Generowanie rozkładu równomiernego

Dariusz Borkowski

L

iczbą prawdziwie losową nazywamy nieprzewidywalną wartość zmiennej losowej. Źródłem liczb prawdziwie losowych mogą być generatory fizyczne np. rzut monetą, szum w urządzeniach elektronicznych, promieniowanie kosmiczne. Niestety tego typu generatory są zbyt wolne lub mają problem ze stabilnością (zmiana warunków fizycznych może istotnie zmienić własności probabilistyczne).

Liczby pseudolosowe są to liczby otrzymywane według ścisłej formuły matematycznej. Są one odtwarzalne a przez to w ogóle nielosowe w sensie matematycznym. Mają jedynie "wygląd" losowości tzn. ich własności statystyczne są bardzo bliskie własnościom liczb prawdziwie losowych. Ktoś kto nie zna formuły, według której są generowane, nie powinien być w stanie stwierdzić, że nie są to liczby prawdziwie losowe. Źródłem liczb pseudolosowych są generatory matematyczne (programy). Dlatego też liczby pseudolosowe są nazywane po prostu liczbami losowymi, a matematyczne algorytmy do ich otrzymywania generatorami liczb losowych.

Generatory liniowe

Ogólna postać generatora liniowego jest następująca:

$$X_n = (a_1 X_{n-1} + a_2 X_{n-2} + \cdots + a_k X_{n-k} + c) \mod m,$$

gdzie $a_1, \ldots, a_k, c, m \in \mathbb{N}$ są parametrami generatora oraz działanie $a \mod b$ zwraca resztę z dzielenia a przez b.

W pierwszej kolejności musimy zainicjować generator poprzez dostarczenie danych początkowych $X_0, X_1, \ldots, X_k$. Maksymalny okres generatora jaki można otrzymać dla odpowiednio dobranych parametrów wynosi $m^k - 1$.

Popularną implementacją (np. w starych bibliotekach C/C++, Pascal) jest generator

$$X_n = (aX_{n-1} + c) \mod m. \qquad (1)$$

a	c	m	Nazwa/autor
$2^{16}+3$	0	2^{31}	RANDU (IBM)
$2^2 \cdot 23^7+1$	0	2^{35}	Zieliński (1966)
69069	1	2^{32}	Marsaglia (1972)
16807	0	$2^{31}-1$	Park, Miller (1980)
40692	0	$2^{31}-249$	L'Ecuyer (1988)
68909602460261	0	2^{48}	Fishman (1990)

Tabela 1: Przykładowe parametry generatora (1) o maksymalnym okresie

Lepsze własności statystyczne od (1) mają generatory w ogólniejszej postaci np.

$$X_n = (1176X_{n-1} + 1476X_{n-2} + 1776X_{n-3}) \mod m,\ m = 2^{32} - 5, \quad (2)$$

$$X_n = 2^{13}(X_{n-1} + X_{n-2} + X_{n-3}) \mod m,\ m = 2^{32} - 5, \quad (3)$$

$$X_n = (1995X_{n-1} + 1998X_{n-2} + 2001X_{n-3}) \mod m,\ m = 2^{35} - 849, \quad (4)$$

$$X_n = 219(X_{n-1} + X_{n-2} + X_{n-3}) \mod m,\ m = 2^{32} - 1629. \quad (5)$$

Podstawową wadą generatorów liniowych jest to, że wielowymiarowe rozkłady wyglądają bardzo nielosowo tzn. wykazują efekt Marasaglii. Polega on na tym, że wielowymiarowe rozkłady układają się na regularnych hiperpłaszczyznach dla których znane są metody znajdowania odległości.

Zadanie 1. *Zaimplementuj w C/C++ generator* (3).

Rozwiązanie:

```
#include <stdio.h>
#include <math.h>
static double a,b,c;
void init_ecng(int ia, int ib, int ic){
        a=ia; b=ib; c=ic;
}
double ecng(){
        static int n;
        static double d,x;
        d=(a+b+c)*8192;
        x=fmod(d,4294967291.0);
        a=b; b=c; c=x;
        /* zamieniamy liczbe 32-bitowa
                na typ double i przedzial
```

```
                        (0,1) */
        if(x<(float)2147483648.)n=(int)x;
        else n=(int) (x-4294967296.);
        return (n*2.3283064365e-10+0.5);
}
void main(void){
        int i;
        init_ecng(12,34,56);
        for(i=0;i<100;i++)
         printf("%f\n",ecng());
}
```

□

Generatory oparte na rejestrach przesuwnych

Tego typu generatory tworzą ciąg bitów według wzoru

$$b_n = (a_1 b_{n-1} + \cdots + a_k b_{n-k}) \mod 2,$$

gdzie $a_1, a_2, \ldots, a_k \in \{0, 1\}$. Następnie wybieramy fragmenty L-bitów i zamieniamy na liczby z przedziału $(0, 1)$

$$U_i = \sum_{j=1}^{L} 2^{-j} b_{is+j} = 0.b_{is+1} \ldots b_{is+L},\ s \leqslant L,$$

gdzie s jest ustaloną liczbą całkowitą. Jeżeli s nie ma wspólnych dzielników z $2^k - 1$, to ciąg $\{U_i\}$ ma maksymalny okres który wynosi $2^k - 1$.

Największą zaletą tych generatorów jest to, że są łatwe do implementacji ponieważ $(a+b) \mod 2 =$

a xor b. Niestety nie spełniają one niektórych najnowszych testów statystycznych.

W literaturze znane są dwa szczególne przypadki tych generatorów. Pierwszym z nich jest generator Tauswortha (1965) opisany wzorem

$$b_n = b_{n-p} \text{ xor } b_{n-q},\ p > q.$$

A sam algorytm Tauswortha jest następujący:

> Niech A będzie L - bitową liczbą całkowitą o bitach początkowych $b_1, \ldots, b_L$ (U_0). Niech B będzie L - bitową zmienną pomocniczą oraz $q < p/2$ i $0 < s < p - q$.

1. $B = ((A \ll q) \text{ xor } A) \ll (L - p)$
2. $A = (A \ll s) \text{ xor } (B \gg (L - s))$
3. Return A: goto 1,

gdzie $A \ll k$ oznacza przesunięcie bitów reprezentacji binarnej o k pozycji w lewo i uzupełnienie

zerami zwolnionych bitów.

Drugi powszechnie znany generator Tezuki (1995) jest kombinacją trzech generatorów Tauswortha i zwraca

$$A^{(1)} \text{ xor } A^{(2)} \text{ xor } A^{(3)},$$

gdzie $A^{(i)}$ jest wynikiem i - tego generatora Tauswortha.

Zadanie 2. *Zaimplementuj w C/C++ algorytm Tezuki dla* $L = 32$ *oraz*

Generator	p	q	s *(inicjalizacja)*	s *(generacja)*
1.	28	9	4	13
2.	29	2	3	20
3.	31	6	1	17

Rozwiązanie:

```
#include <stdio.h>
static unsigned int s1,s2,s3;
void init(unsigned int i, unsigned int j,
unsigned int k){
        unsigned int b;
        s1=i; s2=j, s3=k;
        b=((s1<<9)^s1)<<4;
        s1=(s1<<4)^(b>>28);
        b=((s2<<2)^s2)<<3;
```

```
        s2=(s2<<3)^(b>>29);
        b=((s3<<6)^s3)<<1;
        s3=(s3<<1)^(b>>31);
}
double combT(){
        unsigned int b;
        b=((s1<<9)^s1)<<4;
        s1=(s1<<13)^(b>>19);
        b=((s2<<2)^s2)<<3;
        s2=(s2<<20)^(b>>12);
        b=((s3<<6)^s3)<<1;
        s3=(s3<<17)^(b>>15);
        return (s1^s2^s3)*2.3283064365e-10;
}
void main(void){
        int i;
        init(12,34,56);
        for(i=0;i<100;i++)
```

```
        printf("%f\n",combT());
}
```

□

Generatory Fibonacciego

Pierwszy tego typu generator został zaproponowany przez Taussky i Todd (1956). Jest on implementacją wzoru rekurencyjnego Fibonacciego

$$X_n = X_{n-2} + X_{n-1} \mod m,\ n \geqslant 2.$$

Niestety nie spełnia on testów niezależności liczb losowych. Tej wady nie ma bardziej ogólna postać generatora, którą oznaczamy $F(r, s, \odot)$, a zadana jest wzorem

$$X_n = (X_{n-r} \odot X_{n-s}) \mod m,\ n \geqslant r \geqslant s \geqslant 1,$$

gdzie $\odot$ oznacza jedną z operacji $+$, $-$, $*$, xor. Jeżeli $m = 2^L$, to okres generatorów $F(r, s, +)$, $F(r, s, -)$ wynosi $(2r-1)^{L-1}$, generatora $F(r, s, *)$ wynosi $(2r-1)^{L-3}$, a generatora $F(r, s, \text{xor})$ wy-

nosi $(2r-1)$. Najlepsze własności statystyczne ma generator z operacją mnożenia.

Tabela 2: Przykładowe wartości r i s dające maksymalne okresy

r	17	31	55	68	97	607	1279
s	5	13	24	33	33	273	418

Zadanie 3. *Zaimplementuj w C/C++ generator* $F(97, 33, \circ)$, *gdzie*

$$x \circ y = \begin{cases} x - y & x \geqslant y, \\ x - y + 1 & x < y. \end{cases}$$

Zainicjuj początkową tablicę 97 *liczb za pomocą ciągu bitów*

$$(0.b_1 b_2 \ldots b_{24}, 0.b_{25} b_{26} \ldots b_{48}, \ldots)$$

gdzie b_n *liczymy następująco*

$$\begin{aligned} y_n &= (y_{n-3} \cdot y_{n-2} \cdot y_{n-1}) \mod 179 \\ z_n &= (52 z_{n-1} + 1) \mod 169 \\ b_n &= \begin{cases} 0 & (y_n \cdot z_n \mod 64) < 32 \\ 1 & \textit{w p.p.} \end{cases} \end{aligned}$$

Rozwiązanie:

```
#include <stdio.h>
#include <math.h>
static double uu[97];
static int ip=97;
static int jp=33;
void restart(int i, int j, int k, int l){
int ii,jj,m,wi,wj,wk,wl; double s,t;
wi=i;wj=j;wk=k;wl=l;
for(ii=0;ii<97;ii++){
        s=0;t=0.5;
        for(jj=1;jj<=24;jj++){
                m=(((wi*wj)%179)*wk)%179;
                wi=wj; wj=wk; wk=m;
                wl=(53*wl+1)%169;
                if((wl*m)%64>=32)s+=t;
                t*=0.5;
        }
        uu[ii]=s;
```

```
}
}

double uni(void){
        double pom;
        pom=uu[ip-1]-uu[jp-1];
        if(pom<0.0)pom+=1;
        uu[ip-1]=pom;
        ip--;
        if(ip==0)ip=97;
        jp--;
        if(jp==0)jp=97;
        return(pom);
}

void main(void){
        int i;
        restart(12,34,56,78);
```

```
        for(i=0;i<100;i++)
         printf("%f\n",uni());
}
```

□

Generatory uniwersalne

Przez generator uniwersalny rozumiemy generator dający identyczne wyniki na komputerach, w których liczby całkowite są reprezentowane przez co najmniej 16 bitów, a liczby w arytmetyce zmiennopozycyjnej mają przynajmniej 24 bitową mantysę.

Przykładem takiego generatora jest generator MZT (Marsaglia, Zaman, Tsanga, 1990) generujący liczby losowe o rozkładzie równomiernym na odcinku $[0, 1)$. Spełnia on wszystkie znane testy statystyczne i ma duży okres, równy 2^{144}. Generator ten jest kombinacją dwóch prostych generatorów

$$U_n = V_n \circ c_n,$$

gdzie V_n jest wynikiem generatora $F(97, 33, \circ)$, a

$$
\begin{aligned}
c_n =& c_{n-1} \star (7654321.0/16777216.0),\ n \geqslant 2,\\
c_1 =& 362436.0/16777216.0
\end{aligned}
$$

$$
c \star d = \begin{cases} c - d & c \geqslant d \\ c - d + 16777213.0/16777216.0 & c < d. \end{cases}
$$

Zadanie 4. *Zaimplementuj w C/C++ generator MZT.*

Rozwiązanie:

```
#include <stdio.h>
#include <math.h>
static double uu[97];
static int ip=97;
static int jp=33;
static double cc=362436.0/16777216.0;
static double cd=7654321.0/16777216.0;
static double cm=16777213.0/16777216.0;
void restart(int i, int j, int k, int l){
int ii,jj,m,wi,wj,wk,wl; double s,t;
wi=i;wj=j;wk=k;wl=l;
for(ii=0;ii<97;ii++){
        s=0;t=0.5;
        for(jj=1;jj<=24;jj++){
```

```
                    m=(((wi*wj)%179)*wk)%179;
                    wi=wj; wj=wk; wk=m;
                    wl=(53*wl+1)%169;
                    if((wl*m)%64>=32)s+=t;
                    t*=0.5;
            }
            uu[ii]=s;
}
}
double uni(void){
            double pom;
            pom=uu[ip-1]-uu[jp-1];
            if(pom<0.0)pom+=1;
            uu[ip-1]=pom;
            ip--;
            if(ip==0)ip=97;
            jp--;
            if(jp==0)jp=97;
```

```
        cc-=cd;
        if(cc<0.0)cc+=cm;
        pom-=cc;
        if(pom<0.0)pom+=1;
        return(pom);
}
void main(void){
        int i;
        restart(12,34,56,78);
        for(i=1;i<100;i++)
         printf("%f\n",uni());
}
```

□

Uwaga 1. *Nie ma żadnego wyraźnego powodu by formuły rekurencyjne miały dawać liczby losowe lub nawet wyglądające na losowe. To, że tak jest wydaje się raczej zaskakujące! Całkowicie różny od przedstawionych generatorów jest generator o nazwie RANLUX. Niestety nie ma zrozumienia dla teorii na podstawie, której powstał. Przeszedł on pomyślnie wszystkie najsurowsze testy statystyczne. Jednak jest on pomijany milczeniem przez niemal wszystkich matematyków i ekspertów od generatorów liczb losowych.*