

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:

Модуль	m ,	$0 < m$.
Множник	a ,	$0 \leq a < m$.
Інкремент	c ,	$0 \leq c < m$.
Початкове значення	X_0 ,	$0 \leq c < m$.

Сам генератор описується таким рекурентним співвідношенням:

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

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

1. Модуль $m = 2^{31} - 1 = 7\text{FFFFFFF}_{16}$, 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

```
1  # A class implementing an LCG and relevant methods
2  class LCG:
3      rand_seq = []
4
5      # generator
6      def lcg(self, m, a, c, seed):
7          while True:
8              seed = (a * seed + c) % m
9              yield seed
10
11     def __init__(self,
12                 m = 0x7FFFFFFF, # modulus M (as per MINSTD), Mersenne 31,
13                 # prime
14                 a = 48271, # multiplier a, prime number
15                 c = 0, # increment c
16                 seed = random.SystemRandom().randint(0, 0xFFFFFFFF)):
17         self.m, self.a, self.c, self.seed = m, a, c, seed
18         self.rand_seq = self.lcg(self.m, self.a, self.c, self.seed)
19
20     def randint(self):
21         return next(self.rand_seq)
22
23     def randfloat(self):
24         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

```
1  lcg = LCG()
2  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:

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

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 $(a; b)$

```
1 def calc_freq(seq, a, b):
2     cnt = 0
3     for x in seq:
4         if a < x < b:
5             cnt += 1
6
7     return cnt / len(seq)
```

2.2.2. Критерій Пірсона

Припустимо, що є послідовність випадкових чисел $X = \{x_1, \dots, x_n\}$. Довжина послідовності $|X| = n$. Розділимо цю послідовність на k приблизно рівних інтервалів і позначимо кількість чисел в j -му інтервалі як N_j . Тоді значення критерію Пірсона X^2 для випадкової послідовності X обчислюється так:

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

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*

```
1 def calc_chi_squared_pearson(seq):
2     m = BINS
3     N = len(seq)
4
5     freqs = np.histogram(seq, bins = m)[0] # bin data in m bins
6
7     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*

```
1 def calc_mean(seq):
2     return sum(seq) / len(seq)
3
4 def calc_variance(seq):
5     if len(seq) == 1:
6         return 0
7     mean = calc_mean(seq)
8     variance = sum([pow(x - mean, 2) for x in seq]) / len(seq)
9     return variance
10
11 def calc_stdev(seq):
12     return math.sqrt(calc_variance(seq))
13
14 def calc_stat_params(seq):
15     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).

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. Кожна послідовність була розділена 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 %.

```

C:\Windows\system32\cmd.exe

>python y03s02-imit-modelling-lab-01-solution.py

Task 1.
Random float sequence (first 10 values): [0.030693906839328773, 0.62557704123928
97, 0.2293576613205288, 0.32366960324517896, 0.8554182480347428, 0.8942508850685
558, 0.3844731442557988, 0.9031463716659445, 0.7785066868078461, 0.2962789015361
4755]

Tasks 2-3.
Data is separated in 11 bins.

----Freq---- --Mean---- --Variance-- --Stdev---- --chi^2----
0.5749      0.4993      0.0838      0.2895      9.7228
0.5779      0.5047      0.0832      0.2884      11.7884
0.5751      0.5012      0.0835      0.2889      6.4646
0.5746      0.5039      0.0839      0.2897      10.4488
0.5803      0.5042      0.0832      0.2885      9.3950
0.5721      0.4968      0.0841      0.2900      9.4838
0.5799      0.4986      0.0826      0.2874      6.9400
0.5743      0.4978      0.0839      0.2897      7.2170
0.5854      0.5013      0.0819      0.2861      6.2138
0.5896      0.4992      0.0824      0.2870      11.0802

Task 4.
Random (10, 50) sequence: [46, 15, 21, 26, 41, 49, 41, 32, 38, 32]

>

```

Рис. 1: Результат роботи програмної реалізації

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

Ступені свободи ν	$p = 5\%$	$p = 25\%$	$p = 50\%$	$p = 75\%$	$p = 95\%$
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.

A. ПОВНИЙ ПОЧАТКОВИЙ КОД ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Лістинг A.1: Повний початковий код програмної реалізації

```
1  #!/usr/bin/env python3
2
3  import random # random.SystemRandom() for generating seed
4  import math # math.sqrt
5  import numpy as np
6
7  BINS = 11
8  RANDMIN = 10
9  RANDMAX = 50
10
11 # A class implementing an LCG and relevant methods
12 class LCG:
13     rand_seq = []
14
15     # generator
16     def lcg(self, m, a, c, seed):
17         while True:
18             seed = (a * seed + c) % m
19             yield seed
20
21     def __init__(self,
22                 m = 0x7FFFFFFF, # modulus M (as per MINSTD), Mersenne 31,
23                 ↪ prime
24                 a = 48271, # multiplier a, prime number
25                 c = 0, # increment c
26                 seed = random.SystemRandom().randint(0, 0xFFFFFFFF)):
27         self.m, self.a, self.c, self.seed = m, a, c, seed
28         self.rand_seq = self.lcg(self.m, self.a, self.c, self.seed)
29
30     def randint(self):
31         return next(self.rand_seq)
32
33     def randfloat(self):
34         return next(self.rand_seq) / self.m
35
36     def calc_freq(seq, a, b):
37         cnt = 0
38         for x in seq:
39             if a < x < b:
40                 cnt += 1
41
42         return cnt / len(seq)
43
44     def calc_mean(seq):
```



```

44     return sum(seq) / len(seq)
45
46 def calc_variance(seq):
47     if len(seq) == 1:
48         return 0
49
50     mean = calc_mean(seq)
51     variance = sum([pow(x - mean, 2) for x in seq]) / len(seq)
52     return variance
53
54 def calc_stdev(seq):
55     return math.sqrt(calc_variance(seq))
56
57 def calc_stat_params(seq):
58     return calc_mean(seq), calc_variance(seq), calc_stdev(seq)
59
60 def calc_chi_squared_pearson(seq):
61     m = BINS
62     N = len(seq)
63
64     freqs = np.histogram(seq, bins = m)[0] # bin data in m bins
65
66     return m/N * sum([pow(x - N/m, 2) for x in freqs])
67
68 def calc_seq_props(seq):
69     freq = calc_freq(seq, 0.2113, 0.7887)
70     mean = calc_mean(seq)
71     variance = calc_variance(seq)
72     stdev = calc_stdev(seq)
73     chi_squared = calc_chi_squared_pearson(seq)
74
75     return freq, mean, variance, stdev, chi_squared
76
77 def calc_all_seq_props(dataset):
78     res = []
79     for seq in dataset:
80         res.append(calc_seq_props(seq))
81
82     return res
83
84 def print_res(p):
85     print('\n{:^-12} {:^-12} {:^-12} {:^-12} {:^-12}'.format('Freq',
86         ↪ 'Mean', 'Variance', 'Stdev', 'chi^2'))
87     print()
88     for s in p:
89         print('{:>12.4f} {:>12.4f} {:>12.4f} {:>12.4f}'
90             ↪ '{:>12.4f}'.format(*s))

```

```

90 def build_rand_range(a, b, count = 10):
91     res = []
92     for i in range(count):
93         res.append(random.randrange(a, b))
94
95     return res
96
97
98 def main():
99     lcg = LCG() # instantiate an LCG
100
101     a, b = RANDMIN, RANDMAX # randint bounds
102
103     rand_float_seqs = []
104     rand_int_seq = build_rand_range(a, b)
105
106     # build test sequence
107     for i in range(10):
108         seq = [lcg.randfloat() for x in range(10000)]
109         rand_float_seqs.append(seq)
110
111     print('Task 1. Random float sequence (first 10 values):
112         ↪ {}'.format(rand_float_seqs[0][:10]))
113
114     print('Tasks 2-3. Data is separated in {} bins.'.format(BINS))
115
116     properties = calc_all_seq_props(rand_float_seqs)
117     print_res(properties)
118
119     print('Task 4. Random ({} , {}) sequence: {}'.format(a, b,
120         ↪ rand_int_seq))
121
122 if __name__ == '__main__':
123     main()

```
