Pierwszy raport z symulacji Monte Carlo

Jan Kozłowski

19-12-2024

Wprowadzenie

Celem tego projektu jest opisanie i zbadanie różnych generatorów liczb pseudolosowych, oraz testów, które będą sprawdzać "jakość" tych generatorów. Raport zaczniemy od napisiania algorytmów 4 generatorów, GLCG, RC4, Marsa-LFIB4 oraz Ziff98, następnie opisanie 4 testów: Kołmogorowa-Smirnowa, χ^2, testu pokerowego i dyskretnego testu transformaty Fouriera(Discrete Fourier Transform (Spectral) Test)

Generator Fibonacciego

Patrząc na wzór rekurencyjny generatora GLCG możemy zauważyć, że jego wadą będzie duża korelacja pomiędzy kolejnymi zmiennymi. Jendym z możliwości poprawy tej własności jest zwiększenie rekurencji tzn. zamiast olbiczać n-ty wyraz za pomocią wyrazów $n-1, n-2, \ldots, n-i$ będziemy używać wyrazów $n-q_1, n-q_2, \ldots, n-q_i$, gdzie liczby q_j są "duże". Dodatkowo, jako że ostatnim działaniem rekurencji jest wzięcie reszty dzielenia przez M, to działanie, jakie wykonujemy pomiędzy x_j nie musi być dodawanie, więc nasza rekurencja będzie miałą wtedy postać:

$$x_n = x_{n-q_1} \diamond x_{n-q_2} \diamond \ldots x_{n-q_k} \quad mod M,$$

gdzie $\diamond$ jest pewnym działaniem w liczbach całkowitych. Stąd dostajemy dwa możliwe generatory:
-generator Marsagli:$k=4$, $p_1 = 55, p_2 = 119, p_3 = 179, p_4 = 256, M = 2^{10}, \diamond = +$,
-generator Ziffa:$k=4$, $p_1 = 471, p_2 = 1586, p_3 = 6988, p_4 = 9689, M = 2^{10}, \diamond = xor$,

gdzie operator xor dla dwóch liczb to zamienienie ich na postać binarną i nałożenie xor na każdą parę bitów.

W przykładach będziemy używać dla generatora $GLCG$ parametrów $M = 2^{10}$, $k = 3$, $\{a_i\}_{i=1}^k = (3, 7, 68)$ oraz $\{x_i\}_{i=1}^k = (1, 2, 5)$,

Dla generatora $RC(32)$ będziemy używać parametrów $M = 2^5$,$K = (2, 3, 4, 7)$,

Generatory Ziffa i Marsagli wymagają długiej listy parametrów $x_1, \ldots, x_{q_k}$, więc do jej stworzenia użyjemy generatora $GLCG$.

Test pokerowy

W tym teście będziemy dzielić nasz ciąg liczb $X_1, \ldots, X_{5n}$ na podciągi długości 5 i badać ile jest w danym podciągu różnych liczb. Zapiszmy to jako $Y_1, \ldots, Y_n$, teraz będziemy chcieli na tym wektorze przeprowadzić test χ^2, więc jedynym, co nam zostaje do policzenia są prawdopodobieństwa $p_s = P[Y_i = s]$. Prawdopodobieństwo, że losując pięciokrotnie ze zbioru od 1 do M otrzymamy dokładnie s różnych liczb wynosi

$$\frac{M(M-1)\ldots(M-s+1)}{M^5} S_2(5, s)$$

gdzie $S_2(5,s)$ oznacza liczbę podziału zbioru 5-elementowego na s niepustych podzbiorów. Można policzyć, że $S_2(5,1)=1, S_2(5,2)=15, S_2(5,3)=25, S_2(5,4)=10, S_2(5,5)=1$. Mając to, możemy policzyć

$$O_s = \#\{i : Y_i = s\}.$$

Wtedy statystyka

$$\hat{\chi}^2 = \sum_{i=1}^{5} \frac{(O_i - np_i)^2}{np_i},$$

ma rozkład χ^2 z 4 stopniami swobody.

Dyskretny test transformaty Fouriera

Ten test opiera się na dyskretniej transformacie Fouriera, która zamienia ciąg $x_1, \ldots, x_n$ w ciąg $a_1, \ldots, a_n$ w sposób:

$$a_k = \sum_{i=1}^{n} x_i w_n^{-(n-1)k},$$

gdzie $w_n = e^{i\frac{2\pi}{n}}$

Test ten bierze ciąg bitów, więc w naszym przykładzie będziemy musieli zamienić nasz ciąg w takowy: mając ciąg liczb z przedziału $[0,1]$ zamieńmy liczby mniejsze od 0.5 w 0, a większe lub równe w 1. Nazwijmy ten ciąg bitów $X_1, \ldots, X_n$. Teraz zamieńmy go w ciąg $Y_i = 2X_i - 1$ i na ciąg $Y_1, \ldots, Y_n$ zaaplikujmy dyskretną transformację Fouriera otzrybując ciąg $S_1, \ldots, S_n$. Teraz niech dla $i = 1, \ldots, \frac{n}{2}$ mamy $M_i = |S_i|$. Można pokazać, że przy H_0 95% wartości M_i powinno być mniejsze od $T = \sqrt{\ln(\frac{1}{0.05})n}$. Niech $N_0 = 0.95\frac{n}{2}$, oczekiwana ilość wartości mniejszych niż T, a niech $N_1 = \#\{i : M_i < T\}$ to obserwowana ilośc. Licząc

$$d = \frac{N_1 - N_0}{\sqrt{\frac{n*0.95*0.05}{4}}},$$

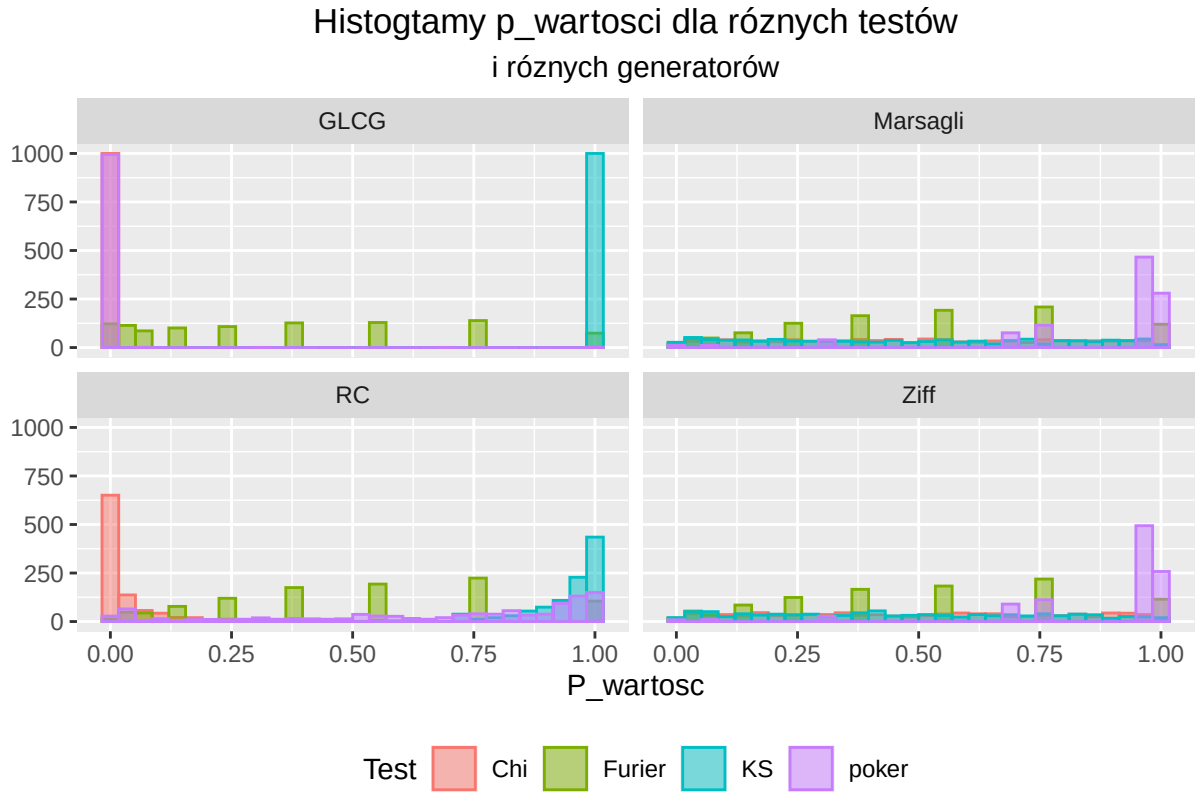
możemy policzyć p_wartość ze wzoru $p_v = erfc(\frac{|d|}{\sqrt{2}})$

Second level testing

Wyobraźmy sobie sytuację, że mamy generator, który za każdym razem zwraca przy testowaniu jego działania tę samą p-wartość, która jest większa niż poziom istotności, wtedy ten generator nie jest dobry, ponieważ generowanie przez niego bity(czyli też p-wartości) powinny być niezależne. Pamiętając o tym spostrzeżeniu, jak i o fakcie, że gdy H_0 jest prawdziwe, to rozkład p-wartości jest jednostwajny na $[0,1]$ możemy stworzyć nowy test nazwany second level testing: podzielny nasz ciąg bitów do testowania na R mniejszych podciągów i każdy z nich przetestujmy otrzymując ciąg p_wattości, na których możemy przeprowadzić test χ^2

Zadanie 1

Teraz zobatrzmy jakie wyniki będą dawać nasze generatory dla różnych testów. Będziemy generować $k = 1000$ p-wartości, każda policzona z ciągu długości $n = 1000$:



Wykres 1

Możemy zauważyć, że p-wartości testu pokerowego dla każdego generatora są bardzo skrajne: albo prawie równe 0 albo 1. Wynikać to może z tego, że losujemy wśród bardzo wielu liczb, ponieważ M jest duże, więc szansa, że w 5-ciu liczbach będą dwie takie same, jest bardzo mała, co przy takiej próbie może dawać duże odchylenia. Kolejną obserwacją jest to, że p_wartości testu Furiera są bardzo dyskretne, tzn skupione w kilku punktach, to może być spowodowane małą próbką. JEśli chcemy porównać wyniki generatorów, to wydaje się, że oba generatory Fibonaciego dają najlepsze wyniki, ale policzmy jeste p-wartości dla second level testing:

	GLCG	RC(32)	Marsagli	Ziff
KS	0	0	0.0189259	0.0085368
Chi	0	0	0.8219373	0.2716187
Poker	0	0	0.0000000	0.0000000
Fourier	0	0	0.0000000	0.0000000

Z omówionych wyżej powodów testy pokerowy i Fouriera dają p-wartości praktycznie równe 0. A dla pozostałych testów najlepiej wypada generator Marsagli.

Zadanie 2

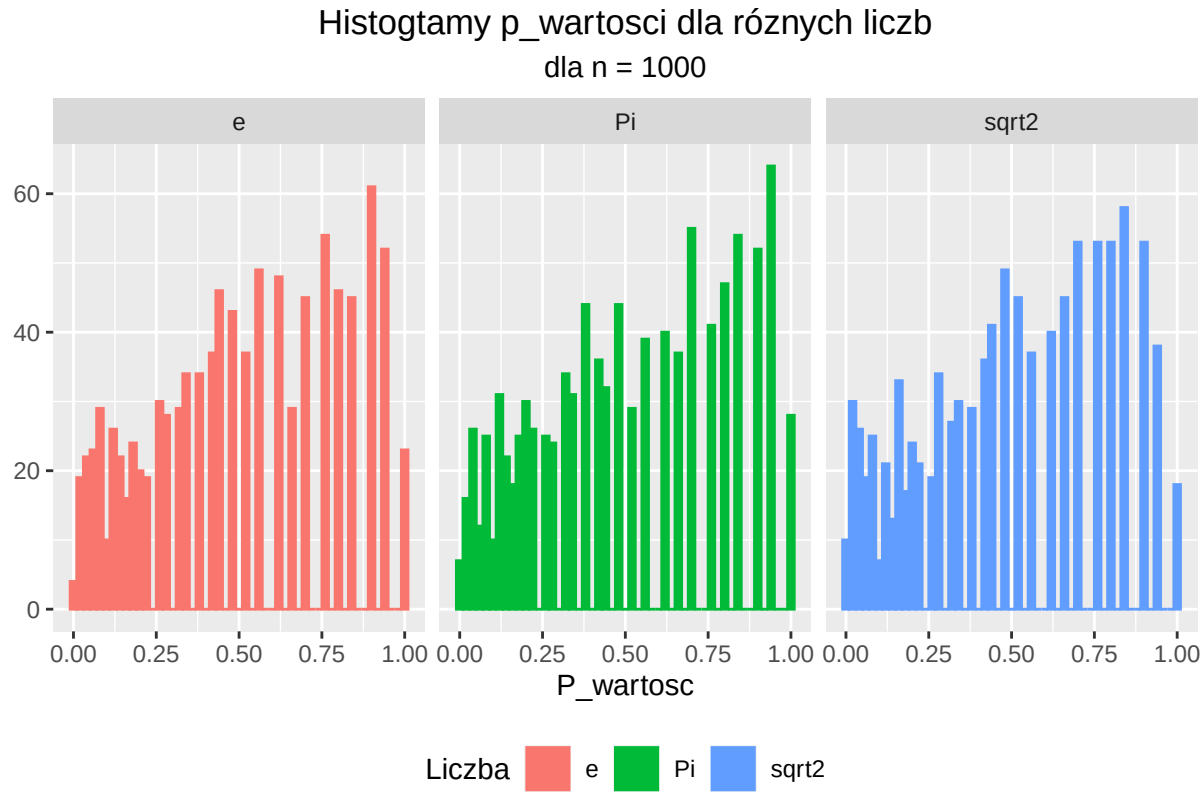
Teraz opiszmny inny sposób generowania liczb w sposób pseudolosowy: weźmy jakąś niewymierną liczbę rzeczywistą i rozpiszmy ją w nieskończonym rozwinięciu dwójkowym. Możemy podejrzewać, że taki ciąg będzie ciągiem niezależnych bitów. Sprawdzimy to na podstawie rozwinięć π,e oraz $\sqrt{2}$ i testu monobit. Zaczniemy od wczytania pierwszego miliona cyfr każdej z tych liczb, następne te ciągi podzieliny na mniejsze ciągi długości $n = 1000$ z każdy taki ciąg bitów $B_1^i, \ldots, B_n^i$ zamieńmy na liczbę

$$S_i = \frac{1}{\sqrt{2}} \sum_{j=1}^{n} B_j$$

Z CTG ciąg S_i dąży do $N(0,1)$, więc możemy policzyć p-warość:

$$p_i = 2(1 - \phi(|S_i|)),$$

gdzie ϕ to dystrybuanta rozkładu normalnego standardowego. Zobaczmy jak wyglądają histogramy tych p-wartości dla naszych liczb π,e i $\sqrt{2}$:

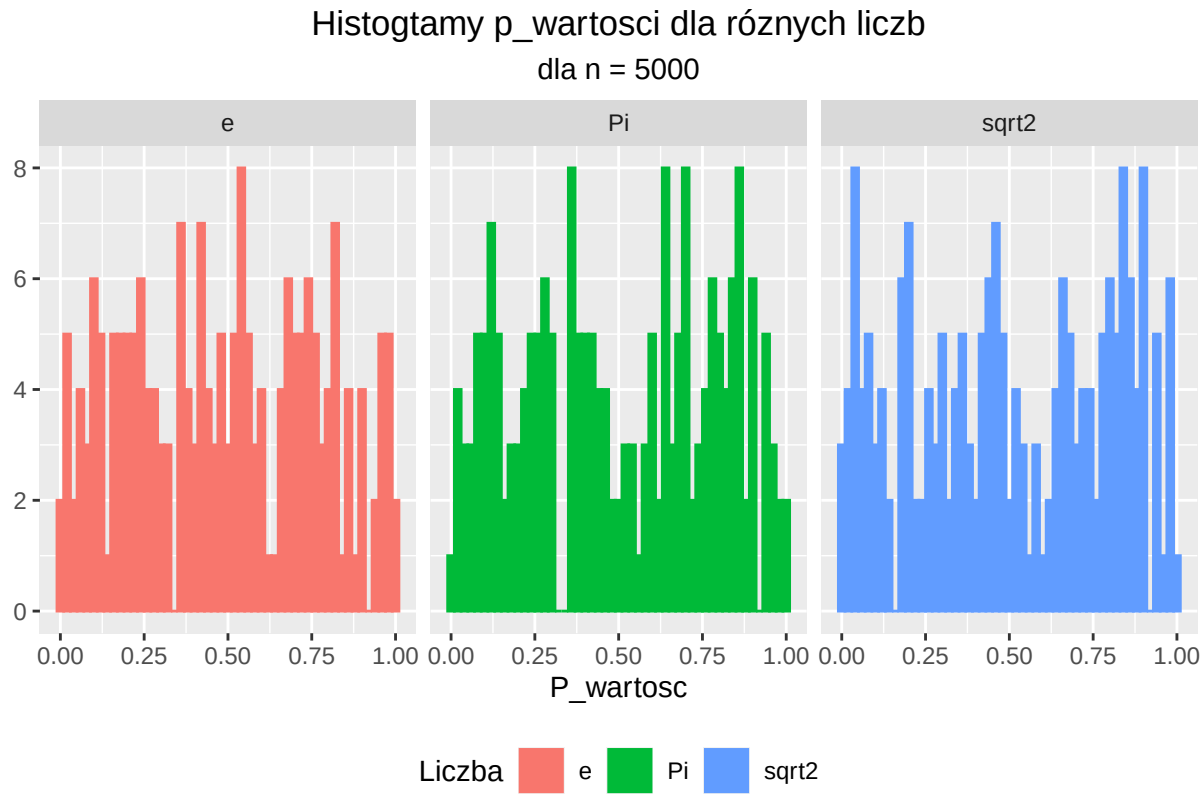


Wykres 2

Widzimy, że liczby te mniej więcej rozkładają się jednostajnie na przedziale $[0,1]$, co mogłoby sugerować, że hipoteza o tym, że nasz początkowy ciąg bitów jest iid. jest prawdziwa, ale wykonajmy jeszcze second level testing i policzmy ich p-wartości:

Pi	e	sqrt2
0	0	0

Wszytkie one są tak małe, że komputer uważa je za równe zero, przyczyną tego może być za mała liczba n, która sprawia, że rozkład S_i nie jest jeszcze "blisko" rozkładu normalnemu, dlatego ustawmy $n = 5000$ i powtóżmy nasze obliczenia:



Wykres 3

Widzimy, że tym razem rozkład tych p-wartości jest bardziej równomiernie równomiernie rozłożony, zobatrzmy jak to wpływa na second level testing i jego p-wartość:

Pi	e	sqrt2
0.7399183	0.689019	0.6215056

Tym razem w żadnym przypadku nie możemy odżucić H_0, lecz musimy pamiętać, że zwiększając n mniejszamy ilość zmiennych S_i, co też wpływa na p_wartość.

Tabela wywoływania funkcji

W tym projekcjie postanowiłem najpierw gererować liczby pseudolosowe i testować je "pierwszo poziomowo" w pythonie, później zapisywać wyniki w postaci .csv, a wykresy i second level testong przeprowadziać w R. Poniżej jest przedstwaiona tabela z nazwami funkcji, które zapisują dane to wykesu o danej nazwie.

	nazwa pliku	nazwa funkcji	parametry
Wykres 1	wyniki_zad1.csv	policz_i_zapisz_wyniki_zad1	n=1000,k=1000
Wykres 2	wyniki_zad2_n_1000.csv	read_numbers_and_save_p_values	n=1000
Wykres 3	wyniki_zad2_n_5000.csv	read_numbers_and_save_p_values	n=5000