Міністерство освіти і науки України
Національний авіаційний університет
Навчально-науковий інститут комп'ютерних інформаційних технологій
Кафедра комп'ютеризованих систем управління

Лабораторна робота №1
з дисципліни «Імітаційне моделювання»
на тему «Побудова імітаційної моделі
генератора псевдовипадкових чисел (ГПВЧ).
Перевірка якості роботи генератора»

Виконав:
студент ННІКІТ
групи СП-325
Клокун В. Д.
Перевірила:
Марченко Н. Б.

Київ 2019

1. Мета роботи

Ознайомитись з еталоном функціонування генератора псевдовипадкових чисел; побудувати імітаційну модель функціонування генератора псевдовипадкових чисел на основі лінійного конгруентного методу та здійснити перевірку якості роботи створеного генератора псевдовипадкових чисел.

2. Хід роботи

2.1. Побудова імітаційної моделі

ЗАВДАННЯ Побудувати імітаційну модель, яка відображає роботу генератора псевдовипадкових чисел. Отримати послідовність псевдовипадкових чисел на інтервалі (0; 1). Використати мультиплікативний метод.

Щоб побудувати модель лінійного конгруентного генератора псевдовипадкових чисел, необхідно обрати 4 цілих числа, які називають параметрами:

$$
\begin{aligned}
\text{Модуль} \quad & m, \quad 0 < m. \\
\text{Множник} \quad & a, \quad 0 \leqslant a < m. \\
\text{Інкремент} \quad & c, \quad 0 \leqslant c < m. \\
\text{Початкове значення} \quad & X_0, \quad 0 \leqslant c < m.
\end{aligned}
$$

Сам генератор описується таким рекурентним співвідношенням:

$$X_{i+1} = (aX_i + c) \bmod m, \quad n \geqslant 0.$$

Для побудови імітаційної моделі скористаємось оновленими параметрами MINSTD, а саме:

1. Модуль $m = 2^{31} - 1 = 7FFFFFFF_{16}$, тобто є 31-м числом Мерсенна M_{31}, яке до того ж є простим.
2. Множник $a = 48271$, просте число.
3. Інкремент $c = 0$.

Оскільки інкремент $c = 0$, при використанні таких параметрів отримуємо частковий випадок лінійного конгруентного генератора — генератор Леймера. Його максимальний період дорівнює $m - 1$. Генератор Леймера, який використовує параметри MINSTD, матиме максимально можливий період $m-1$, оскільки ці параметри відповідають умовам максимально можливого періоду. Початкове значення X_0 генеруватимемо за допомогою криптографічно стійкого генератора, вбудованого в операційну систему. Так як m — просте число, то будь-яке початкове значення X_0 буде взаємно простим до m і не зменшуватиме якість згенерованих чисел.

Обравши параметри, будуємо імітаційну модель, яка відображає роботу лінійного конгруентного генератора псевдовипадкових чисел. Імітаційна модель реалізована у вигляді класу, написаного на мові програмування Python (лістинг 2.1).

Лістинг 2.1: Реалізація моделі лінійного конгруентного генератора

```python
# A class implementing an LCG and relevant methods
class LCG:
    rand_seq = []

    # generator
    def lcg(self, m, a, c, seed):
        while True:
            seed = (a * seed + c) % m
            yield seed

    def __init__(self,
                 m = 0x7FFFFFFF, # modulus M (as per MINSTD), Mersenne 31, prime
                 a = 48271, # multiplier a, prime number
                 c = 0, # increment c
                 seed = random.SystemRandom().randint(0, 0xFFFFFFFF)):
        self.m, self.a, self.c, self.seed = m, a, c, seed
        self.rand_seq = self.lcg(self.m, self.a, self.c, self.seed)

    def randint(self):
        return next(self.rand_seq)

    def randfloat(self):
        return next(self.rand_seq) / self.m
```

Щоб побудувати послідовність випадкових чисел в інтервалі $(0; 1)$, необхідно використати функцію `LCG.randfloat()`, яка нормалізує згенеровані випадкові цілі числа до чисел з бажаного інтервалу за формулою $x_{i+1} = \frac{X_{i+1}}{m}$. Далі треба зберегти згенеровані числа і записати їх у список (лістинг 2.2).

Лістинг 2.2: Приклад використання функції `LCG.randfloat()` для генерації 10000 випадкових чисел

```python
lcg = LCG()
seq_float = [lcg.randfloat() for x in range(10000)]
```

2.2. Перевірка якості роботи генератора

ЗАВДАННЯ Виконати перевірку якості роботи генератора на рівномірність розподілу псевдовипадкових чисел на інтервалі $(0; 1)$, використовуючи критерій Пірсона та частотний тест.

2.2.1. Частотний тест

Нехай μ_x — математичне сподівання значення змінної x, а σ_x — її дисперсія. Частотний тест дозволяє з'ясувати, скільки чисел потрапило в інтервал $(\mu_x - \sigma_x; \mu_x + \sigma_x)$. Так як математичне сподівання ідеального генератора $\mu_i = 0{,}5$, а його дисперсія $\sigma_i = 0{,}2887$, то для ідеального генератора випадкових чисел цей інтервал такий:

$$(\mu_x - \sigma_x; \mu_x + \sigma_x) = (0{,}5 - 0{,}2887; 0{,}5 + 0{,}2887) = (0{,}2113; 0{,}7887), \qquad (1)$$

Оскільки $0{,}7887 - 0{,}2113 = 0{,}5774$, то в інтервал (1) мають потрапляти близько 57,7 % усіх випадкових чисел.

Щоб реалізувати частотний тест за заданими умовами, створюємо функцію, яка обчислюватиме відношення кількості чисел, що потрапили в інтервал $(a; b)$, до загальної кількості чисел у певній послідовності *seq* (лістинг 2.3).

Лістинг 2.3: Функція для обчислення частоти входження чисел в інтервал $(a; b)$

```
def calc_freq(seq, a, b):
    cnt = 0
    for x in seq:
        if a < x < b:
            cnt += 1

    return cnt / len(seq)
```

2.2.2. Критерій Пірсона

Припустимо, що є послідовність випадкових чисел $X = \{x_1, \dots, x_n\}$. Послідовність X має довжину $|X| = n$. Кожне число послідовності X знаходиться в інтервалі від a до b, тобто $\forall x_i \in X$, $x_i \in [a; b]$. Розділимо інтервал $[a; b]$ на k приблизно рівних інтервалів $I_1, I_2, \dots, I_k$. Тоді різниця значень між інтервалами $z = \frac{(b-a)}{k}$. Отже, отримані інтервали можна представити так:

$$I_1 = [a + z(1-1); a + z \cdot 1)\,,$$
$$I_2 = [a + z(2-1); a + z \cdot 2)\,,$$
$$I_3 = [a + z(3-1); a + z \cdot 3)\,,$$

$$I_4 = [a + z(4-1); a + z \cdot 4)\,,$$
$$\ldots$$
$$I_j = [a + z(j-1); a + z \cdot j)\,,$$
$$\ldots$$
$$I_k = [a + z(k-1); a + z \cdot k]\,.$$

Наприклад, для інтервалу [0,1; 1,1], де кількість підінтервалів $k = 5$, матимемо $z = 1{,}1 - 0{,}1 = 1$. Тоді інтервали $I_1, \ldots, I_5$ будуть такими:

$$I_1 = [0{,}1; 0{,}3)\,, \quad I_2 = [0{,}3; 0{,}5)\,, \quad I_3 = [0{,}5; 0{,}7)\,, \quad I_4 = [0{,}7; 0{,}9)\,, \quad I_5 = [0{,}9; 1{,}1]\,.$$

Отже, позначимо кількість чисел з послідовності X, які знаходяться в j-му інтервалі, як N_j. Нагадаємо, що n — кількість чисел у послідовності X, а k — кількість підінтервалів $I_1, \ldots, I_k$, на які розбивається область значень чисел $[a; b]$ з послідовності X. Тоді значення критерію Пірсона X^2 для випадкової послідовності X обчислюється так:

$$\mathrm{X}^2 = \frac{k}{n} \sum_{j=1}^{k} \left(N_j - \frac{n}{k}\right).$$

Щоб реалізувати обчислення критерію Пірсона для заданої послідовності *seq*, була створена відповідна функція (лістинг 2.4).

Лістинг 2.4: Функція для обчислення значення критерію Пірсона для послідовності *seq*

```
def calc_chi_squared_pearson(seq):
    m = BINS
    N = len(seq)

    freqs = np.histogram(seq, bins = m)[0] # bin data in m bins

    return m/N * sum([pow( x - N/m, 2) for x in freqs])
```

2.3. Обчислення статистичних параметрів

ЗАВДАННЯ Для отриманої послідовності псевдовипадкових чисел обчислити значення статистичних параметрів: математичне сподівання, дисперсія, середньоквадратичне відхилення.

Для перевірки рівномірності розподілу чисел у послідовності нас цікавлять її статистичні параметри, а саме: математичне очікування μ_x, дисперсія σ_x^2 і середньоквадратичне відхилення σ_x. Математичне сподівання послідовності μ_x

обчислюється так:

$$\mu_x = \frac{1}{n}\sum_{i=1}^{n} x_i.$$

Для рівномірного випадкового розподілу очікуване значення математичного сподівання $\mu_x \approx 0{,}5$. Наступним статистичним параметром є дисперсія послідовності σ_x^2. Вона обчислюється так:

$$\sigma_x^2 = \frac{1}{n}\sum_{i=1}^{n} (x_i - m_x)^2 .$$

Для рівномірного випадкового розподілу очікуване значення дисперсії $\sigma_x^2 \approx 0{,}0833$. Останнім цікавлячим нас параметром є середньоквадратичне відхилення послідовності σ_x. Воно обчислюється так:

$$\sigma_x = \sqrt{\sigma_x^2}.$$

Для рівномірного випадкового розподілу очікуване значення середньоквадратичного відхилення $\sigma_x \approx 0{,}2887$.

Для обчислення кожного з вищенаведених статистичних параметрів заданої послідовності *seq* були розроблені відповідні функції, а також функція, яка обчислює і повертає всі параметри разом, у вигляді кортежу (лістинг 2.5).

Лістинг 2.5: Функції для обчислення статистичних параметрів послідовності *seq*

```
def calc_mean(seq):
    return sum(seq) / len(seq)

def calc_variance(seq):
    if len(seq) == 1:
        return 0
    mean = calc_mean(seq)
    variance = sum([pow(x - mean, 2) for x in seq]) / len(seq)
    return variance

def calc_stdev(seq):
    return math.sqrt(calc_variance(seq))

def calc_stat_params(seq):
    return calc_mean(seq), calc_variance(seq), calc_stdev(seq)
```

2.4. *Отримання послідовності псевдовипадкових чисел на довільному інтервалі*

ЗАВДАННЯ Побудувати імітаційну модель, яка відображає роботу генератора псевдовипадкових чисел. Отримати послідовність псевдовипадкових чисел на інтервалі $(a; b)$.

Для отримання послідовностей псевдовипадкових чисел на інтервалі $(a; b)$ використовується функція `random.randrange(a, b)`, яка надається модулем `random` стандартної бібліотеки мови програмування Python і повертає випадкове число з інтервалу $(a; b)$.

2.5. *Розробка реалізації*

Описавши необхідні моделі для виконання завдань, впроваджуємо створені функції та розроблюємо кінцеву реалізацію для виконання завдань (лістинг А.1). Запускаємо створену реалізацію та спостерігаємо результат (рис. 1).

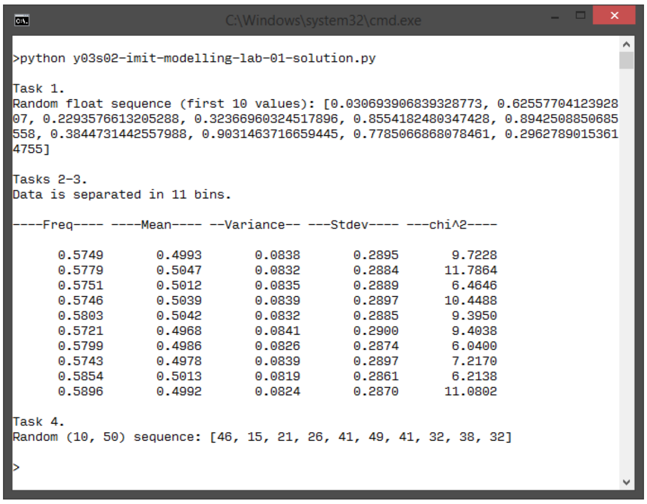

Рис. 1: Результат роботи програмної реалізації

Оцінимо якість розробленого генератора та згенерованих послідовностей. Як бачимо з результату, було згенеровано 10 послідовностей. Отримані значення частотного тесту послідовностей близькі до цільових значень еталонного генератора псевдовипадкових чисел (розділ 2.2.1).

Розглянемо значення критерію Пірсона. Області значень випадкових чисел кожної послідовності були розділені на $k = 11$ інтервалів, тому значення ступенів свободи $v = k - 1 = 10$. Порівнюємо експериментальні значення з теоретичними (табл. 1) і бачимо, що отримані p-значення близькі до 50 %, що є показником якісного генератора.

Табл. 1: Деякі значення χ^2-розподілу для ступенів свободи $\upsilon = 10$

Ступені свободи υ	$p = 5\,\%$	$p = 25\,\%$	$p = 50\,\%$	$p = 75\,\%$	$p = 95\,\%$
10	3,940	6,737	9,342	12,550	18,310

Статистичні параметри згенерованих послідовностей також близькі до параметрів еталонного генератора (розділ 2.3). Отже, можемо зробити висновок, що розроблена реалізація є високоякісним лінійним конгруентним генератором псевдовипадкових чисел.

3. Висновок

Виконуючи дану лабораторну роботу ми ознайомились з еталоном функціонування генератора псевдовипадкових чисел; побудували імітаційну модель функціонування генератора псевдовипадкових чисел на основі лінійного конгруентного методу та здійснили перевірку якості роботи створеного генератора псевдовипадкових чисел.

А. Повний початковий код програмної реалізації

Лістинг А.1: Повний початковий код програмної реалізації

```python
#!/usr/bin/env python3

import random # random.SystemRandom() for generating seed
import math # math.sqrt
import numpy as np

BINS = 11
RANDMIN = 10
RANDMAX = 50

# A class implementing an LCG and relevant methods
class LCG:
    rand_seq = []

    # generator
    def lcg(self, m, a, c, seed):
        while True:
            seed = (a * seed + c) % m
            yield seed

```

```
    def __init__(self,
                 m = 0x7FFFFFFF, # modulus M (as per MINSTD), Mersenne 31, prime
                 a = 48271, # multiplier a, prime number
                 c = 0, # increment c
                 seed = random.SystemRandom().randint(0, 0xFFFFFFFF)):
        self.m, self.a, self.c, self.seed = m, a, c, seed
        self.rand_seq = self.lcg(self.m, self.a, self.c, self.seed)

    def randint(self):
        return next(self.rand_seq)

    def randfloat(self):
        return next(self.rand_seq) / self.m

def calc_freq(seq, a, b):
    cnt = 0
    for x in seq:
        if a < x < b:
            cnt += 1

    return cnt / len(seq)

def calc_mean(seq):
    return sum(seq) / len(seq)

def calc_variance(seq):
    if len(seq) == 1:
        return 0

    mean = calc_mean(seq)
    variance = sum([pow(x - mean, 2) for x in seq]) / len(seq)
    return variance

def calc_stdev(seq):
    return math.sqrt(calc_variance(seq))

def calc_stat_params(seq):
    return calc_mean(seq), calc_variance(seq), calc_stdev(seq)

def calc_chi_squared_pearson(seq):
    m = BINS
    N = len(seq)

    freqs = np.histogram(seq, bins = m)[0] # bin data in m bins

    return m/N * sum([pow( x - N/m, 2) for x in freqs])

```

```python
def calc_seq_props(seq):
    freq = calc_freq(seq, 0.2113, 0.7887)
    mean = calc_mean(seq)
    variance = calc_variance(seq)
    stdev = calc_stdev(seq)
    chi_squared = calc_chi_squared_pearson(seq)

    return freq, mean, variance, stdev, chi_squared

def calc_all_seq_props(dataset):
    res = []
    for seq in dataset:
        res.append(calc_seq_props(seq))

    return res

def print_res(p):
    print('\n{:-^12} {:-^12} {:-^12} {:-^12} {:-^12}'.format('Freq', 'Mean', 'Variance', 'Stdev', 'chi^2'))
    print()
    for s in p:
        print('{:>12.4f} {:>12.4f} {:>12.4f} {:>12.4f} {:>12.4f}'.format(*s))

def build_rand_range(a, b, count = 10):
    res = []
    for i in range(count):
        res.append(random.randrange(a, b))

    return res


def main():
    lcg = LCG() # instantiate an LCG

    a, b = RANDMIN, RANDMAX # randint bounds

    rand_float_seqs = []
    rand_int_seq = build_rand_range(a, b)

    # build test sequence
    for i in range(10):
        seq = [lcg.randfloat() for x in range(10000)]
        rand_float_seqs.append(seq)

    print('\nTask 1.\nRandom float sequence (first 10 values): {}'.format(rand_float_seqs[0][:10]))

```

```python
    print('\nTasks 2-3.\nData is separated in {} bins.'.format(BINS))

    properties = calc_all_seq_props(rand_float_seqs)
    print_res(properties)

    print('\nTask 4.\nRandom ({}, {}) sequence: {}'.format(a, b,
     ↪  rand_int_seq))


if __name__ == '__main__':
    main()
```