WSTĘP DO MATEMATYKI FINANSOWEJ

BARTOSZ KOŁODZIEJEK

WYDZIAŁ MATEMATYKI I NAUK INFORMACYJNYCH

Laboratoria 1

Projekt „NERW 2 PW. Nauka - Edukacja - Rozwój - Współpraca" współfinansowany ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Zadanie 10 pn. „Modyfikacja programów studiów na kierunkach prowadzonych przez Wydział Matematyki i Nauk Informacyjnych", realizowane w ramach projektu „NERW 2 PW. Nauka - Edukacja - Rozwój - Współpraca", współfinansowanego jest ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

17 lutego 2025

Legenda: – Definicja, **TW.** – Twierdzenie, – Przykład, – Uwaga, **LEM.** – Lemat, – Oznaczenie

1. Laboratoria

1.1. L1 - Generowanie liczb pseudolosowych.

(1) Generatory pseudolosowych liczb całkowitych oraz z rozkładu U([0, 1]).

- Generatory liniowe są jedną z najstarszych metod generowania liczb pseudolosowych i bazują na rekurencyjnym wzorze:

$$x_{n+1} = (a_1 x_n + a_2 x_{n-1} + \ldots + a_k x_{n-k+1} + a_0) \bmod m,$$

  gdzie a_i oraz m są ustalonymi liczbami całkowitymi. oraz $(x_0, \ldots, x_{k-1})$ pełni rolę *ziarna* (seed) generatora.
  Przypadek szczególny dla $k = 1$ daje równanie: $k = 1$:

$$x_{n+1} = (a_1 x_n + a_0) \bmod m.$$

  Łatwo widać, że jest to ciąg okresowy, a jego okres wynosi co najwyżej m. Generator ten jest oczywiście beznadziejny (wykres $x_{n+1} \sim x_n$ jest mało losowy), ale dużo mniej niż mogłoby się wydawać. W tym przypadkiem rolę *ziarna* pełni tylko x_0.
  Żeby otrzymać liczbę całkowitą z przedziału $\{0, \ldots, N\}$, $N < m$, stosujemy operację $x_n \bmod N$. Liczbę z przedziału $[0, 1]$ otrzymamy dzieląc x_n przez $m - 1$.
- Każdy generator liczb pseudolosowych ma skończony okres, czyli liczbę iteracji, po której ciąg zaczyna się powtarzać. Wynika to z ograniczonej liczby stanów, jakie może przyjąć komputerowa reprezentacja zmiennych. Długi okres jest kluczowy dla zapewnienia wysokiej jakości losowości.
- Dobre praktyki: (`http://www0.cs.ucl.ac.uk/staff/D.Jones/GoodPracticeRNG.pdf`)
  - **Unikaj wbudowanych generatorów systemowych.** Mogą posiadać niedoskonałości i brak gwarancji stabilności między wersjami systemu.
  - **Używaj sprawdzonych bibliotek RNG.** Istnieją standardowe narzędzia przechodzące rygorystyczne testy statystyczne (np. TestU01, DieHard, DieHarder).
  - **Zwróć uwagę na ustawienie ziarna.** Różne ziarna prowadzą do różnych sekwencji liczb, co jest kluczowe dla replikowalności wyników (np. w analizie finansowej lub symulacjach Monte Carlo). Ciągi liczb pesudolosowych są ponumerowane tzw. ziarnem (seed). Ziarno może być ciągiem kilku liczb. Jako ziarno można wygenerować zmienną z innego generatora lub po prostu systemowy time(). Szczególnie istotne jeśli generujemy na wielu niezależnych maszynach (paradoks urodzin: ile minimalnie osób należy wybrać, żeby prawdopodobieństwo znalezienia wśród nich co najmniej dwóch osób obchodzących urodziny tego samego dnia było większe od 1/2? $n = 23$)
- Jednym z najczęściej stosowanych generatorów jest *Mersenne Twister* (MT), który posiada bardzo długi okres ($2^{19937} - 1$), ale ma również wady, np. nie przechodzi niektórych testów losowości. Krytykę tego generatora przedstawiał m.in. George Marsaglia, pionier w dziedzinie badań nad generatorami pseudolosowymi.

(2) **Testy jakości generatorów liczb losowych**

- **Testy Diehard i Dieharder** `https://webhome.phy.duke.edu/~rgb/General/dieharder.php`
  Testy Diehard zostały stworzone przez George'a Marsaglię i stanowią zestaw rygorystycznych testów oceniających jakość generatorów liczb pseudolosowych. Obejmują one m.in. testy:
  - Test "Birthday Spacings" – sprawdza rozkład losowych odstępów,
  - Test "Overlapping Permutations" – bada równomierność występowania permutacji liczb,
  - Test "Monkey Test" – analizuje częstotliwość ciągów bitowych,
  - Test "Runs Test" – ocenia długość i częstotliwość ciągów rosnących i malejących.
  - itd

  Dieharder to unowocześniona wersja testów Diehard, rozwinięta przez Roberta G. Browna, zawierająca dodatkowe testy oraz poprawki w stosunku do oryginalnego zestawu.
- **Jak uruchomić testy Dieharder?**
  Testy Dieharder są dostępne jako pakiet w systemach Linux i można je zainstalować np. poprzez:

```
sudo apt install dieharder
```

  Aby przeprowadzić testy dla określonego generatora liczb losowych, można użyć polecenia:

```
dieharder -a -g 202
```

gdzie:

– -a – uruchamia wszystkie dostępne testy,

– -g 202 – wybiera generator liczb losowych (np. /dev/random lub niestandardowy generator).

Do przetestowania własnego generatora można wygenerować plik binarny i przekierować jego dane do testera:

```
./moj_generator | dieharder -a -g 200
```

(3) Generowanie liczb pseudolosowych z różnych rozkładów.
- Metoda odwracania dystrybuanty: Jeśli F jest ściśle rosnącą i ciągłą dystrybuantą oraz $U \sim \mathrm{U}([0,1])$, to $F^{-1}(U)$ ma rozkład o dystrybuancie F.
- Czasem można generować z danego rozkładu korzystając z jego interpretacji. Np. dla rozkład dwumianowego $\mathrm{b}(n,p)$ mamy poniższy algorytm

```
X = 0
For i = 1 to n do
Generuj U o rozkładzie równomiernym U(0,1)
If U < p then X = X + 1
Return X
```

- Ogólny algorytm generowania liczb z rozkładu dyskretnego $\{x_i, p_i\}_i$. Niech

$$q_0 = 0, \quad q_i = \sum_{j=1}^{i} p_j.$$

```
Generuj U o rozkładzie równomiernym U(0,1).
Znajdz i, dla którego        q_{i-1}< U <= q_i.
Zwróć x_i.
```

- Metoda eliminacji. Załóżmy, że wykres gęstości f mieści się w prostokącie $[a,b] \times [0,d]$.

```
Generuj X z rozkladu U(a,b) oraz Y z rozkladu U(0,d)
                                  tak dlugo az Y<= f(X)
Zwroc X.
```

- Modyfikacja metody eliminacji. Zakładamy, że $f(x) \leq ag(x)$, gdzie g to gęstość z której umiemy generować.

```
Generuj X z rozkladu o gęstości g oraz Y z rozkladu U(0,1)
                        tak dlugo az Y<= f(X) / (a g(X))
Zwroc X.
```

(4) Zadania:

(a) Zaimplementuj liniowy generator dla $k = 1$, $m = 2^{35}$, $a_1 = \lfloor \pi \cdot 10^9 \rfloor$ oraz $a_0 = \lfloor e \cdot 10^9 \rfloor$ i wygeneruj na jego podstawie dużo liczb z przedziału $[0,1]$. Obejrzyj histogram.

(b) Korzystając z metody eliminacji (lub jej modyfikacji), wygeneruj zmienne losowe z rozkładu o gęstości

$$f(x) = \frac{1 + \cos(2\pi x)}{1 + e^{-2\pi^2}} \frac{e^{-x^2/2}}{\sqrt{2\pi}}.$$

(c) Wygeneruj ciąg zmiennych losowych z rozkładu Poissona. Na dwa sposoby, ale nie korzystaj z wbudowanego algorytmu! (Przypomnij sobie definicję procesu Poissona.)

(d) Wygeneruj dużo liczb losowych z rozkładu o dystrybuancie

$$F(x) = \frac{1}{3} F_{\mathrm{b}(10,1/3)}(x) + \frac{1}{3}\Phi(x) + \frac{1}{3} F_{\mathrm{EXP}(1)}(x).$$

Porównaj na wykresie dystrybuantę empiryczną (*ecdf*) z teoretyczną.

(e) Poczytaj `http://www.cse.yorku.ca/~oz/marsaglia-rng.html` i zaimplementuj w Pythonie któryś z opisanych generatorów (np. KISS, czyli Keep it Simple Stupid) lub np. ten

```
/* Public domain code for JKISS RNG */
static unsigned int x = 123456789, y = 987654321, z = 43219876,
c = 6543217; /* Seed variables */
unsigned int JKISS()
{
        unsigned long long t;
        x = 314527869 * x + 1234567;
        y ^= y << 5;
```

```
        y ^= y >> 7;
        y ^= y << 22;
        t = 4294584393ULL * z + c;
        c = t >> 32;
        z = t;
        return x + y + z;
}
Jeśli chcemy liczbę z (0,1), to robimy  (4294967296 = 2^{32} - 1)
double x;
x = JKISS() / 4294967296.0;
```

- Kod C można wywoływać w Pythonie na kilka sposobów, `https://scipy-lectures.org/advanced/interfacing_with_c/interfacing_with_c.html`, ale ZDECYDOWANIE POLECAM zaprzyjaźnić się z Cythonem lub Numbą:
  - **Cython:** Pozwala na przyspieszenie kodu Pythona poprzez kompilację do kodu C.
  - **Numba:** Alternatywa dla Cythona oparta na JIT (Just-In-Time) compilation. Umożliwia kompilację Pythona do kodu zoptymalizowanego pod kątem wydajności w czasie wykonywania.
- Porównaj szybkość generowania z wbudowanym generatorem w Pythonie.
- Poświęć chwilę by zrozumieć jaka idea stoi za generatorami KISS.

(f) Przeprowadź testy DieHarder na generatorze wpudowanym w Pythonie oraz na generatorach z (a) i (e).