograma observaţiilor asupra unei variabile aleatoare

Pentru a evidenţia aspectele practice legat de variabilele aleatoare discrete/continue, considerăm graficul unei densităţi de probabilitate şi divizăm domeniul său de definiţie prin puncte echidistante x_i, cu pasul h fixat. Probabilitatea ca variabila aleatoare X să ia valori în intervalul $[x_i, x_{i+1})$ este aria trapezului curbiliniu de baze segmentul $[x_i, x_{i+1})$ şi arcul de grafic deasupra acestui segment. Aria trapezului o putem aproxima cu aria dreptunghiului de bază $[x_i, x_{i+1})$ şi înălţime $f((x_i + x_{i+1})/2)$ (Fig.1, sus), fie cu aria dreptunghiului de bază $[x_i, x_{i+1})$ şi înălţime $f(x_i)$ (Fig.1, jos). Deci $P(x_i \leq X < x_{i+1}) = \int_{x_i}^{x_{i+1}} f(x)\, dx$=aria trapezului curbiliniu, este aproximativ aria dreptunghiului menţionat. Reuniunea tuturor dreptunghiurilor astfel construite se numeşte histogramă asociată densităţii de probabilitate. Histograma evidenţiază aproximativ distribuţia valorilor variabilei aleatoare în intervalele $[x_i, x_{i+1})$.

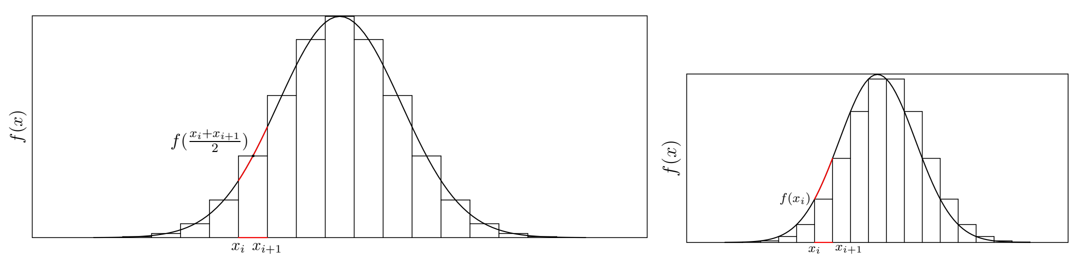

Fig.1: Histograma asociată unei densităţi de probabilitate

Precizarea distribuţiei de probabilitate a unei variabile aleatoare continue, se bazează de obicei pe istoricul observaţiilor asupra valorilor sale. În experimentele de laborator sau experimentele virtuale, de simulare, se înregistrează valorile observate, măsurate sau generate, ale unei variabile aleatoare, adică o listă de numere reale $x_1, x_2, \ldots, x_N$. Cel mai adesea nu se cunoaşte distribuţia de probabilitate a variabilei studiate. Informaţia primară se obţine asociind seriei de date înregistrate o histogramă cu m bare, m fixat apriori. Şi anume:

- se determină valoarea minimă, $xmin$, şi valoarea maximă, $xmax$, a seriei de date.
- Se divide segmentul $[xmin, xmax]$ prin puncte echidistante cu pasul,

$h = \dfrac{xmax - xmin}{m}$. Notăm cu y_j punctele de diviziune, $y_j = xmin + j * h$, $j = \overline{0, m}$.

• Se calculează numărul de valori, n_j, ale seriei de date, care aparţin intervalului

$$I_j = [y_j, y_{j+1}),\ j = \overline{0, m-2},$$
$$\text{respectiv intervalului } I_{m-1} = [y_{m-1}, xmax]$$

• Deoarece din cele N valori ale seriei de date, n_j cad în intervalul I_j, rezultă că am obţinut informaţia că probabilitatea ca variabila observată să ia valori în intervalul I_j este aproximativ:

$$P(y_j \leq X < y_{j+1}) \approx \frac{n_j}{N}$$

Dar această probabilitate este comparând cu cazul teoretic de mai sus aria dreptunghiului (a barei) ce are baza segmentul $[y_j, y_{j+1})$ • Rezultă astfel că deasupra intervalului I_j desenăm un dreptunghi de arie $A = n_j/N$. Dar aria este baza ori înalţimea dreptunghiului, H_j, şi deci din $A = Baza \times H_j$, rezultă că înălţimea dreptunghiului (a barei) este $H_j = A/B$. Cunoscând lungimea bazei ca fiind $h = y_{j+1} - y_j$, rezultă că înalţimea dreptunghiului este $H_j = \dfrac{n_j}{h\,N}$, $j = \overline{0, m-1}$.

Obiectul grafic rezultat din desenarea celor m dreptunghiuri (bare) se numeşte *histograma seriei de date* sau *histograma distribuţiei de frecvenţe*, deoarece ea ilustrează modul în care datele sunt distribuite în intervalele $[y_j, y_{j+1})$.

În Fig.2 sunt ilustrate histogramele a două serii de date.

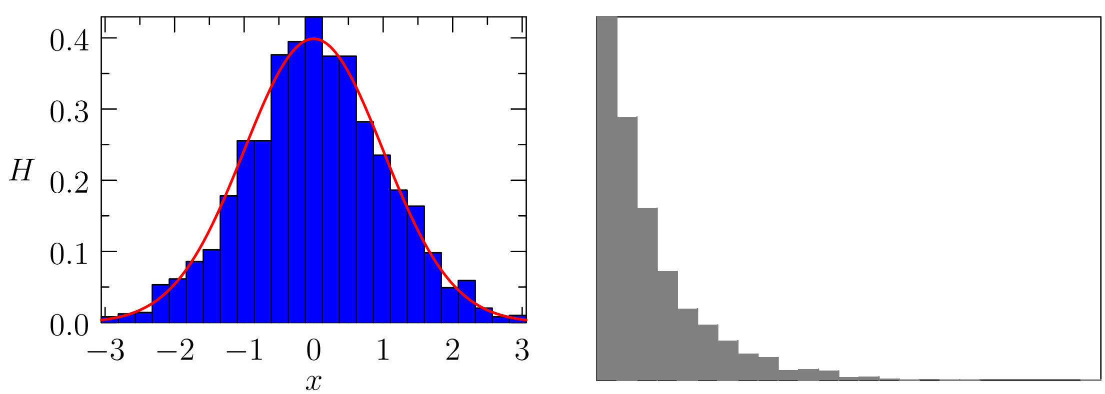

Fig.2: Histograme asociate la două serii de date numerice, rezultate din simularea unor variabile aleatoare ce au densitatea ilustrată în roşu.

0.2 Generatori de numere pseudo-aleatoare. Generatorul liniar congruenţial

A simula o variabilă aleatoare, X, revine la a rula un algoritm determinist care produce sau generează un şir de numere, $x_1, x_2, \dots, x_N$, ce au proprietăţile unui şir de valori de observaţii/măsurători independente asupra variabilei X.

Numerele astfel generate sunt pseudo-aleatoare şi se folosesc în probleme de simulare a proceceselor din sisteme complexe, cu intrări aleatoare. De asemenea, un generator de numere pseudo–aleatore este invocat în diverse etape ale algoritmilor probabilişti. Pe de alte parte, numerele pseudo-aleatoare se generează în scopul asigurării securităţii proceselor într-un sistem de operare sau pentru asigurarea securităţii transmiterii informaţiei pe un canal de comunicaţie.

Modalitatea de generare a numerelor pseudo–aleatoare depinde de scopul pentru care acestea sunt folosite. Noi abordăm doar problematica generării numerelor aleatoare pentru prima clasă de probleme enunţată, nu şi a celor folosite în criptografie.

Baza oricărei simulări a unui fenomen sau proces aleator o constituie numerele uniform distribuite pe intervalul $[0, 1)$.

Aleatorul este în general greu de definit, dar cel mai adesea un fenomen este considerat aleator dacă este imprevizibil şi nereproductibil. Cum un algoritm determinist nu poate genera şiruri de numere cu aceste calităţi, numerele generate se numesc pseudo–aleatoare, nu aleatoare. Algoritmii folosiţi în simularea variabilelor aleatoare se numesc *generatori de numere pseudo–aleatoare*.

Metoda de generare de numere aleatoare uniform distribuite, cel mai frecvent folosită până de curând, este metoda liniar congruenţială. Generatorul liniar congruenţial a fost introdus de Lehmer, fost profesor la Universitatea Berkeley, care a fost şi unul dintre fondatorii teoriei computaţionale a numerelor.

O astfel de metodă generează numere în inelul, $\mathbb{Z}_p$, al claselor de resturi modulo p, unde p este un număr natural fixat, numit *modul*. Dacă $n \in \mathbb{Z}$, $n \pmod p$ este restul împărţirii lui n la p. Teorema impărţirii cu rest din aritmetică, asigură că pentru $n \in \mathbb{Z}$, $p \in \mathbb{N} \setminus \{0\}$ există $q \in \mathbb{Z}$ şi $r \in \mathbb{N}$ astfel încât $n = qp + r$, $r \in \{0, 1, 2, \dots, p-1\}$. Prin urmare inelul $\mathbb{Z}_p$ conţine clasele de resturi $\{0, 1, 2, \dots, p-1\}$.

Generatorul liniar congruenţial produce un şir de numere $x_1, x_2 \dots, x_N$ din $\mathbb{Z}_p$ printr-o formulă recursivă:

$$x_n = ax_{n-1} + b \pmod p \tag{1}$$

unde parametrii a, b sunt fixaţi în $\mathbb{Z}_p$, iar x_0 se numeşte valoare iniţială sau *seed*. Cu alte cuvinte x_n este restul împărţirii numărului $ax_{n-1} + b$ la modulul p.

Cum elementele şirlui (x_n) aparţin mulţimii $\{0, 1, 2, \dots, p-1\}$, şirul asociat (u_n), cu $u_n = \dfrac{x_n}{p}$, este un şir de numere din intervalul $[0, 1)$.

Are şirul (u_n), $n = \overline{1, N}$, atributele unui şir de valori de observaţie asupra unui şi de variabile i.i.d (U_n), $U_n \sim \text{Unif}[0, 1)$, $n = \overline{1, n}$? Pentru a răspunde acestei întrebări, caracterizăm şirul $(x_n) \subset \mathbb{Z}_p$.

• Şirul (x_n) definit în mod recursiv pornind de la valoarea iniţială x_0 este reproductibil (repetabil) şi reproductibilitatea nu este un atribut al aleatorului. Opţiunea pentru astfel de generatori se explică prin faptul că permit verificarea şi *debugging*-ul codului implicat în simulare, folosind de mai multe ori acelaşi şir de numere produse de un generator.

• Şirul (x_n), fiind un şir de elemente dintr-o mulţime finită, este periodic, adică există un T astfel încât $x_{k+T} = x_k$, $\forall\ k \in \mathbb{N}$.

De exemplu, generatorul de modul $p = 16$, $a = 3, b = 4$ şi $x_0 = 0$ conduce la şirul periodic $0, 4, 0, 4 \ldots$. Luând $x_0 = 1$ obţinem şirul:

$$7, 9, 15, 1, 7, 9, 15, 1, \ldots$$

ce are perioada 4 (secvenţa 7, 9, 15, 1 se repetă). Luând $p = 16$, $a = 5, b = 3$, valoarea iniţială nu are importanţă, deoarece generatorul produce şirul periodic cu secvenţa repetitivă:

$$0, 3, 2, 13, 4, 7, 6, 1, 8, 11, 10, 5, 12, 15, 14, 9$$

Evident că un şir periodic nu poate fi considerat aleator. Dacă însă perioada este suficient de mare în raport cu numărul de termeni ce se folosesc în simulare, atunci el este acceptabil dacă şirul (u_n), cu $u_n = x_n/p$ trece anumite teste.

Sunt consideraţi generatori buni, cei pentru care lungimea perioadei este aceeaşi pentru orice valoare iniţială, şi mai mult perioada este maximum posibil.

Studii experimentale îndelungate, pe diferite tipuri de calculatoare, recomandă următorii generatori liniar congruenţiali ai căror parametri (p, a, b, x_0) sunt, respectiv:

$$\begin{array}{l} (2147483647, 16807, 0, 1), \\ (2147483647, 950706376, 0, 1), \\ (2147483647, 630360016, 0, 1), \\ (2147483648, 452807053, 0, 1), \\ (2147483647, 1078318381, 0, 1), \end{array} \tag{2}$$

$p = 2147483647 = 2^{31} - 1$, este număr prim de tip Mersenne, adică un număr prim de forma $2^q - 1$. Astfel generatorii asociaţi au perioada maximă $p - 1 = 2147483646$. Prin urmare se poate genera un şir de peste 2 miliarde de numere distincte x din $\mathbb{Z}_p$.

După ce perioada maximă a fost asigurată, generatorul este supus unor teste de uniformitate.

`ISO C` pune la dispoziţie funcţia din `stdlib.h`:

```
int rand(void);
```

ce implementează generatorul liniar congruenţial de modul $p = 2^{31}$ Funcţia returnează numere de tip `int` (pe 32 biţi) din mulţimea $\{0, 1, 2, \ldots, \texttt{RAND\_MAX}\}$, unde constanta `RAND_MAX` este $2^{31} - 1$.

Setarea valorii iniţiale se realizează cu funcţia:

```
void srand(unsigned seed);
```

Astfel codul:

```
#include<stdlib.h>
#define N 1000

int main()
 {
  int i, r;
  double u[N];
  srand(231);

 for(i=0;i<N;i++)
  {
  r=rand();
  u[i]=(double)r/RAND_MAX;
  }
return 0;
 }
```

generează un şir de 1000 de numere pseudo–aleatoare "uniform distribuite" pe $[0, 1)$.

De ce s-a ales pentru `rand()` un modul, putere a lui 2, $p = 2^m$? Motivul este că restul împărţirii unui număr întreg pozitiv, x, la 2^m este numărul binar constând din ultimii `m` biţi ai lui x, care se poate calcula foarte rapid.

Într-adevăr, dacă x se exprimă în binar prin: $x = (b_{31} \dots b_m b_{m-1} \dots b_1 b_0)_2$, atunci

$$\begin{aligned} x &= b_{31}2^{31} + \cdots b_m 2^m + b_{m-1}2^{m-1} + \cdots b_1 2 + b_0 2^0 \\ &= 2^m(b_{31}2^{31-m} + \cdots + b_m 2^0) + b_{m-1}2^{m-1} + \cdots b_0 2^0 \end{aligned}$$

Prin urmare restul împărţirii lui x la 2^m este în binar numărul $b_{m-1}2^{m-1} + \cdots b_0 2^0 = (0 \dots 0 b_{m-1} \dots b_1 b_0)_2$.

Pentru calculul rapid a numărului $(0 \dots 0 b_{m-1} \dots b_1 b_0)_2$ se ţine seama că

• $p = 2^m = (00 \dots 0 \underbrace{1}_{c_m} 0 \dots 0)_1$

• $p - 1 = (00 \dots 00 \underbrace{1 \dots 1}_{c_{m-1} \dots c_1 c_0})$;

• Practic se defineşte `masca`=p-1 efectuând operaţii pe biţi, astfel:
`masca=(1<<m)-1;`

• Atunci restul împărţirii lui `int x` la $p = 2^m$ este `r=masca & x`;

Testul de k–uniformitate: Elementele unui şir de numere pseudo–aleatoare (u_n), $n = \overline{1, k}$, $u_n \in [0, 1)$, trebuie să aibă atributele unor valori de observaţie asupra unui şir de variabile aleatoare $U_1, U_2, \dots, U_k$, independente şi uniform distribuite pe intervalul $[0, 1)$.

Pentru a prezenta unul din cele mai folosite teste de uniformitate şi independenţă, reamintim că dacă U este o variabilă aleatoare uniform distribuită pe intervalul $[0, 1)$, atunci probabilitatea ca U să ia valori într-un subinterval $[a, b] \subset [0, 1)$ este egală cu lungimea subintervalului: $P(U \in [a, b]) = b - a$, iar dacă $U_1, \dots, U_k$, sunt variabile aleatoare independente şi identic distribuite după legea uniformă pe $[0,1)$, atunci probabilitatea ca vectorul aleator $(U_1, U_2, \ldots, U_k)$ să ia valori în k–paralelipipedul $[a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_k, b_k] \subset [0,1)^n$ este egală cu volumul paralelipiedului:

$$\begin{array}{l} P(a_1 \leq U_1 \leq b_1, \ldots, a_k \leq U_k \leq b_k) = \\ P(a_1 \leq U_1 \leq b_1) \cdots P(a_k \leq U_k \leq b_k) = \\ (b_1 - a_1) \cdots (b_k - a_k), \quad \forall\, a_i, b_i, 0 \leq a_i < b_i < 1, i = \overline{1,k} \end{array} \tag{3}$$

Având un şir de numere (u_n) din intervalul $[0,1)$, se pune problema să decidem dacă acest şir este "aproximativ" uniform distribuit. În acest scop se consideră vectorii constituiţi din k–numere consecutive ale şirului (u_n):

$$(u_0, u_1, \ldots, u_{k-1}), (u_1, u_2, \ldots, u_k), (u_2, u_3, \ldots, u_{k+1}), \ldots, \tag{4}$$

Pentru orice k–paralelipiped $D = [a_1, b_1] \times \cdots \times [a_k, b_k]$, $0 \leq a_i < b_i < 1$, $i = \overline{1,k}$, se defineşte funcţia caracteristică:

$$\mathbf{1}_D(u) = \begin{cases} 1 & \text{dacă } y \in D \\ 0 & \text{dacă } y \notin D \end{cases} \tag{5}$$

Suma $\sum_{i=0}^{n-1} \mathbf{1}_D(u_i, u_{i+1}, \ldots, u_{i+k-1})$ dă numărul de k–vectori, dintre cei n constituiţi ca în (4), ce aparţin paralelipipedului D.

Definiţia 0.2.1 *Şirul $(u_n) \subset [0,1)$ se numeşte şir k–uniform, $k \geq 2$, dacă pentru orice k–paralelipiped $D = [a_1, b_1] \times \cdots \times [a_k, b_k]$, $0 \leq a_i < b_i < 1$, $i = \overline{1,k}$,*

$$\lim_{n \to \infty} \frac{1}{n} \sum_{i=0}^{n-1} \mathbf{1}_D(u_i, u_{i+1}, \ldots, u_{i+k-1}) = (b_1 - a_1) \cdots (b_k - a_k) = vol(D). \tag{6}$$

Practic, definiţia spune că dacă estimăm probabilitatea evenimentului ca k–vectorii constituiţi din şirul generat să cadă într-un k–paralelipiped, ca limita frecvenţelor experimentale de producere a evenimentului, atunci şirul este k–uniform dacă din k–vectorii constituţi, proporţia celor ce cad într-un k–paralelipiped este aproximativ egală cu volumul paralelipipedului.

Testele de k–uniformitate asupra generatorilor liniar congruenţiali au adus numeroase surprize. Să analizăm, de exemplu, generatorul `randu` care a fost inclus în biblioteca ştiinţifică a mainframe-urilor IBM (IBM 360/370) un număr mare de ani şi folosit pentru simulări în numeroase proiecte ştiinţifice de anvergură. Parametrii generatorului `randu` sunt:

$$p = 2^{31}, a = 65539 = 2^{16} + 3, b = 0$$

După mulţi ani de folosire, Marsaglia, profesor la Universitatea din Florida, a observat următoarea deficienţă a generatorului `randu`:

$$\begin{aligned} x_{n+2} &= (2^{16}+3)x_{n+1} = (2^{16}+3)^2 x_n \\ &= (2^{32} + 6 \cdot 2^{16} + 9)x_n = \underbrace{2 \cdot 2^{31}}_{0 \; mod \; 2^{31}} x_n + (6 \cdot 2^{16} + 18 - 9)x_n \\ &= [6 \cdot (2^{16}+3) - 9]x_n = 6(2^{16}+3)x_n - 9x_n = 6x_{n+1} - 9x_n, \end{aligned}$$

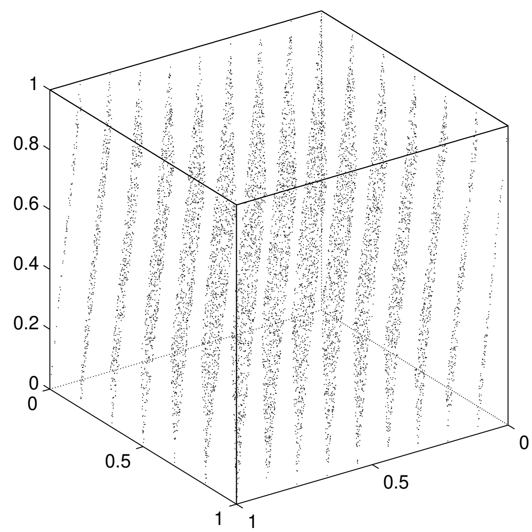

Fig.3: Ilustrarea dispunerii pe plane paralele a tripletelor de numere (u_n, u_{n+1}, u_{n+2}) asociate unui şir (u_n), generat de `randu`.

adică $9x_n - 6x_{n+1} + x_{n+2} = 0 \bmod 2^{31}$. Relaţia $9x_n - 6x_{n+1} + x_{n+2} = 0 \bmod 2^{31}$ este echivalentă cu $9x_n - 6x_{n+1} + x_{n+2} = k\,2^{31}, k \in \mathbb{Z}$. Prin împărţire la 2^{31} rezultă că pentru orice n, tripletele (u_n, u_{n+1}, u_{n+2}) aparţin unor plane de ecuaţie $9x - 6y + z = k$:

$$9u_n - 6u_{n+1} + u_{n+2} = k, \quad u_n = x_n/2^{31}.$$

Dintre toate planele paralele de ecuaţie $9x-6y+z = k$, $k \in \mathbb{Z}$, intersectează cubul unitate doar cele corespunzătoare lui $k \in \{-5, -4, \dots, 9\}$. Prin urmare tripletele de numere generate, (u_n, u_{n+1}, u_{n+2}), sunt dispuse în cubul $[0, 1)^3$ pe 15 plane paralele (Fig.3).

Această particularitate ilustrează că punctele (u_n, u_{n+1}, u_{n+2}) nu sunt uniform dispersate în cub, şi deci şirul (u_n) nu are atributele unui şir uniform distribuit.

După depistarea acestei deficienţe a generatorului `randu` s-a demonstrat că orice generator liniar congruenţial are acest defect de regularitate în anumite dimensiuni k, adică k–punctele:

$$(u_0, u_1 \dots, u_{k-1}), (u_1, u_2, \dots, u_k), \dots$$

constituite din elemente ale şirului (u_n) sunt dispuse pe un număr redus de hiperplane din hipercubul $[0, 1)^k$, în loc să fie dispersate în hipercub.

În Fig.4 sunt reprezentaţi vectorii (u_k, u_{k+1}), $k = \overline{1, 3000}$, constituiţi din perechi de numere aleatoare consecutive, produse de generatorul liniar congruenţial de parametri $a = 65, b = 1, p = 2048$ (stânga), respectiv $a = 3, b = 0, p = 2048$ (dreapta).

Toate limbajele comune de programare `C, C++, Java`, conţin funcţii ce implementează un generator liniar congruenţial. Ştiind că au acest defect, funcţiile respective nu sunt indicate în probleme serioase de simulare.

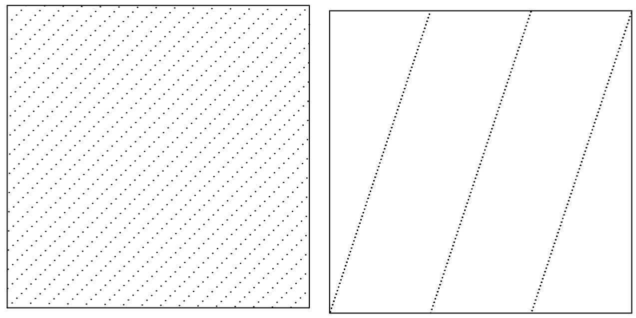

Fig.4: Structura regulată a numerelor aleatoare produse de generatori liniar congruenţiali.

Proprietăţile de regularitate ale şirului generat prin metoda liniar congruenţială au condus la dezvoltarea unor metode ne–congruenţiale de generare de numere pseudo–aleatoare uniform distribuite pe $[0, 1)$.

Cel mai performant generator existent la ora actuală este generatorul numit, Mersenne–Twister, devoltat de M. Matsumoto şi T. Nishimura

Perioada şirului generat de Mersenne–Twister este $2^{19937} - 1$. Şirurile generate au trecut teste de k–uniformitate pentru orice $k \leq 623$.

Acest generator este implementat în `Python`, `MATLAB`, `PHP`, `Ruby`, etc.