Przegląd generatorów liczb pseudolosowych i analiza ich wybranych własności

Jan Karwowski

Zakład Sztucznej Inteligencji i Metod Obliczeniowych
Wydział Matematyki i Nauk Informacyjnych PW

19 V 2016

Liczby (pseudo)losowe

- Generowane przez komputer
- Zastępują metody uzyskiwania realizacji zmiennej losowej z zewnątrz w środowisku deterministycznym
- Program ma stan wewnętrzny
- Program zwraca liczby zmieniając ten stan

Zastosowania ciągów losowych

- Metody symulacyjne
- Generowanie przykładów testowych
- Imitowanie niedoskonałości

Pożądane własności ciągów losowych

- Dobra imitacja rozkładu jednostajnego.
- Kolejne elementy sprawiają wrażenie niezależnych.
- Długi okres ciągu.
- Nie potrzeba dużej nieprzewidywalności

Metody teoretyczne

Okres ciągu

Krótki – mała niezależność „zmiennych losowych”

k-equidistribution ciągu X

Dla dowolnego niezerowego ciągu $\{a_1, \ldots, a_k\}$ prawdopodobieństwo jego pojawienia się w ciągu $X = x_1, x_2, \ldots$ jest równe prawdopodobieństwu pojawienia się ciągu $\{x_{m+1}, x_{m+2}, x_{m+k}\}$ dla dowolnego m.

Testy empiryczne I

Testy empiryczne

Wygenerowanie N elementów i przeprowadzenie na nich testu statystycznego.

- Test równomierności
- Częstość par – liczby całkowite z niedużego przedziału $[0, k)$, zliczanie jak często występuje dana para
- Test odstępów – zliczamy wartości zmiennej r takiej, że $X_i, X_{i+1}, \ldots X_{i+r-1} \notin [\alpha, \beta]$, a $X_{i+r} \in [\alpha, \beta]$.
- Test kolekcjonera
- Test permutacji

Statystyka testowa: χ^2.

Donald Knuth. *The Art of Computer Programming, Volume Two, Seminumerical Algorithms*. 1998

Testy empiryczne II

Test spektralny

W przestrzeni t-wymiarowej $[0,1]^t$, rozważamy zbiór punktów postaci: $Y_i = (X_i, X_{i+1}, \ldots, X_{i+t-1})$, dla pewnych kolejnych i.

- Test przeznaczony dla generatorów LCG.
- Rozważmy wszystkie rodziny $t-1$-wymiarowych równoległych hiperpłaszczyzn, które pokrywają wszystkie Y_i.
- Jaka jest maksymalna odległość między dwiema sąsiednimi płaszczyznami w takiej rodzinie?
- Donald Knuth. *The Art of Computer Programming, Volume Two, Seminumerical Algorithms*. 1998
- RANDU: $X_i + 1 = 65539 X_i \mod 2^{31}$

Zły generator (Knuth)

Donald Knuth. *The Art of Computer Programming, Volume Two, Seminumerical Algorithms*. 1998

`http://www.informit.com/articles/article.aspx?p=2221790`

Linear Congruential Generator I

Ciąg, parametry: a, c, m

$$X_{i+1} = aX_i + c \mod m$$

Informacje

- Najpopularniejszy w bibliotekach standardowych
- Dobrze zbadany
- Jakość bardzo zależy od parametrów
- Może mieć okres $m \Rightarrow$ nie ma złych seedów
- Stan: jedna liczba

Linear Congruential Generator II

Twierdzenie

Ciąg LCG ma okres m wtw. gdy:

- c jest względnie pierwsze z m,
- $b = a - 1$ jest wielokrotnością p, dla każdego p będącego liczbą pierwszą dzielącą m,
- b jest wielokrotnością 4, jeśli m jest wielokrotnością 4.

Dowód:
Thomas E Hull and Alan R Dobell. "Random number generators". In: *SIAM review* 4.3 (1962), pp. 230–254, tw. 1

Linear Congruential Generator III

Mniej znaczące bity z generatora są słabe

Jeśli $m = 2^e$, ciąg najmniej znaczących bitów można wyliczyć z

$$X_{i+1} = aX_i + c \mod 2$$

Trzy możliwe ciągi do uzyskania. . . jeden gdy a, c spełniają założenia twierdzenia.
Później o sztuczkach.

Uproszczona wersja, $c = 0$

$$X_{i+1} = aX_i \mod m$$

- RANDU
- Współcześnie nieużywany

Multiple Recursive Generator I

Ciąg, parametry a_i, k, m

$$X_n = \sum_{i=1}^{k} a_i X_{n-i} \mod m$$

Informacje

- Maksymalny możliwy okres:
- Dobrej jakości mało znaczące bity (dalej trochę gorsze niż bardziej znaczące)
- Trudny dobór parametrów
- seed: niezerowy wektor liczb

Multiple Recursive Generator II

Twierdzenie

Dla liczby pierwszej p ciąg

$$X_n = \sum_{i=1}^{k} a_i X_{n-i} \mod p$$

ma okres $p^k - 1 \Leftrightarrow$ wielomian $x^k - a_1 x^{k-1} - \cdots - a_k$ jest wielomianem pierwotnym modulo p.
Można pokazać, że dla $m = p^r$ okres wynosi $p^{r-1}(p^k - 1)$.
Dowód: Neal Zierler. "Linear recurring sequences". In: *Journal of the Society for Industrial and Applied Mathematics* 7.1 (1959), pp. 31–48

Multiple Recursive Generator III

Lagged-Fibonacci generator

$$X_n = X_{n-k} \pm X_{n-l} \mod m$$

Szczególny przypadek.
Czasami w wersji floating-point.

Dobór parametrów LF

Neal Zierler and John Brillhart. "On primitive trinomials (mod 2), II". In: *Information and Control* 14 (1969), pp. 566–569

Yoshiharu Kurita and Makoto Matsumoto. "Primitive t-nomials (t= 3, 5) over GF(2) whose degree is a Mersenne exponent $\leqslant$ 44497". In: *Mathematics of Computation* 56 (1991), pp. 817–821

Generalized Feedback Shift Register I

LFSR (Linear Feedback Shift Register

Operuje na k-bitowym ciągu.

1. Wylicz nową wartość skrajnego bitu funkcją liniową poprzednich bitów (można ją zrealizować używając XOR).
2. Przesuń ciąg o jeden bit i wstaw na puste miejsce wyliczoną wartość.

Rozwiązania sprzętowe.

GFSR

$$X_{i+n} = X_{i+m} \oplus X_i$$

Wrażliwy na ciąg inicjujący.
Theodore G Lewis and William H Payne. "Generalized feedback shift register pseudorandom number algorithm". In: *Journal of the ACM (JACM)* 20.3 (1973), pp. 456–468

Generalized Feedback Shift Register II

TGFSR

Makoto Matsumoto and Yoshiharu Kurita. "Twisted GFSR generators". In: *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 2.3 (1992), pp. 179–194 Modyfikacja GFSR.

$$X_{i+n} = X_{i+m} \oplus x_i A$$

Lepsze własności niż GFSR, jeśli dobrze dobrana macierz A.

Mersenne Twister (MT19937) I

Wariant TGFSR.
Makoto Matsumoto and Takuji Nishimura. "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator". In: *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 8.1 (1998), pp. 3–30

Ciąg

X_i – w-bitowa liczba

$$X_{i+n} = X_{i+m} \oplus (x_k^u | x_{k+x}^l) A$$

$$Z_i = X_i T$$

, gdzie: x^u – bardziej znaczące $w - r$ bitów, x^l – mniej znaczące r bitów

Mersenne Twister (MT19937) II

parametry

$$A = \begin{pmatrix} & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \\ a_{w-1} & a_{w-2} & \cdots & \cdots & a_0 \end{pmatrix}$$

$w = 32, n = 624$, $m = 397$, $r = 31$, $a_{w-1} \cdots a_0$ =0x9908B0DF
oraz odpowiednia procedura inicjująca wektor x.

Informacje

- Okres $2^{19937} - 1 \approx 10^{6000}$. Liczba pierwsza
- Równomierny rozkład kolejnych losowych wartości 32 bitowych $[0, 2^{32} - 1]^{623}$

Inne rozkłady I

Rozkład jednostajny na $[0, 1)$

$$X \in U(0, m)$$

$$Y = \frac{X}{m+1}$$

Użycie odwrotności dystrybuanty

$$Y_i = F^{-1}(X_i)$$

Inne rozkłady II

Szybszy sposób na rozkład normalny

Mervin E. Muller G. E. P. Box. "A Note on the Generation of Random Normal Deviates". In: *The Annals of Mathematical Statistics* 29.2 (1958), pp. 610–611. ISSN: 00034851. URL: http://www.jstor.org/stable/2237361

- Losowanie punktu (x_1, x_2) z koła o środku w $(0, 0)$ i promieniu 1.
- $S = x_1^2 + x_2^2$
- $g_i = x_i \sqrt{\frac{-2 \ln r}{S}}$

$g_1 = \cos(\theta)\sqrt{-2 \ln S}$

θ – rozkład jednostajny, S – rozkład jednostajny (wsp. biegunowe).

Baterie testów

- Operują na podanej procedurze generowania lub wygenerowanych elementach w pliku
- Zawierają zestaw testów empirycznych
- Liczą statystykę testową i p-value
- Podają dla których testów p-value było złe i jak bardzo

DIEHARD

George Marsaglia. *DIEHARD: a battery of tests of randomness*. 1995.
URL: `http://stat.fsu.edu/pub/diehard/`

- Pierwsza uznana bateria testów
- Niekonfigurowalna
- Poważne ograniczenia

TestU01

Pierre L'Ecuyer and Richard Simard. "TestU01: A C Library for Empirical Testing of Random Number Generators". In: *ACM Trans. Math. Softw.* 33.4 (Aug. 2007). ISSN: 0098-3500. DOI: 10.1145/1268776.1268777. URL: http://doi.acm.org/10.1145/1268776.1268777

- Rozwijana od 1986
- „Oficjalna" wersja w 2007
- Początkowo implementacja testów z *The Art of Computer Programming*
- Biblioteka pozwala uruchomić wybrane testy z wybranymi parametrami lub predefiniowany zestaw.
- Można uruchomić ciągu na danych odczytanych z pliku lub podać implementację generatora

Inne

- US National Institute of Standards and Technology – `http://csrc.nist.gov/rng`
- SPRNG – generatory i testy. Michael Mascagni and Ashok Srinivasan. "Algorithm 806: SPRNG: A scalable library for pseudorandom number generation". In: *ACM Transactions on Mathematical Software (TOMS)* 26.3 (2000), pp. 436–461
- ...

Generatory w popularnych rozwiązaniach I

Java

$$Y_i = X_i/2^{16} \text{ dzielenie całkowite}$$

$$X_i + 1 = 25214903917X_i + 11 \mod 2^{48}$$

$$X_0 = seed \oplus C$$

16 najmniej znaczących bitów jest zawsze odrzucane!
Uwaga: specjalna metoda losowania double!!

Generatory w popularnych rozwiązaniach II

.NET

$$X_i = X_{i-55} - X_{i-24} \mod 2^{32}$$

Procedura inicjująca stan z pojedynczego inta.
Wg. Knutha wartości 55 i 24 zostały zaproponowane przez G. J. Mitchella i D. P. Moore'a w 1958
Kod inicjujący za: Brian P Flannery et al. "Numerical recipes in C". In: *Press Syndicate of the University of Cambridge, New York* 24 (1992)

Generatory w popularnych rozwiązaniach III

R, MATLAB, wiele innych

Mersenne Twister 19937

$$X_{i+n} = X_{i+m} \oplus (x_k^u | x_{k+x}^l) A$$

$$Z_i = X_i T$$

(parametry wcześniej)

RANDU (dla porządku)

$X_i + 1 = 65539 X_i \mod 2^{31}$ Okres 2^{29} dla odpowiednich seedów. (dobry okres, ale słaby generator)
Używany w latach '60 przez IBM (potem innych).

Pomocne funkcje w bibliotekach I

- Zwracanie k losowych bitów/bajtów

```
private static final long addend = 0xBL;
private static final long mask = (1L << 48) - 1;
private static final long multiplier = 0x5DEECE66DL;


protected int next(int bits) {
    long oldseed, nextseed;
    AtomicLong seed = this.seed;
     do {
         oldseed = seed.get();
         nextseed = (oldseed * multiplier + addend) & mask;
     } while (!seed.compareAndSet(oldseed, nextseed));
     return (int)(nextseed >>> (48 - bits));
}

public int nextInt() {
    return next(32);
}


public void nextBytes(byte[] bytes) {
    for (int i = 0, len = bytes.length; i < len; )
        for (int rnd = nextInt(),
                 n = Math.min(len - i, Integer.SIZE/Byte.SIZE);
             n--> 0; rnd >>= Byte.SIZE)
            bytes[i++] = (byte)rnd;
}
```

Pomocne funkcje w bibliotekach II

- Zwracanie liczby całkowitej z zadanego przedziału

```
public boolean nextBoolean() {
    return next(1) != 0;
}

public int nextInt(int bound) {
    if (bound <= 0)
        throw new IllegalArgumentException(BadBound);

    int r = next(31);
    int m = bound - 1;
    if ((bound & m) == 0)  // i.e., bound is a power of 2
        r = (int)((bound * (long)r) >> 31);
    else {
        for (int u = r;
             u - (r = u % bound) + m < 0;
             u = next(31))
            ;
    }
    return r;
}
```

Pomocne funkcje w bibliotekach III

- Zwracanie liczby zmiennoprzecinkowej z przedziału $[0, 1)$

```
public float nextFloat() {
    return next(24) / ((float)(1 << 24));
}

private static final double DOUBLE_UNIT = 0x1.0p-53; // 1.0 / (1L << 53)
public double nextDouble() {
    return (((long)(next(26)) << 27) + next(27)) * DOUBLE_UNIT;
}
```

Pomocne funkcje w bibliotekach IV

Inicjowanie bez podawania ziarna

Bardziej zaawansowane niż time(NULL).
Zapobieganie inicjacji dwoma identycznymi ziarnami w krótkim okresie.
Przykład z Javy 8:

```
public Random() {
    this(seedUniquifier() ^ System.nanoTime());
}

private static long seedUniquifier() {
    // L'Ecuyer, "Tables of Linear Congruential Generators of
    // Different Sizes and Good Lattice Structure", 1999
    for (;;) {
        long current = seedUniquifier.get();
        long next = current * 181783497276652981L;
        if (seedUniquifier.compareAndSet(current, next))
            return next;
    }
}

private static final AtomicLong seedUniquifier
    = new AtomicLong(8682522807148012L);
```

Test

- Test z użyciem BigCrush z TestU01
- Generatory według materiałów udostępnionych przez autorów platform
- Seed 20150517, poza RANDU – 32

```
#include "ulcg.h"
#include "unif01.h"
#include "usoft.h"
#include "bbattery.h"

int main (void)
{
    unif01_Gen *gen;
    gen = usoft_CreateJava48(20150517,1);
    bbattery_BigCrush (gen);
    ulcg_DeleteGen (gen);
    return 0;
}
```

RANDU

```
========= Summary results of BigCrush =========

 Version:          TestU01 1.2.3
 Generator:        ulcg_CreateLCG
 Number of statistics:  155
 Total CPU time:   03:08:17.00
 The following tests gave p-values outside [0.001, 0.9990]:
 (eps  means a value < 1.0e-300):
 (eps1 means a value < 1.0e-15):

**Tu nastepuja 144 testy**

       Test                          p-value
 ----------------------------------------------
  1  SerialOver, r = 0                eps
  2  SerialOver, r = 22               eps
  3  CollisionOver, t = 2             eps
  4  CollisionOver, t = 2             eps
  5  CollisionOver, t = 3             eps
  6  CollisionOver, t = 3             eps
  7  CollisionOver, t = 7             eps
  8  CollisionOver, t = 7             eps
  9  CollisionOver, t = 14            eps
 10  CollisionOver, t = 14            eps
 11  CollisionOver, t = 21            eps
 12  CollisionOver, t = 21            eps
 13  BirthdaySpacings, t = 2          eps
 14  BirthdaySpacings, t = 3          eps
 15  BirthdaySpacings, t = 4          eps
 16  BirthdaySpacings, t = 7          eps
 17  BirthdaySpacings, t = 7          eps
 18  BirthdaySpacings, t = 8          eps
 19  BirthdaySpacings, t = 8          eps
```

Java

```
========= Summary results of BigCrush =========
Number of statistics:  160
The following tests gave p-values outside [0.001, 0.9990]:
(eps  means a value < 1.0e-300):
(eps1 means a value < 1.0e-15):
      Test                          p-value
-----------------------------------------------
 13  BirthdaySpacings, t = 2           eps
 14  BirthdaySpacings, t = 3           eps
 15  BirthdaySpacings, t = 4           eps
 16  BirthdaySpacings, t = 7           eps
 17  BirthdaySpacings, t = 7           eps
 18  BirthdaySpacings, t = 8           eps
 19  BirthdaySpacings, t = 8           eps
 20  BirthdaySpacings, t = 16          eps
 21  BirthdaySpacings, t = 16          eps
 22  ClosePairs mNP, t = 3          3.9e-35
 22  ClosePairs mNP1, t = 3         9.9e-47
 22  ClosePairs mNP2S, t = 3           eps
 23  ClosePairs mNP, t = 5          8.7e-5
 23  ClosePairs mNP1, t = 5         1.8e-4
 23  ClosePairs mNP2, t = 5         3.2e-4
 23  ClosePairs mNP2S, t = 5           eps
 24  ClosePairs mNP1, t = 9         4.9e-4
 24  ClosePairs mNP2S, t = 9       1.2e-151
 25  ClosePairs mNP, t = 16         3.2e-34
 25  ClosePairs mNP1, t = 16        6.4e-68
 25  ClosePairs mNP2, t = 16        5.7e-5
 25  ClosePairs mNP2S, t = 16          eps
 32  CouponCollector, r = 20           eps
 39  Run, r = 15                    1 -  9.5e-8
 60  WeightDistrib, r = 20             eps
 89  PeriodsInStrings, r = 20       0.9997
```

.NET

```
========= Summary results of BigCrush =========

Version:          TestU01 1.2.3
Generator:        umrg_CreateLagFib
Number of statistics:  160
Total CPU time:   02:53:09.53
The following tests gave p-values outside [0.001, 0.9990]:
(eps  means a value < 1.0e-300):
(eps1 means a value < 1.0e-15):

      Test                          p-value
-----------------------------------------------
30  CouponCollector, r = 0            eps
31  CouponCollector, r = 10           eps
32  CouponCollector, r = 20           eps
33  CouponCollector, r = 27           eps
34  Gap, r = 0                        eps
35  Gap, r = 25                       eps
36  Gap, r = 0                        eps
37  Gap, r = 20                       eps
59  WeightDistrib, r = 0              eps
60  WeightDistrib, r = 20             eps
61  WeightDistrib, r = 28             eps
62  WeightDistrib, r = 0              eps
63  WeightDistrib, r = 10             eps
64  WeightDistrib, r = 26             eps
77  RandomWalk1 H (L=1000, r=20)    7.9e-7
97  HammingIndep, L=300, r=0          eps
98  HammingIndep, L=300, r=26         eps
100  HammingIndep, L=1200, r=25       4.3e-4
-----------------------------------------------
All other tests were passed
```

MT19937

```
========= Summary results of BigCrush =========

Version:          TestU01 1.2.3
Generator:        ugfsr_CreateMT19937_98
Number of statistics:  160
Total CPU time:   03:09:41.53
The following tests gave p-values outside [0.001, 0.9990]:
(eps  means a value < 1.0e-300):
(eps1 means a value < 1.0e-15):

      Test                          p-value
----------------------------------------------
23  ClosePairs mNP1, t = 5           8.7e-4
23  ClosePairs NJumps, t = 5         1.1e-5
80  LinearComp, r = 0              1 - eps1
81  LinearComp, r = 29             1 - eps1
----------------------------------------------
All other tests were passed
```

Wnioski

- Standardowe generatory w znanych językach programowania są wystarczająco dobre do (niekryptograficznych) zastosowań.
- Zaprojektowanie dobrego generatora jest trudne.
- Warto jest sprawdzić kilka generatorów w konkretnym zastosowaniu.
- MT19937

Literatura I

- Brian P Flannery et al. “Numerical recipes in C”. In: *Press Syndicate of the University of Cambridge, New York* 24 (1992).
- Mervin E. Muller G. E. P. Box. “A Note on the Generation of Random Normal Deviates”. In: *The Annals of Mathematical Statistics* 29.2 (1958), pp. 610–611. ISSN: 00034851. URL: http://www.jstor.org/stable/2237361.
- Thomas E Hull and Alan R Dobell. “Random number generators”. In: *SIAM review* 4.3 (1962), pp. 230–254.
- Donald Knuth. *The Art of Computer Programming, Volume Two, Seminumerical Algorithms*. 1998.
- Yoshiharu Kurita and Makoto Matsumoto. “Primitive t-nomials (t= 3, 5) over GF(2) whose degree is a Mersenne exponent $\leqslant$ 44497”. In: *Mathematics of Computation* 56 (1991), pp. 817–821.

Literatura II

Pierre L'Ecuyer and Richard Simard. "TestU01: A C Library for Empirical Testing of Random Number Generators". In: *ACM Trans. Math. Softw.* 33.4 (Aug. 2007). ISSN: 0098-3500. DOI: 10.1145/1268776.1268777. URL: http://doi.acm.org/10.1145/1268776.1268777.

Theodore G Lewis and William H Payne. "Generalized feedback shift register pseudorandom number algorithm". In: *Journal of the ACM (JACM)* 20.3 (1973), pp. 456–468.

George Marsaglia. *DIEHARD: a battery of tests of randomness*. 1995. URL: http://stat.fsu.edu/pub/diehard/.

Michael Mascagni and Ashok Srinivasan. "Algorithm 806: SPRNG: A scalable library for pseudorandom number generation". In: *ACM Transactions on Mathematical Software (TOMS)* 26.3 (2000), pp. 436–461.

Literatura III

- Makoto Matsumoto and Yoshiharu Kurita. “Twisted GFSR generators”. In: *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 2.3 (1992), pp. 179–194.
- Makoto Matsumoto and Takuji Nishimura. “Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator”. In: *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 8.1 (1998), pp. 3–30.
- Neal Zierler. “Linear recurring sequences”. In: *Journal of the Society for Industrial and Applied Mathematics* 7.1 (1959), pp. 31–48.
- Neal Zierler and John Brillhart. “On primitive trinomials (mod 2), II”. In: *Information and Control* 14 (1969), pp. 566–569.