Roksana Krysztofiak

Wprowadzenie do symulacji i metod Monte Carlo - projekt 1

Cel projektu

W naszym projekcie chcemy sprawdzić jakość różnych generatorów liczb pseudolosowych oraz porównać je za pomocą różnych testów statystycznych.

Generatory

Użyjemy 4 różnych generatorów w celu otrzymania liczb pseudolosowych. W tym projekcie wybrałam RC4(32), LCG, GLCG, BBSG oraz MT. Ostatnie dwa nie były opisywane w treści naszego zadania, zatem zrobimy to teraz.

BBSG

Blum-Blum-Shub (BBSG) to generator liczb pseudolosowych, którego działanie opiera się na trudności rozkładu dużych liczb na czynniki pierwsze. Algorytm dla tego generatora, gdy generujemy pseudolosową sekwencję bitów $z_1, z_2, ..., z_l$ długośli l:

— Wybieramy dwie duże liczby pierwsze p i q, które dają resztę 3 w działaniu modulo 4. Następnie obliczamy $n = p \cdot q$

— Wybieramy ziarno początkowe x_0, które jest liczbą całkowitą z przedziału [1, n-1] oraz względnie pierwsze z n

— Dla i od 1 do l:

a) $x_i < -x_{i-1}^2 mod n$

b) $z_i < -$ najmniej znaczący bit z x_i

— Wynikiem jest sekwencja $z_1, z_2, ..., z_l$.

MT

Mersenne Twister, to bardzo popularny generator liczb pseudolosowych, wykorzystywany m.in. w Pythonie w funkcji random.random z modułu random. Ma on bardzo długi okres wynoszący $2^{19937} - 1$. Poniżej znajduje się implementacja algorytmu Mersenne Twister w pseudokodzie:

```
// Utworz tablice 624 elementow do przechowywania
//stanu generatora
int[0..623] MT
int index = 0

// Inicjuj generator przy uzyciu ziarna
function initializeGenerator(int seed) {
    MT[0] := seed
    for i from 1 to 623 {
        MT[i] := last 32 bits of(1812433253 * (MT[i-1]
          xor (right shift by 30 bits(MT[i-1]))) + i)
    }
}

// Utworz pseudolosowa liczbe na podstawie indeksu,
// wywolaj generateNumbers() aby utworzyc nowa tablice
//pomocnicza co 624 liczby
function extractNumber() {
    if index == 0 {
        generateNumbers()
    }

    int y := MT[index]
    y := y xor (right shift by 11 bits(y))
    y := y xor (left shift by 7 bits(y) and (2636928640))
    y := y xor (left shift by 15 bits(y) and(4022730752))
    y := y xor (right shift by 18 bits(y))

    index := (index + 1) mod 624
    return y
}

// Generuj tablice 624 liczb
```

```
function generateNumbers() {
    for i from 0 to 623 {
        int y := 32nd bit of(MT[i]) + last 31
          bits of(MT[(i+1) mod 624])
        MT[i] := MT[(i + 397) mod 624] xor
          (right shift by 1 bit(y))
        if (y mod 2) == 1 {
            MT[i] := MT[i] xor (2567483615)
        }
    }
}
```

Testy

W celu sprawdzenia jakości generatorów przyjrzymy się p_wartością różnych testów. Tutaj wykorzystałam testy Kołomogorowa-Smirnowa(KS), chi-kwadrat(CHI2) oraz Cramera-von Misesa(CVM).

CVM

Test Cramera–von Misesa(CVM) jest statystycznym testem zgodności używanym do sprawdzenia, czy dane pochodzą z określonego rozkładu. Jest oparty na kwadratowej odległości między empiryczną dystrybuantą $F_n(x)$ a teoretyczną dystrybuantą $F(x)$.

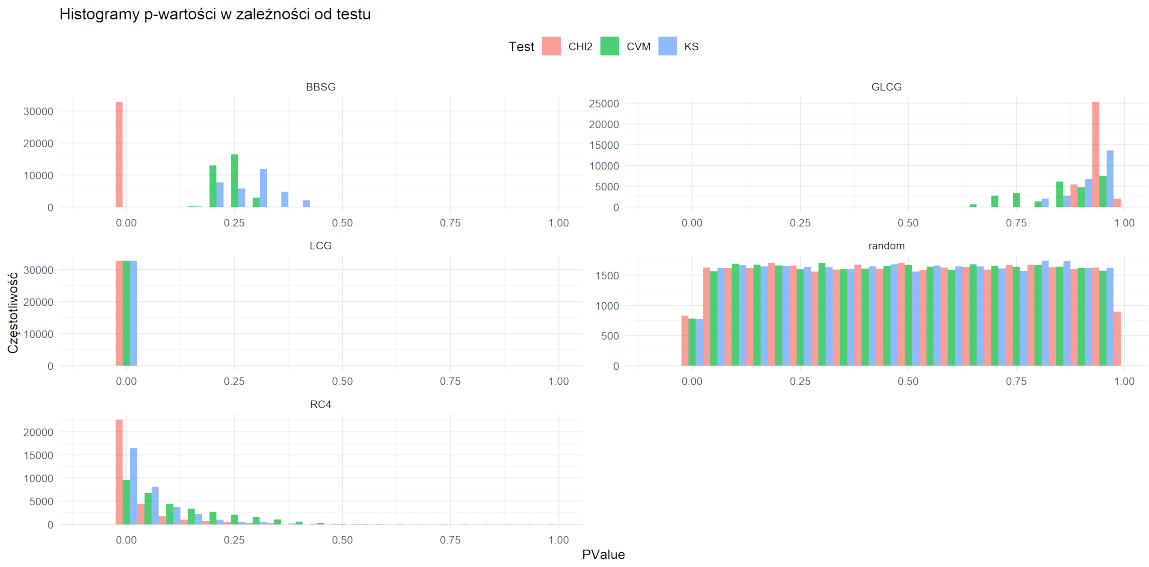
Statystyka testu jest zdefiniowana jako:

$$W^2 = n \int_{-\infty}^{\infty} (F_n(x) - F(x))^2 \, dF(x),$$

gdzie n to liczba obserwacji, $F_n(x)$ jest empiryczną dystrybuantą danych, a $F(x)$ to teoretyczna dystrybuanta.

Wyniki i wnioski

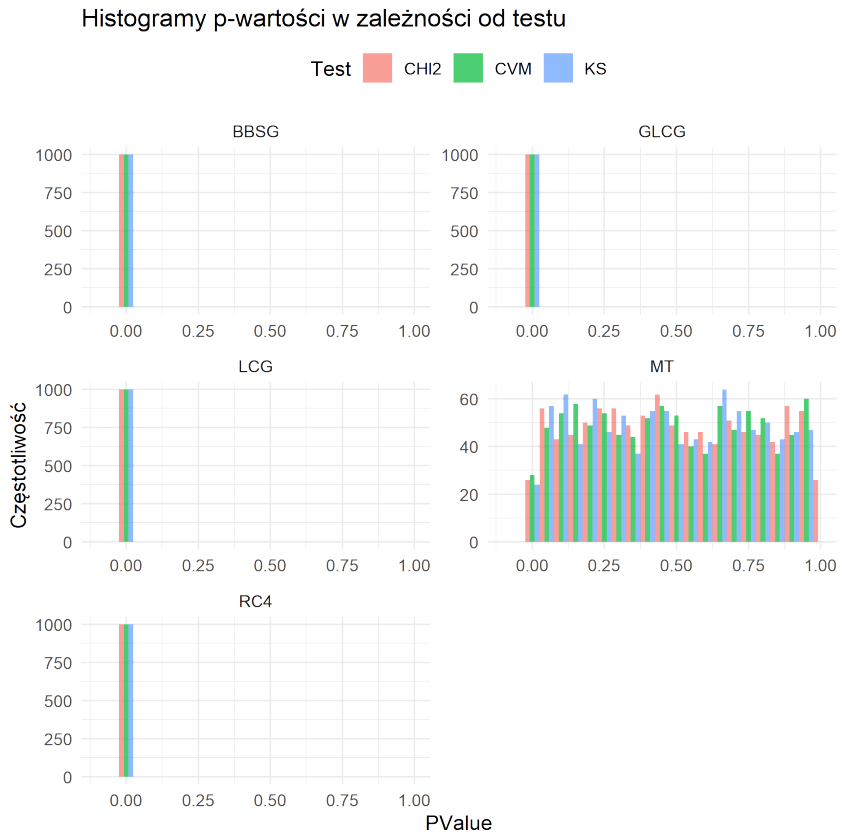
Na początku poprzez każdy generator stworzyliśmy 1000 liczb. Następnie poddaliśmy je naszym testom w celu otrzymania p_wartości. Z powodu, że przy prawdziwej hipotezie zerowej rozkład p_wartości powinien mieć rozkład jednostajny na przedziale(0,1), powtarzamy ten eksperyment więcej razy i wyniki przedstawiamy na wykresie poniżej. W tym przypadku obliczenia powtarzaliśmy 2^{15} razy.



Wyk.1 Rozkład p wartości testów dla wygenerowanych ciagów liczb długości 1000

Widzimy, że rozkład p_wartości dla generatora Mersenne Twistera najbardziej przybliża rozkład jednostajny. Jest on najlepszym ze wszytkich opisanych tu generatorów. Co ciekawe, wysokie wartości dla GLCG mogłyby sugerować, że jest on dobrym generatorem, natomiast rozkład tych p_wartości już nie. Warto zauważyć, że dla BBSG dla testu chi-kwadrat są bardzo bliskie zeru, natomiast dla Kołomogorowa-Smirnowa i Cramera-von Misesa pojedyncze wyższe wartości p_wartości sugerowałyby, że nasze wygenerowane liczby pochodzą z rozkładu jednostajnego.

Zobaczmy jednak jak rozłożą się p_wartości w momencie, gdy każdy generator będzie tworzył 2^{15} liczb pseudolosowych. Tutaj eksperyment powtórzymy 1000 razy.



Wyk.2 Rozkład p wartości testów dla wygenerowanych ciagów liczb długości 2^{15}

O ile zwiększenie ilości generowanych liczb nie wpłynęło bardzo wykres generatora Mersenne Twistera, to w pozostałych generatorach p_wartości są już bardzo bliskie 0. Wpływ na to może mieć np to, że okres naszych generatorów jest mniejszy od ilości stworzonych liczb.

Spojrzyjmy jeszcze na wyniki second-level testing.

	RC4	LCG	GLCG	BBSG	MT
CHI2	0.0	0.0	0.0	0.0	0.67
KS	0.0	0.0	0.0	0.0	0.47
CVM	0.0	0.0	0.0	0.0	0.35

Tabela 1. Wyniki second-level testing dla wybranych generatorów.

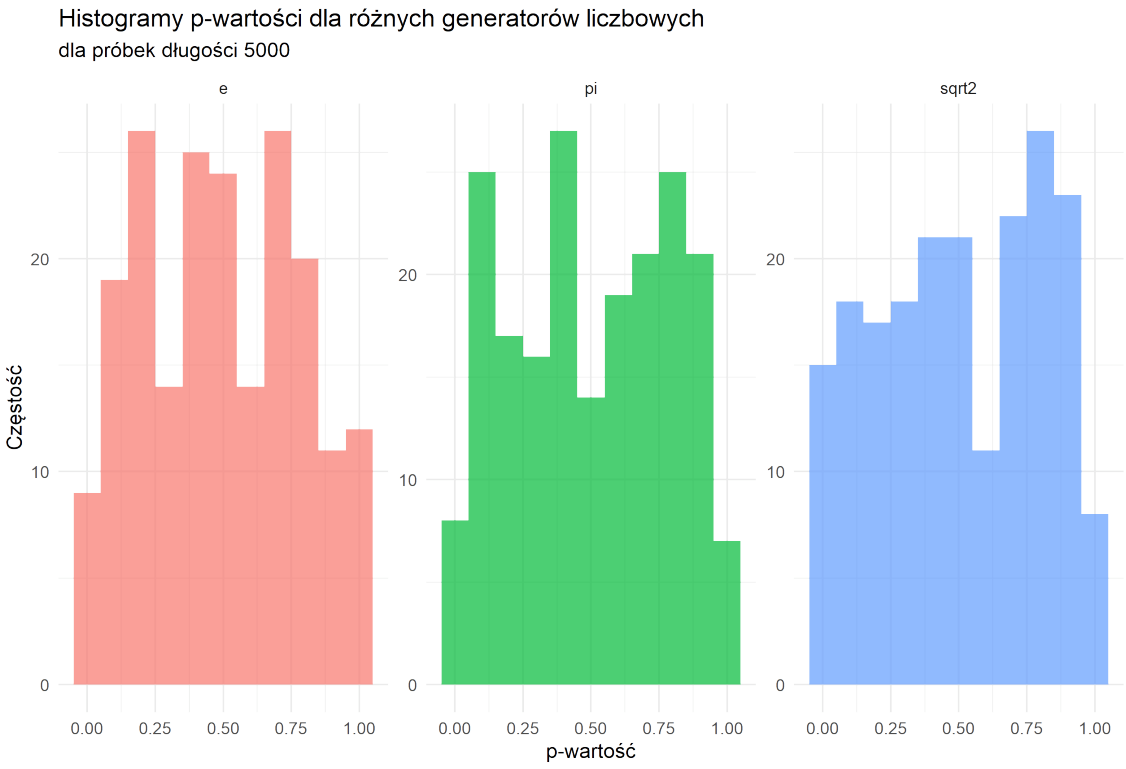
Jak widzimy, nasze przypuszczenia się potwierdziły. Najlepszym generatorem liczb pseudolosowych spośród wszyskich generatorów wybranych przez nas jest generator Mersenne Twistera. Pozostałe generatory dały wyniki zerowe.

W drugiej części zadania sprawdzaliśmy czy liczby π, e, $\sqrt{2}$ mogą posłużyć nam jako generatory liczb. Wykorzystując frequency monobit test oraz binarny zapis tych liczb(milionowe rozwinięcie) otrzymaliśmy wyniki p_wartości:

liczba - generator	p_wartość
π	0.6137
e	0.9269
$\sqrt{2}$	0.8177

Tabela 2. Wyniki frequency monobit test dla trzech liczb niewymiernych.

Wszystkie trzy liczby jako generatory dają po stestowaniu zadowalający wynik. Jednak warto zauważyć, że są to liczby wykorzystujące 1 000 000 miejsc w zapisie binarnym. W celu przeprowadzenia second-level testing dla tych liczb, musimy ich zapis binarny podzielić na mniejsze części.



Wyk.3 Rozkład p wartości dla fragmentów liczb niewymiernych jako generatorów w frequency monobit test

Powyżej widzimy wyniki p_wartości dla frequency monobit test obliczanych dla próbek długości 5000. Ich rozkład nie jest idealnie jednostajny, jednak możemy przypuszczać, że generatory tej długości są dobrym wyborem

w celu tworzenia liczb pseudolosowych. Porównamy jednak jeszcze wyniki second-level testing dla różnej długości próbki generatora.

	n = 10 000	n = 5 000	n = 1 000
π	0.2622	0.7399	0.0
e	0.2757	0.6890	0.0
$\sqrt{2}$	0.3669	0.6215	0.0

Tabela 3. P_wartości dla second-level testing dla trzech liczb niewymiernych.

W powyższej tabeli n oznacza jaką długość miały nasze generatory stworzone z części zapisu tych liczb wymiernych. Widzimy, że generatory długości 5 000 były już wystarczające aby tworzyć dobre pseudolosowe liczby, natomiast n=1000 było za krótką długością.

Odnośnik do kodu

Obiekt	**Kod w Pythonie**	**Kod w R**
Wyk. 1, Wyk.2	generate_and_test_generators() - funkcja wykorzystana do tworzenia wyników	read_pvalues - funkcja do wczytania danych, p_wartosi_wykres - zmienna zawierająca tworzony wykres
Wyk. 3	monobit_fragmenty(5000) - zapisanie p_wartości, przeprowadzenie second-level testing	p_values_data5000 - wczytana ramka danych, liczby_5000 - zmienna zawierająca tworzony wykres
Tabela 1	generate_and_test_generators() - funkcja wykorzystana do tworzenia wyników	-
Tabela 2	frequency_monobit_test() - funkcja wykorzystana do tworzenia wyników, wyświetlenie wyników - pod komentarzem: monobit na digits_number	-
Tabela 3	monobit_fragmenty() - funkcja gdzie zmienną są długość podziałów liczb	-