Zadanie 1

Ciągi bitowe generowałem za pomocą języka C++. Jako gorszy generator (LCG) użyłem funkcji rand(), a jako lepszy implementacji Mersenne Twister z biblioteki standardowej C++. Zgodnie z https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf aby wszystkie testy poprawnie działały potrzebujemy co najmniej 1.35 mln bitów. Ja generowałem ich 1.5 mln.

Test name	Result value (P-value)	Status
1. Frequency (Monobit) Test	0.651268982440641	Passed
2. Frequency Test within a Block	0.31512111736589876	Passed
3. Runs Test	1.5505451789882516	Failed
4. Test for the Longest Run of Ones in a Block	0.7107959452392546	Passed
5. Binary Matrix Rank Test	0.32424774256565514	Passed
6. Non-overlapping Template Matching Test	0.06696452972740533	Passed
7. Overlapping Template Matching Test	0.7732257193252015	Passed
8. Maurer's "Universal Statistical" Test	0.3908055270009183	Passed
9. Linear Complexity Test	0.5827913202062671	Passed
10. Serial Test	P-value 1: 0.6784666184169064 P-value 2: 0.44964922304766364	Passed
11. Approximate Entropy Test	0.7779743832683992	Passed
12. Cumulative Sums (Cusum) Test	P-value Forward: 0.8209830287167155 P-value Reverse: 0.7996623888954619	Passed
13. Random Excursions Test	0.07325581912234776	Passed
14. Random Excursions Variant Test	0.03442689241312491	Passed

Rysunek 1: Wyniki testu NIST dla generatora LCG

Jak widać gorszy generator poradził sobie względnie dobrze, nie przeszedł tylko testu nr 3. Test ten sprawdza czy dany generator wystarczająco często oscyluje między 0, a 1, tzn. czy ilość sekwencji zer i jedynek o różnych długościach jest zgodna z oczekiwaną. Generator LCG cechuje się tym, że generuje liczby równomiernie na całym swoim przedziale, w przypadku funkcji rand() w C jest to przedział [0,RAND_MAX]. Jeśli więc chcemy wygenerować liczby całkowite na przedziale [a,b) to będą one rozłożone równomiernie tylko wtedy, gdy

$b \mid \text{RAND\_MAX}$. Na moim komputerze $\text{RAND\_MAX} = 2147483647$. Oczywiście $2 \nmid 2147483647$, a więc generator ten nie będzie dobry w generowaniu losowych ciągów bitowych.

Test name	Result value (P-value)	Status
1. Frequency (Monobit) Test	0.35444598024979956	Passed
2. Frequency Test within a Block	0.6035534928239878	Passed
3. Runs Test	0.17041902314485347	Passed
4. Test for the Longest Run of Ones in a Block	0.039597459179885414	Passed
5. Binary Matrix Rank Test	0.5111453231294695	Passed
6. Non-overlapping Template Matching Test	0.9115494720856038	Passed
7. Overlapping Template Matching Test	0.5240173282405224	Passed
8. Maurer's "Universal Statistical" Test	0.9111592737794687	Passed
9. Linear Complexity Test	0.48287643374860134	Passed
10. Serial Test	P-value 1: 0.25552117399489904 P-value 2: 0.17131211727926632	Passed
11. Approximate Entropy Test	0.32202212712799416	Passed
12. Cumulative Sums (Cusum) Test	P-value Forward: 0.2946475627124663 P-value Reverse: 0.8073260647090819	Passed
13. Random Excursions Test	0.01894738359118387	Passed
14. Random Excursions Variant Test	0.043567285873641004	Passed

Rysunek 2: Wyniki testu NIST dla generatora Mersenne Twister

Generator Mersenne Twister zgodnie z oczekiwaniami przeszedł wszystkie testy.

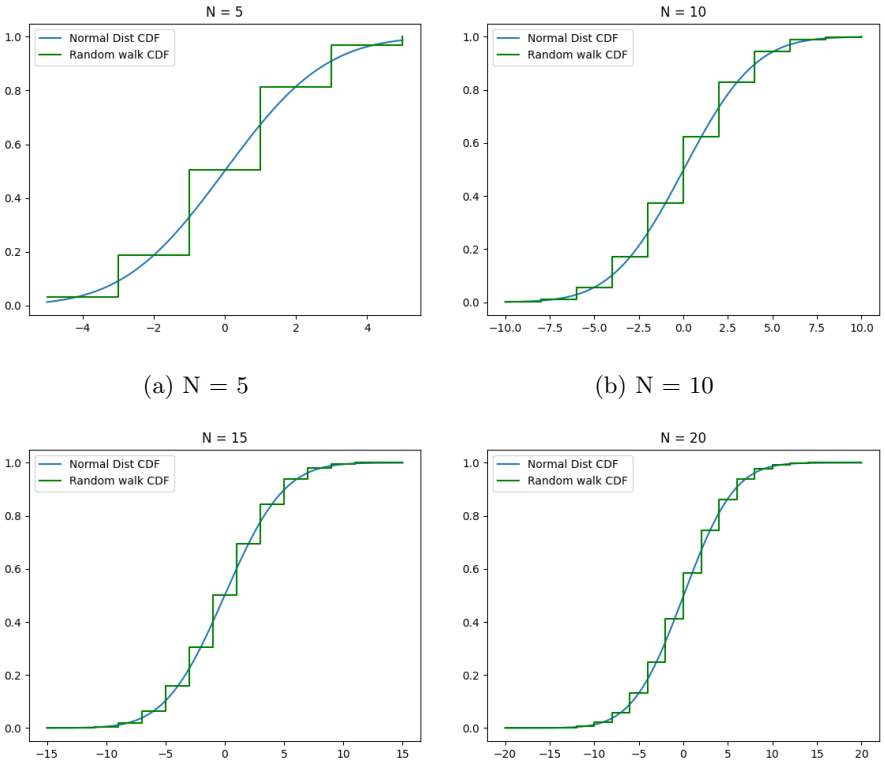
Test name	Result value (P-value)	Status
1. Frequency (Monobit) Test	0.005188607552315094	Failed
2. Frequency Test within a Block	0.017205950425851387	Passed
3. Runs Test	1.3610484905292686	Failed
4. Test for the Longest Run of Ones in a Block	1.3868112373787695e-8	Failed
5. Binary Matrix Rank Test		Error
6. Non-overlapping Template Matching Test		Error
7. Overlapping Template Matching Test		Error
8. Maurer's "Universal Statistical" Test		Error
9. Linear Complexity Test		Error
10. Serial Test		Error
11. Approximate Entropy Test	2.321345246485639e-15	Failed
12. Cumulative Sums (Cusum) Test	P-value Forward: 0.007300868808884253 P-value Reverse: 0.5051886075523151	Failed
13. Random Excursions Test		Error
14. Random Excursions Variant Test		Error

Rysunek 3: Wyniki testu NIST dla generatora hashu SHA1 mojego nazwiska

Oczywiście, tak jak pisałem wyżej, hash SHA1 jest zbyt krótki (322 bitów) aby przejść testy, do których wymagane jest 1.3 mln bitów, więc hash mojego nazwiska przeszedł tylko jeden test, podczas gdy na większości miał error.

Zadanie 2

Poniżej są wykresy wygenerowane za pomocą symulacji napisanej w języku python. Symulacja polega na niezależnym wygenerowaniu 10000 razy zmiennej S(N), zliczeniu wystąpień poszczególnych wartości, a następnie wygenerowaniu dystrybuanty.



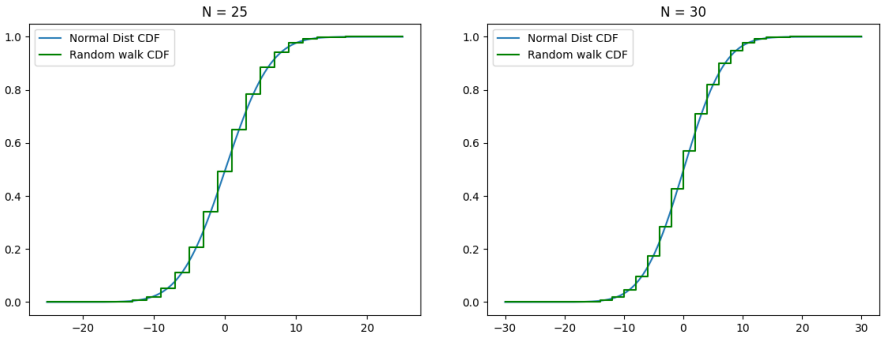
(a) N = 5

(b) N = 10

(c) N = 15

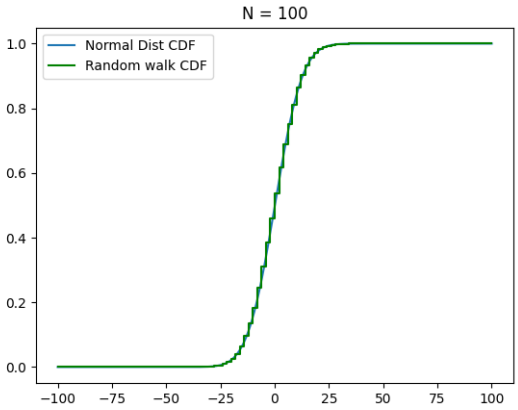
(d) N = 20

Rysunek 4: Wykresy przedstawiające wyniki dla $N \in [5, 10, 15, 20]$



(a) N = 25 (b) N = 30

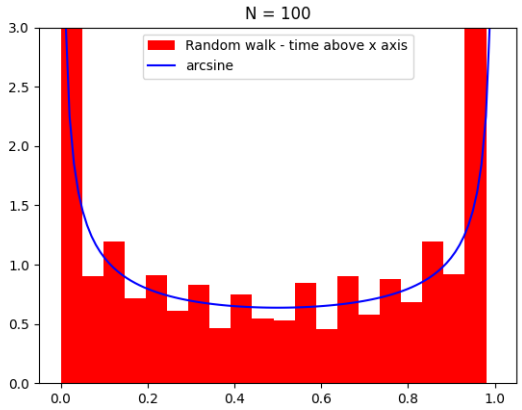
Rysunek 5: Wykresy przedstawiające wyniki dla $N \in [25, 30]$



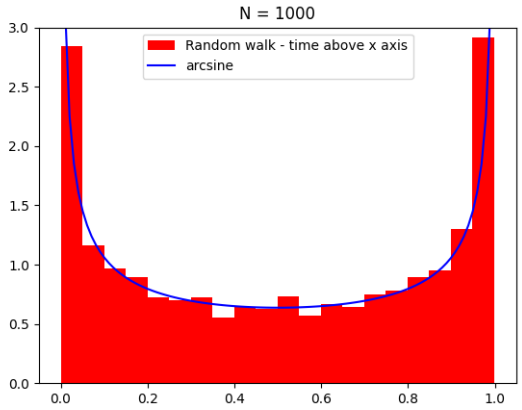
Rysunek 6: Wykres przedstawiający wyniki eksperymentu dla N = 100

Zgodnie z intuicją, wraz z rosnącymi wartościami N symulacja "random walk" coraz dokładniej aproksymuje rozkład normalny.

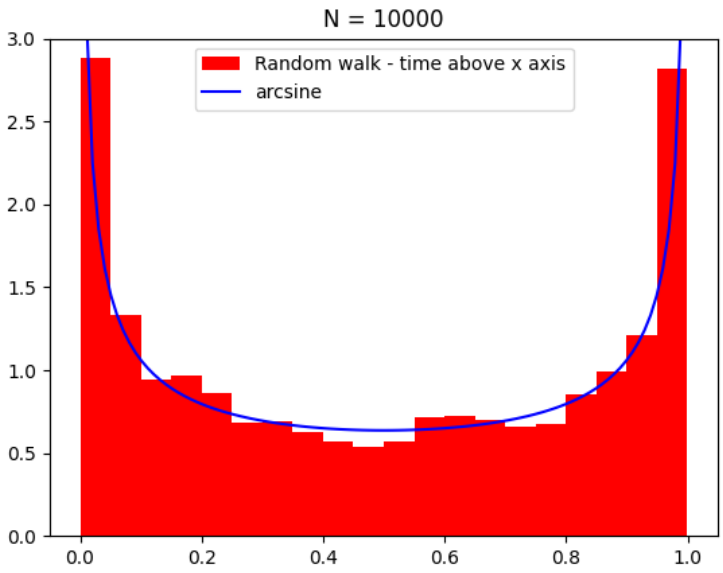
Zadanie 3



Rysunek 7: Wykres przedstawiający wyniki eksperymentu dla N = 100



Rysunek 8: Wykres przedstawiający wyniki eksperymentu dla N = 1000



Rysunek 9: Wykres przedstawiający wyniki eksperymentu dla N = 10000

Nasz eksperyment wraz z rosnącym N coraz ściślej przybliża rozkład arcsinusa.