**AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE**

**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

KATEDRA AUTOMATYKI

Modelowanie układów fizycznych i biologicznych

# *Generator liczb losowych*
# *Random number generator*

Autor: *Żaneta Błaszczuk, Rafał Kozik, Filip Kubicz, Jakub Nowak, Jakub Porębski*
Kierunek studiów: *Automatyka i Robotyka*
Opiekun pracy: *dr inż. Ireneusz Wochlik*

Kraków, 2014

# 1. Wstęp

## 1.1. Cel zastosowania generatorów liczb pseudolosowych

Liczby pseudolosowe przede wszystkim wykorzystuje się w 3 dziedzinach:

1. Obliczeniach numerycznych – wykorzystywane w metodach obliczeniowych, które nie wymagają dużej ilości liczb o danym rozkładzie.

2. Kryptografia – wykorzystywane do generowania kluczy prywatnych.

3. Złudzenie losowości w grach – wykorzystując generatory gracz ma poczucie przewidywalności i niepowtarzalności.

## 1.2. Liczby pseudolosowe

Liczbami pseudolosowymi nazywamy liczby wykazujące cechy liczb prawdziwie losowych uzyskanych poprzez działanie algorytmu. Zaletą korzystania z takich liczb jest możliwość szybkiego pozyskiwania rezultatów, ograniczona mocą obliczeniową komputera.

## 1.3. Generatory liczb pseudolosowych

Liczby pozyskiwane są z odpowiednich algorytmów, obliczających kolejną liczbę na podstawie poprzedniego wyniku. W związku z tym konieczne jest podanie pierwszej liczby. W praktyce generatory wykorzystują jako pierwszą liczbę aktualny czas.

# 2. Testowanie generatorów

## 2.1. Cel testowania generatorów

Generatory testowane są przede wszystkim by odrzucić te z nich, które dają wyniki dalekie od losowych. Najważniejszym założeniem jest, by nie dopuścić do przepuszczenia złego generatora, nawet kosztem pomyłkowego odrzucenia działającego. W dzisiejszych czasach mamy do dyspozycji bardzo wiele różnych generatorów. Wybierając generator źle działający możemy mieć do czynienia z bardzo dotkliwymi skutkami dla użytkownika. Dlatego też zostały sformułowane „podstawowe cechy dobrego generatora”:

1. Jednorodność – prawdopodobieństwo wystąpienia 1 lub 0 wynosi 0,5 w każdym punkcie generowanego ciągu bitów.

2. Skalowalność - każdy podciąg ciągu, który zakończył się pozytywnym testem, również powinien zakończyć się tym samym pozytywnym testem.

3. Zgodność – zachowanie generatora powinno dawać podobne rezultaty niezależnie od początkowej wartości.

### 2.2. Analiza rezultatów

Testy, na których się oparłem pochodzą z pakietu DieHard, oraz z jego następcy DieHarder, uznawanego za jednego z najlepszych pakietów. Rozpowszechniony jest na licencji GPL, co pozwala na zgłębienie kodu oraz modyfikacji, jeśli zajdzie taka potrzeba. Atutem DieHard jest też przejrzysty i powtarzalny sposób prezentacji testów. DieHard wynik przedstawia nam w postaci p-wartości, która informuje nas o odchyleniu rozkładu od wartości oczekiwanej. Dobre generatory w testach zbliżają się ze swoją p-wartością do 1. Natomiast wartością graniczną, przy której generator przestaje być losowy jest $p - warto = 0,05$. Ten sam test powtarza się zwykle dla danego generatora n razy (zazwyczaj n = 100), w celu uniknięcia zdarzenia, że generator raz przeszedł dany test. Następnie otrzymane p-wartości przepuszcza się przez test Kolmogorova-Smirnova, by sprawdzić czy ich wartości mają rozkład równomierny.

## 3. Przykładowe testy

### 3.1. Wybrane generatory do testów

#### KISS – (Keep It Simple Stupid)

Generator złożony z trzech prostych generatorów. Ideą jego jest bycie szybkim i prostym. KISS składa się z:

$$x(n) = a \cdot x(n-1) + 1 \ mod \ 2^{32} \tag{1}$$

$$y(n) = y(n-1)(I + L^{13})(I + R^{17})(I + L^{5}) \tag{2}$$

$$z(n) = 2 \cdot z(n-1) + z(n-2) + carry \ mod \ 2^{32} \tag{3}$$

Generator ten nadaje się do programowania w asemblerze, gdzie trwa około 200 nanosekund przy procesorze Pentium 120.

#### Generator System Microsoft Fortran

Generator oparty na 32 bitach opisany wzorem:

$$X(n) = 48271 \cdot x(n-1) \ mod \ 2^{31} - 1, \tag{4}$$

zbliżony do Lehmer, charakteryzuje się jednak tym, że zaproponowano w nim lepszy mnożnik.

#### Odwrócony generator

Opisany przez Einchenauer-Herrmann. Obliczenia zajmują dużo więcej czasu. Sprawdza się gorzej, niż klasyczny RNG mod $2^{32}$

### 3.2. Wybrane testy

#### Test urodzin

Polega on na „paradoksie urodzin", czyli prawdopodobieństwie, że w grupie osób, znajdą się osoby które mają urodziny tego samego dnia. Testowana grupa ma 512 osób, a dzień określa 24 bitowa liczba. Rozkład powinien być podobny do rozkładu Poissona.

**Minimum Distance test**

Wybiera się 8000 losowych punktów w kwadracie o boku 10000. Oblicza się $d$ (minimalną odległość pomiędzy $\frac{n^2-n}{2}$ parami punktów). Rozkład powinien przedstawiać rozkład równomierny.

**Sums test**

Polega na przekształceniu losowych liczb całkowitych na liczby zmiennoprzecinkowe. Następnie wylicza się sumy pokrywających się podciągów składających się ze 100 elementów. Otrzymany ciąg sum powinien mieć rozkład normalny.

# 4. Testowanie

## 4.1. Test urodzin

| generator | Finalna p-wartość | Wynik |
|---|---|---|
| KISS | 0.934361 | PASSED |
| System Microsoft Fortran | 0.995980 | PASSED |
| Odwrócony generator | 0.804086 | PASSED |

Wszystkie generatory zdały test.

## 4.2. Minimum Distance test

| generator | Finalna p-wartość | Wynik |
|---|---|---|
| KISS | 0.309341 | PASSED |
| System Microsoft Fortran | 0.772850 | PASSED |
| Odwrócony generator | 0.046785 | FAILED |

Odwrócony generator nie zdał testu. Z testem dobrze poradził sobie natomiast SMF.

## 4.3. Sums test

| generator | Finalna p-wartość | Wynik |
|---|---|---|
| KISS | 0.593360 | PASSED |
| System Microsoft Fortran | 0.808943 | PASSED |
| Odwrócony generator | 0.315619 | PASSED |

Wszystkie generatory zdały test, chodź najlepiej poradził sobie SMF.

## 4.4. Podsumowanie

Z wybranych generatorów najlepiej sprawuje się generator System Microsoft Fortran, który zakończył każdy test pozytywnie i jego p-wartość była w tych testach najbardziej zbliżona do 1.