Министерство науки и высшего образования РФ

ФГАОУ ВО Пермский национальный исследовательский

политехнический университет

Кафедра «Вычислительная математика, механика и биомеханика»

Отчет по лабораторной работе №3

по дисциплине «Моделирование процессов и систем»

по теме: «Генератор случайных чисел»

Выполнил: студент группу ИСТ-22-1б Кожин В. А.

Проверил: доцент каф. ВММБ Максимов Петр Викторович

Пермь, 2024

**Содержание:**

[Введение 3](#_Toc179202707)

[Основная часть 4](#_Toc179202708)

[Заключение 10](#_Toc179202709)

[Приложение 11](#_Toc179202710)

# Введение

**Цель работы:** реализовать генератор псевдослучайных чисел на основе линейного конгруэнтного метода. Провести анализ мощности генератора.

**Задание к лабораторной работе:**

1. Реализовать алгоритм генерации псевдослучайных чисел на основе линейного конгруэнтного метода.
2. Подобрать параметры генератора так, чтобы он был близок по характеристикам к эталонному генератору.
3. Определить длину периода ГСПЧ (количество чисел до первого повторения).
4. Сравнить результаты написанного генератора псевдослучайных чисел с результатами встроенного генератора.

**Рассматриваемый метод:**

Линейный конгруэнтный метод был предложен Д.Г.Лемером на проходившем в 1949 году симпозиуме и опубликован в 1951 году в трудах симпозиума. Суть метода заключается в вычислении последовательности случайных чисел , пологая что , где  – модуль (натуральное число, относительно которого вычисляет остаток от деления; ),  – множитель (),  – приращение (),  – начальное значение ().

# Основная часть

1. Реализовать алгоритм генерации псевдослучайных чисел на основе линейного конгруэнтного метода.

Алгоритм реализован на языке программирования Python и продемонстрирован на рисунке №1, где seed – начальное значение , factor – множитель , offset – приращение  и modulus – модуль :

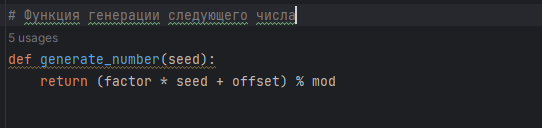


Рис. №1. Реализации линейного конгруэнтного метода.

1. Подобрать параметры генератора так, чтобы он был близок по характеристикам к эталонному генератору.

Действующие параметры продемонстрированы на рисунке №2:

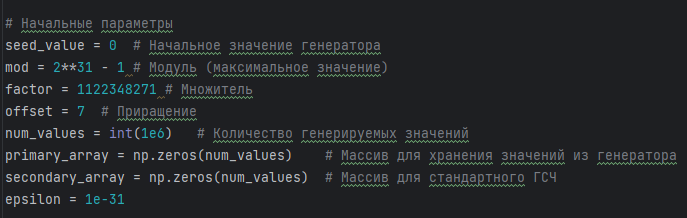


Рис. №2. Параметры для генератора

1. Определить длину периода ГСПЧ (количество чисел до первого повторения).

При текущих параметрах методом Флойда черепаха – заяц, в котором черепаха делает шаг, а заяц по два шага, и так будет продолжаться пока их значения не будут совпадать, после чего черепаха будет шагать до зайца, считая период. Алгоритм продемонстрирован на рисунке №3:

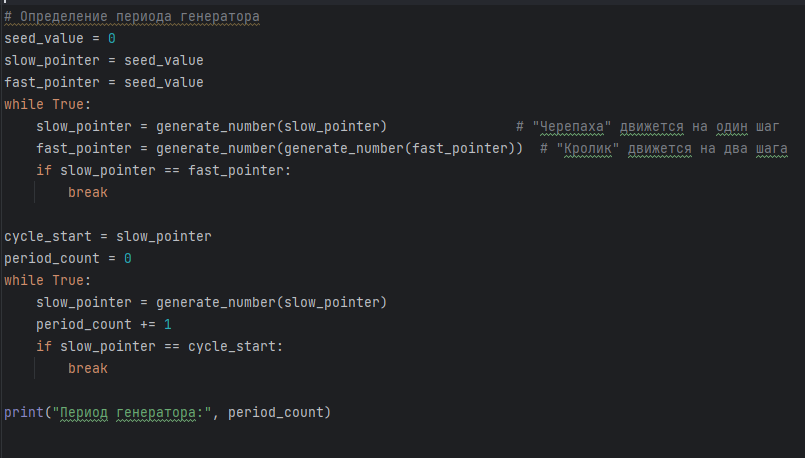


Рис. №3. Метод Флойда “черепаха – заяц”

Результат вычислений данного алгоритма показал, что период ГСЧ равен 238609294

1. Сравнить результаты написанного генератора псевдослучайных чисел с результатами встроенного генератора.

Результаты показали, что ГСЧ выдает практически подобные результаты что и встроенный генератор чисел. Время прохождения требований, описанных ниже, и генерации чисел, незначительно отличается, возможно, такой результат связан с небольшим количеством элементов. Пример работы продемонстрирован на рисунке №4:

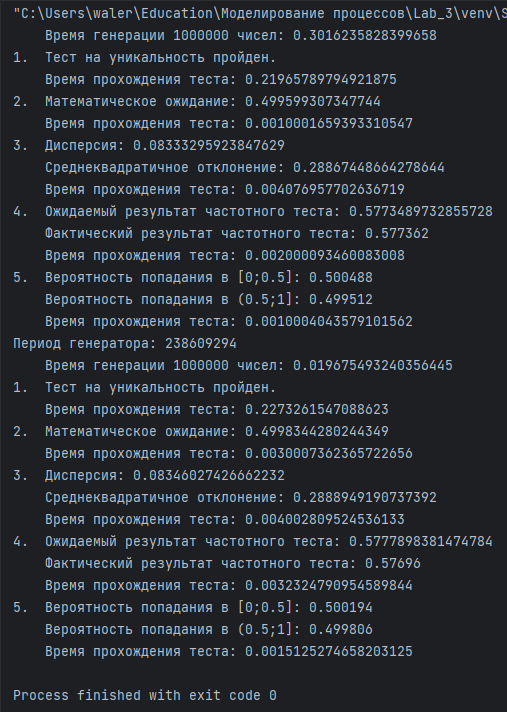


Рис. №4. Результаты работы программы

Далее продемонстрированы требования и алгоритмы позволяющие проверить систему:

1. Должен уметь генерировать не менее 1 000 000 неповторяющихся случайных чисел в интервале от 0 до 1.

Проверка данного условия продемонстрирована на рисунке №5, в данном алгоритме последовательно происходит сортировка полученного массива, состоящего из 1000000 сгенерированных чисел, и проверка на уникальность с помощью цикла for и сравнение элемента [i] и элемента [i+1], такое решение позволяет сэкономить время за счёт ресурсов памяти, входными данными является массив:

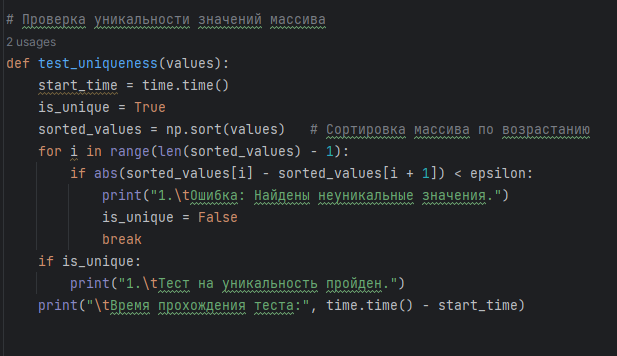


Рис. №5. Тест на уникальность

1. Математическое ожидание, посчитанное для всех сгенерированных чисел, должно быть близко к 0.5.

Проверка данного условия продемонстрирована на рисунке №6, в данном алгоритме математическое ожидание считается за счёт частного от суммы всех элементов и количества всех элементов, входными данными является сам массив:

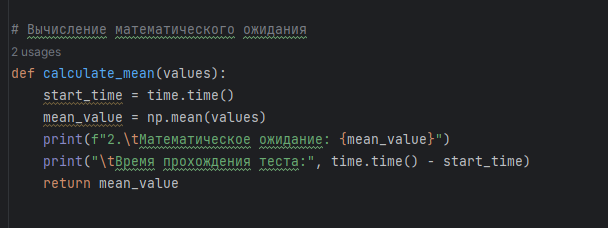


Рис. №6. Математическое ожидание

1. Дисперсия, посчитанная для всех сгенерированных чисел, должна быть близка к 0.0833 (среднеквадратическое отклонение 0.2887).

Проверка данного условия продемонстрирована на рисунке №7, в данном алгоритме дисперсия считается за счёт частного суммы всех квадратов отклонения элементов массива от математического ожидания и количество элементов массива, среднеквадратичное отклонение считается, как квадратный корень из дисперсии, на вход поступают массив и математическое ожидание:

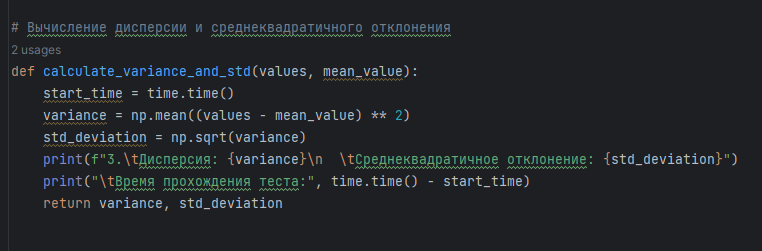


Рис. №7 Дисперсия и среднеквадратичное отклонение

1. При прохождении частотного теста в заданный интервал должно попадать около 57.7% чисел.

Проверка данного условия продемонстрирована на рисунке №8, в данном алгоритме установлены верхняя и нижняя граница с помощью среднеквадратичного отклонения, после чего каждый элемент проверяется на попадание в этот диапазон:

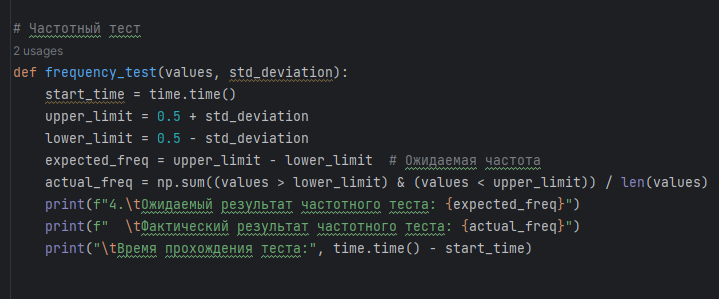


Рис. №8. Частотный тест

1. Вероятности попадания чисел в интервалы [от 0 до 0.5] и [от 0.5 до 1] должны быть около 0.5.

Проверка данного условия продемонстрирована на рисунке №9, в данном алгоритме происходит проверка array[i] <= 0,5 и в соответствии с ней производится подсчёт количества элементов в каждом диапазоне:

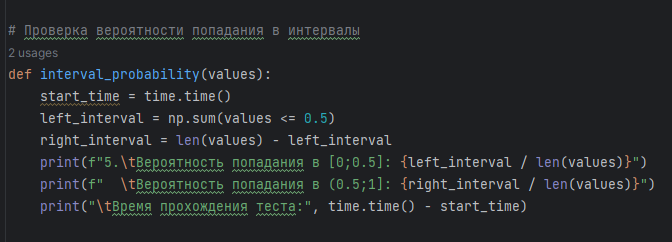


Рис. №9. Вероятность попадания

Тестирование работы программы и проверка результатов решения продемонстрировала правильность написанного кода программы.

# Заключение

В лабораторной работе были изучен принцип работы генератора псевдослучайных чисел на основе линейного конгруэнтного метода. Описана работа алгоритма. Написана компьютерная программа, наглядно показывающую работу генератора и удовлетворяющая всем поставленным требованиям. Проведено сравнение со встроенным генератором языка Python. Все поставленные задачи выполнены.

Программа включает в себя:

* Алгоритм генерации случайных чисел;
* Проверка качества генератора.
* Показатели для сравнения со встроенным генераторам.

# Приложение

Код программы на языке Python:

import numpy as np  
import time  
  
# Начальные параметры  
seed\_value = 0 # Начальное значение генератора  
mod = 2\*\*31 - 1 # Модуль (максимальное значение)  
factor = 1122348271 # Множитель  
offset = 7 # Приращение  
num\_values = int(1e6) # Количество генерируемых значений  
primary\_array = np.zeros(num\_values) # Массив для хранения значений из генератора  
secondary\_array = np.zeros(num\_values) # Массив для стандартного ГСЧ  
epsilon = 1e-31  
  
# Функция генерации следующего числа  
def generate\_number(seed):  
 return (factor \* seed + offset) % mod  
  
  
# Проверка уникальности значений массива  
def test\_uniqueness(values):  
 start\_time = time.time()  
 is\_unique = True  
 sorted\_values = np.sort(values) # Сортировка массива по возрастанию  
 for i in range(len(sorted\_values) - 1):  
 if abs(sorted\_values[i] - sorted\_values[i + 1]) < epsilon:  
 print("1.\tОшибка: Найдены неуникальные значения.")  
 is\_unique = False  
 break  
 if is\_unique:  
 print("1.\tТест на уникальность пройден.")  
 print("\tВремя прохождения теста:", time.time() - start\_time)  
  
  
# Вычисление математического ожидания  
def calculate\_mean(values):  
 start\_time = time.time()  
 mean\_value = np.mean(values)  
 print(f"2.\tМатематическое ожидание: {mean\_value}")  
 print("\tВремя прохождения теста:", time.time() - start\_time)  
 return mean\_value  
  
  
# Вычисление дисперсии и среднеквадратичного отклонения  
def calculate\_variance\_and\_std(values, mean\_value):  
 start\_time = time.time()  
 variance = np.mean((values - mean\_value) \*\* 2)  
 std\_deviation = np.sqrt(variance)  
 print(f"3.\tДисперсия: {variance}\n \tСреднеквадратичное отклонение: {std\_deviation}")  
 print("\tВремя прохождения теста:", time.time() - start\_time)  
 return variance, std\_deviation  
  
  
# Частотный тест  
def frequency\_test(values, std\_deviation):  
 start\_time = time.time()  
 upper\_limit = 0.5 + std\_deviation  
 lower\_limit = 0.5 - std\_deviation  
 expected\_freq = upper\_limit - lower\_limit # Ожидаемая частота  
 actual\_freq = np.sum((values > lower\_limit) & (values < upper\_limit)) / len(values)  
 print(f"4.\tОжидаемый результат частотного теста: {expected\_freq}")  
 print(f" \tФактический результат частотного теста: {actual\_freq}")  
 print("\tВремя прохождения теста:", time.time() - start\_time)  
  
  
# Проверка вероятности попадания в интервалы  
def interval\_probability(values):  
 start\_time = time.time()  
 left\_interval = np.sum(values <= 0.5)  
 right\_interval = len(values) - left\_interval  
 print(f"5.\tВероятность попадания в [0;0.5]: {left\_interval / len(values)}")  
 print(f" \tВероятность попадания в (0.5;1]: {right\_interval / len(values)}")  
 print("\tВремя прохождения теста:", time.time() - start\_time)  
  
# Генерация случайных чисел с помощью ЛКМ  
start\_time = time.time()  
for i in range(num\_values):  
 seed\_value = generate\_number(seed\_value)  
 primary\_array[i] = seed\_value / mod # Преобразование значений в диапазон [0, 1]  
print(f"\tВремя генерации {num\_values} чисел: {time.time() - start\_time}")  
  
# Тесты для ЛКМ  
test\_uniqueness(primary\_array)  
mean\_value = calculate\_mean(primary\_array)  
variance, std\_deviation = calculate\_variance\_and\_std(primary\_array, mean\_value)  
frequency\_test(primary\_array, std\_deviation)  
interval\_probability(primary\_array)  
  
  
  
# Определение периода генератора  
seed\_value = 0  
slow\_pointer = seed\_value  
fast\_pointer = seed\_value  
while True:  
 slow\_pointer = generate\_number(slow\_pointer) # "Черепаха" движется на один шаг  
 fast\_pointer = generate\_number(generate\_number(fast\_pointer)) # "Кролик" движется на два шага  
 if slow\_pointer == fast\_pointer:  
 break  
  
cycle\_start = slow\_pointer  
period\_count = 0  
while True:  
 slow\_pointer = generate\_number(slow\_pointer)  
 period\_count += 1  
 if slow\_pointer == cycle\_start:  
 break  
  
print("Период генератора:", period\_count)  
  
  
# Генерация случайных чисел с помощью стандартного генератора NumPy  
start\_time = time.time()  
secondary\_array = np.random.rand(num\_values)  
print(f"\tВремя генерации {num\_values} чисел: {time.time() - start\_time}")  
  
# Тесты для стандартного генератора  
test\_uniqueness(secondary\_array)  
comp\_mean\_value = calculate\_mean(secondary\_array)  
comp\_variance, comp\_std\_deviation = calculate\_variance\_and\_std(secondary\_array, comp\_mean\_value)  
frequency\_test(secondary\_array, comp\_std\_deviation)  
interval\_probability(secondary\_array)