Cursul 8

Generatori de numere pseudo-aleatoare

8.1 Generatori de numere pseudo-aleatoare

A simula o variabilă aleatoare X revine la a rula un algoritm determinist care produce sau generează un şir de numere $x_1, x_2, \ldots, x_N$ 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 proceselor din sisteme complexe cu intrări aleatoare.

Un generator de numere pseudo-aleatore este invocat în diverse etape ale algoritmilor probabilişti. Pe de altă parte, numerele pseudo-aleatoare se generează în scopul asigurării securităţii proceselor într-un sistem de operare sau a transmiterii informaţiei pe un canal de comunicaţie.

Modalitatea de generare a numerelor pseudo-aleatoare depinde de scopul pentru care acestea sunt folosite. Noi abordăm problematica generării acestor numere doar 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. Algoritmii folosiţi în simularea variabilelor aleatoare se numesc *generatori de numere pseudo-aleatoare*.

8.1.1 Generatorul liniar congruenţial

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 împărţirii cu rest din aritmetică asigură că pentru $n \in \mathbb{Z}$ şi $p \in \mathbb{N}^*$ există $q \in \mathbb{Z}$ şi $r \in \mathbb{N}$ astfel încât $n = qp + r$, $r \in \{0, 1, 2, \ldots, p-1\}$. Prin urmare, inelul $\mathbb{Z}_p$ conţine clasele de resturi $\{0, 1, 2, \ldots, p-1\}$.

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

$$x_n = ax_{n-1} + b \pmod{p},\ n \geq 1, \tag{8.1}$$

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 p.

Cum elementele şirului (x_n) aparţin mulţimii $\{0, 1, 2, \ldots, p-1\}$, 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 şir de variabile independente identic distribuite (i.i.d.) $U_n \sim \text{Unif}[0, 1)$, $n = \overline{1, N}$?

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 $T \in \mathbb{N}^*$ 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) trece anumite teste.

Sunt consideraţi *generatori buni* cei pentru care lungimea perioadei este aceeaşi pentru orice valoare iniţială şi perioada este maximă.

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

$$\begin{array}{l}(2147483647, 16807, 0, 1)\\ (2147483647, 950706376, 0, 1)\\ (2147483647, 630360016, 0, 1)\\ (2147483648, 452807053, 0, 1)\\ (2147483647, 1078318381, 0, 1).\end{array} \tag{8.2}$$

$p = 2147483647 = 2^{31} - 1$ 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 liniar congruenţial 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 de 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 de forma $p = 2^m$? 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 $x = (b_{31} \ldots b_m b_{m-1} \ldots b_1 b_0)_2$, 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

$$b_{m-1}2^{m-1} + \cdots + b_0 2^0 = (0 \ldots 0 b_{m-1} \ldots b_1 b_0)_2.$$

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

• $p = 2^m = (00 \ldots 0 \underbrace{1}_{c_m} 0 \ldots 0)_2$;

• $p-1=(00\ldots00\underbrace{1\ldots1}_{c_{m-1}\ldots c_1c_0})_2$;

• Practic, se defineşte `masca`$=p-1$ efectuând operaţii pe biţi, astfel:

```
masca=(1<<m)-1;
```

• Restul împărţirii lui `int x` la $p=2^m$ este `r=masca & x`.

Testul de k–uniformitate:

Elementele unui şir de numere pseudo–aleatoare $u_n\in[0,1)$, $n=\overline{1,k}$, trebuie să aibă atributele unor valori de observaţie asupra unui şir de variabile aleatoare $U_1,U_2,\ldots,U_k$ 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. De asemenea, se poate demonstra că dacă $U_1,\ldots,U_k$ sunt variabile aleatoare i.i.d. după legea uniformă pe $[0,1)$, atunci probabilitatea ca vectorul $(U_1,U_2,\ldots,U_k)$ 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 paralelipipedului:

$$\begin{aligned}P(a_1\le U_1\le b_1,\ldots,a_k\le U_k\le b_k)&=P(a_1\le U_1\le b_1)\cdots P(a_k\le U_k\le b_k)\\&=(b_1-a_1)\cdots(b_k-a_k),\end{aligned}$$

pentru orice a_i,b_i cu $0\le a_i<b_i<1, i=\overline{1,k}$.

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):

$$(u_0,u_1,\ldots,u_{k-1}),\ (u_1,u_2,\ldots,u_k),\ (u_2,u_3,\ldots,u_{k+1}),\ldots \tag{8.3}$$

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

$$\mathbf{1}_D(u)=\begin{cases}1, & \text{dacă } u\in D,\\ 0, & \text{dacă } u\notin D.\end{cases} \tag{8.4}$$

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 (8.3) ce aparţin paralelipipedului D.

Definiţia 8.1.1 Şirul $(u_n)\subset[0,1)$ se numeşte *şir k–uniform*, $k\ge2$, dacă pentru orice k–paralelipiped $D=[a_1,b_1]\times\cdots\times[a_k,b_k]$, $0\le a_i<b_i<1$, $i=\overline{1,k}$,

$$\lim_{n\to\infty}\frac{1}{n}\sum_{i=0}^{n-1}\mathbf{1}_D(u_i,u_{i+1},\ldots,u_{i+k-1})=(b_1-a_1)\cdots(b_k-a_k)=\mathrm{vol}(D). \tag{8.5}$$

Practic, definiţia k-uniformităţii spune că dacă estimăm probabilitatea evenimentului ca cei 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`:

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

adică $9x_n - 6x_{n+1} + x_{n+2} = 0 \,(\mathrm{mod}\, 2^{31})$. Relaţia $9x_n - 6x_{n+1} + x_{n+2} = 0 \,(\mathrm{mod}\, 2^{31})$ 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 $k \in \{-5, -4, \ldots, 9\}$. 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. 8.1).

Această particularitate ilustrează faptul că punctele (u_n, u_{n+1}, u_{n+2}) nu sunt uniform dispersate în cub, 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

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

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

Toate limbajele comune de programare (de exemplu, `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.

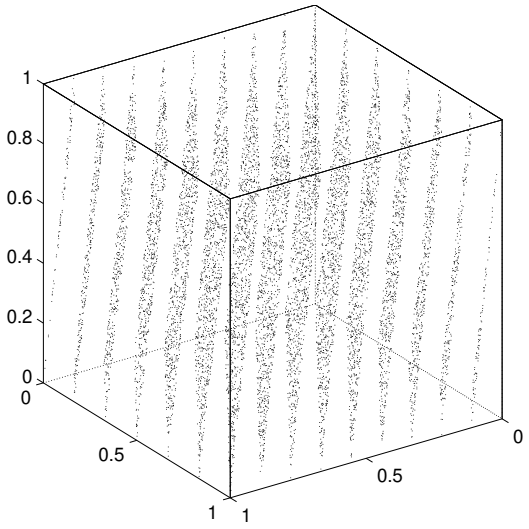

Fig.8.1: Ilustrarea dispunerii pe plane paralele a tripletelor de numere (u_n, u_{n+1}, u_{n+2}) asociate unui şir (u_n) generat de `randu`.

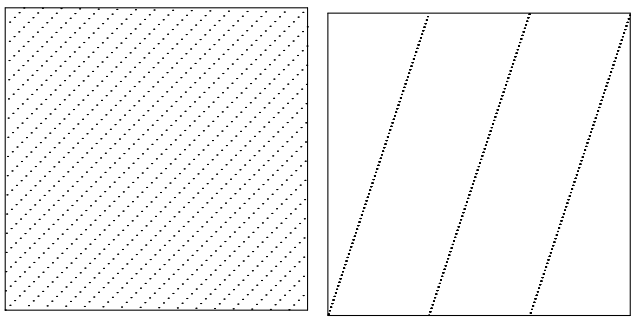

Fig.8.2: Structura regulată a numerelor aleatoare produse de generatori liniar congruenţiali.

8.1.2 Generatorul Mersenne-Twister

Proprietăţile de regularitate ale şirului (u_n) generat prin metoda liniar congruenţială au condus la necesitatea dezvoltării unor metode necongruenţiale de generare a unor numere pseudo-aleatoare uniform distribuite pe $[0, 1)$. Cel mai performant generator existent la ora actuală pentru calculatoare pe 32 de biţi este generatorul 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 $1 \leq k \leq 623$.

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

Detalii de folosire a generatorului Mersenne–Twister

Fişierul `mersenne.c` conţine codul pentru a putea genera numere pseudo-aleatoare din intervalul $[0, 1]$, $[0, 1)$, respectiv $(0, 1)$.

Vom folosi în mod deosebit funcţia:

```
double genrand_real2(void);
```

care returnează valori din $[0, 1)$.

Generatorul Mersenne–Twister are setat un seed implicit. După ce codul propriu care invocă generatorul merge OK, se setează un alt seed, care din motive de securitate se indică a fi ceasul calculatorului (adică timpul indicat de calculator în momentul execuţiei programului).

Pentru a realiza această setare, se include fişierul header `time.h`, în care este definit tipul `time_t` şi funcţia `time`.

Declarând:

```
time_t secunde;// si apoi apeland functia time:
secunde=time(NULL);
```

avem stocat în `secunde` numărul de secunde care s-au scurs de la 1 ianuarie 1970 până în momentul execuţiei liniei `secunde=time(NULL)`.

`secunde` este apoi folosit ca seed pentru generatorul Mersenne–Twister:
`init_genrand(secunde);`

Într-un articol publicat în anul 2018 s-a reuşit implementarea unui generator de numere pseudo-aleatoare, prin adaptarea generatorului Mersenne–Twister, pentru calculatoare pe 64 de biţi: *the 64-bit Maximally Equidistributed F2-Linear Generators with Mersenne Prime Period* (MELG-64) (vezi `https://github.com/sharase/melg-64`).