Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе №3

Название Случайные числа

Дисциплина Моделирование

Студент Зайцева А. А.

Группа ИУ7-72Б

Оценка (баллы) _____

Преподаватель Рудаков И. В.

Москва — 2022 г.

1 Задание

Написать программу, которая генерирует алгоритмическим способом и получает табличным способом случайные последовательности одноразрядных, двухразрядных и трехразрядных целых чисел.

Разработать количественный критерий оценки случайности последовательности чисел. Для каждой полученной последовательности вычислить и вывести значение критерия.

Предусмотреть возможность ввода последовательности чисел и оценки их случайности с помощью критерия.

2 Теоретические сведения

2.1 Способы получения случайных чисел

На практике наиболее распространены 3 способа получения случайных чисел.

Аппаратный способ

При использовании аппаратного способа случайные числа вырабатываются специальной электронной приставкой (генератором случайных чисел). Реализация данного способа не требует дополнительных вычислений, необходима только одна операция – обращение к вычислительному устройству.

В качестве физического эффекта, лежащего в основе генерации случайных чисел, может использоваться, например, шум в электронных приборах. Для генерации необходимы источник шума, ключевая схема, формирователь импульсов и пересчетная схема.

Табличный способ

Случайные числа берутся из заранее подготовленной таблицы, которая находится во внешней или оперативной памяти. Числа в таблице

проверены на случайность и некоррелированы.

Алгоритмический способ

Алгоритмический способ основан на использовании специальных алгоритмов. К таким алгоритмам, например, относятся следующие:

- алгоритм Фон-Неймана (метод серединных квадратов);
- метод перемешивания (сдвигов);
- линейный конгруэнтный генератор;
- вихрь Мерсенна.

2.2 Критерий оценки случайности последовательности

Для количественного отображения того, насколько «случайной» является последовательность, было решено рассматривать разницы значений соседних элементов последовательности.

Пусть дана последовательность X из n целых чисел из некоторого интервала $[x_{min}; x_{max}]$:

$$\begin{gathered} x = (x_1, ..., x_n), \quad \text{где} \\ x_{min} \leq x_i \leq x_{max}, \forall i \in \{1, ..., n\}. \end{gathered} \tag{1}$$

По ним вычисляется последовательность D разностей соседних элементов:

$$\begin{gathered} D = (d_1, ..., d_k), \quad \text{где} \\ k = \left[\frac{n}{2}\right], \\ d_i = x_{2i+1} - x_{2i}. \end{gathered} \tag{2}$$

Всевозможные разности целых чисел из интервала $[x_{min}; x_{max}]$ можно представить в виде последовательности из $s = 2(x_{max} - x_{min}) - 1$ целых чисел DP:

$$DP = (x_{min} - x_{max} + 1, x_{min} - x_{max} + 2, ..., x_{max} - x_{min}). \tag{3}$$

Вероятность p_i того, что разность соседних элементов равна числу $dp_i \in DP$, можно вычислить по следующей формуле:

$$p_i = P\left(\text{dif} = dp_i\right) = \frac{x_{max} - x_{min} - |dp_i|}{(x_{max} - x_{min})^2} \tag{4}$$

Тогда, подсчитав для каждой теоретически возможной разности $dp_i \in DP$ количество ее вхождений y_i в полученную последовательность D, можно рассчитать статистику χ^2 по следующей формуле:

$$V = \frac{1}{k} \sum_{i=1}^{s} \frac{y_i^2}{p_i} - k. \tag{5}$$

Затем с помощью этого количественного критерия V дается качественная оценка «случайности» последовательности. С помощью таблицы процентных точек распределения χ^2 с количеством степеней свободы $v = x_{max} - x_{min}$, определяется, между какими точками находится вычисленное значение V. Если оно оказывается между 5% и 95% точками, то делается вывод, что числа исходной последовательности X – случайные; если оно оказывается левее 1% точки или правее 99% точки, то последовательность X признается не случайной; в ином случае последовательность X характеризуется как «подозрительная».

2.3 Вихрь Мерсенна для генерации псевдослучайных чисел

Для получения случайных чисел алгоритмическим способом выбран вихрь Мерсенна.

Существуют по меньшей мере два общих варианта алгоритма, различающихся только величиной используемого простого числа Мерсенна. В данной работе будет использован наиболее распространенный из них – алгоритм MT19937.

Алгоритм работы вихря Мерсенна состоит из двух частей: рекурсивной и закалки. Рекурсивная часть представляет собой регистр сдвига с

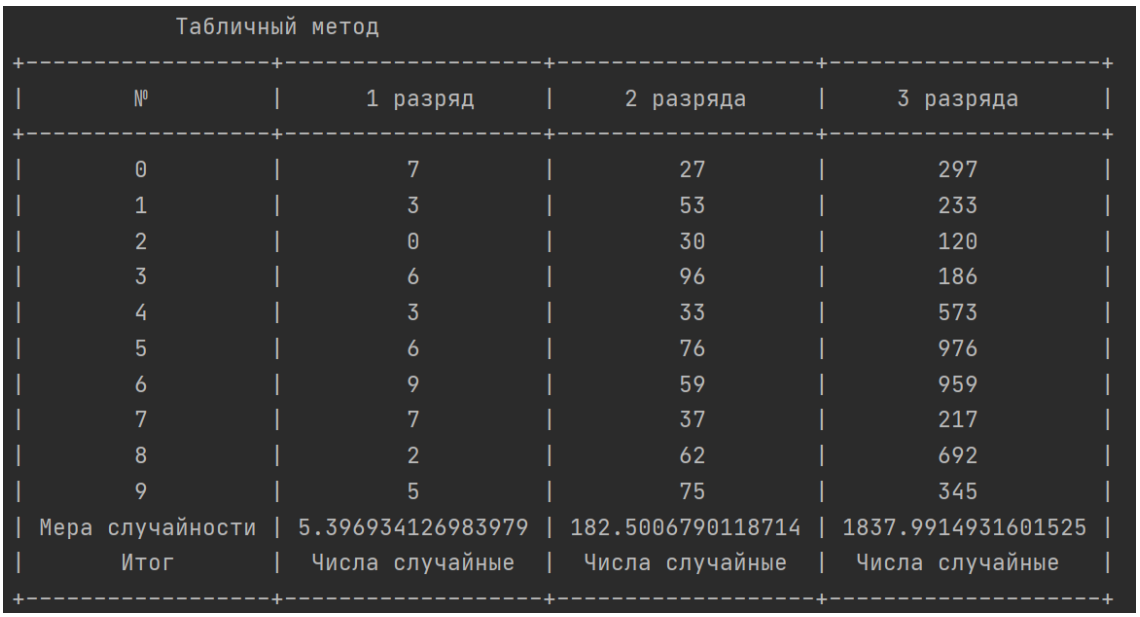
линейной обратной связью, в котором все биты слова определяются рекурсивно.

Регистр сдвига состоит из 624 элементов, 19937 клеток. Первый элемент состоит из 1 бита, а остальные – из 32. Процесс генерации начинается с логического умножения на битовую маску, которая отбрасывает 31 бит (кроме наиболее значащих). Следующим шагом выполняется инициализация (x0, x1,. . . , x623) любыми беззнаковыми 32-разрядными целыми числами. Следующие шаги включают в себя объединение и переходные состояния.

Параметры MT19937 были тщательно подобраны так, что характеристический многочлен примитивный (nw — r равно числу Мерсенна 19937), параметры закалки выбраны так, чтобы можно было получить «хорошее» равномерное распределение; значение последней строки матрицы (последнего элемента) выбирается случайным образом.

3 Результаты работы программы

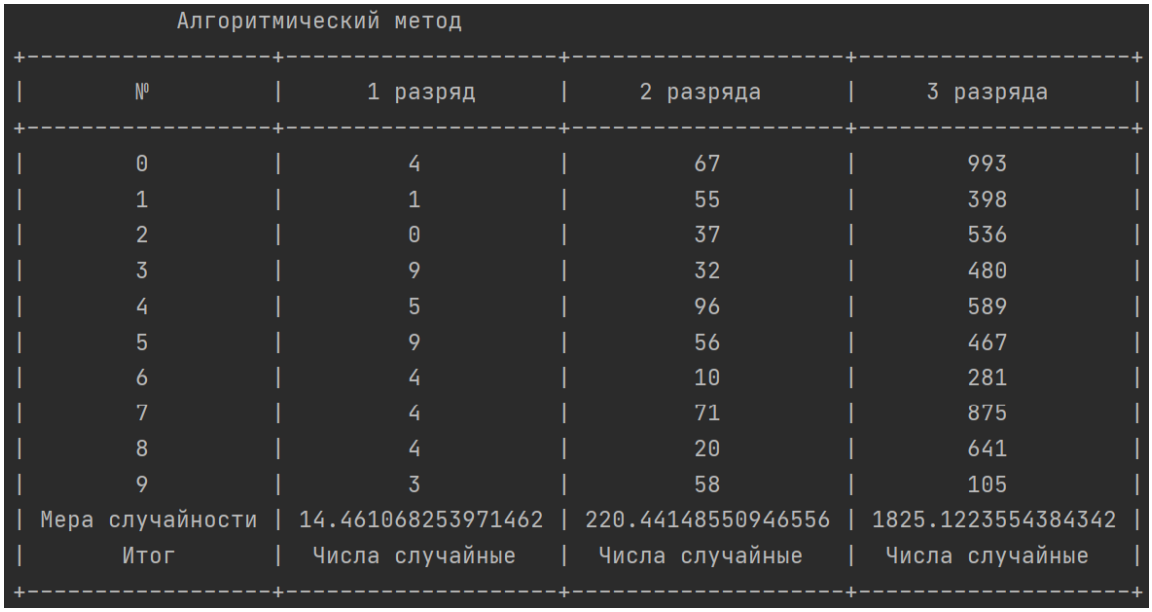
В программе с помощью табличного метода и вихря Мерсенна генерируются последовательности из 100000 случайных одно-, двух- и трехразрядных чисел. Затем каждая последовательность оценивается с помощью количественного критерия «случайности», по которому затем дается качественная оценка. В таблицах выводятся оценки каждой последовательности и 10 ее элементов. Вывод таблиц приведен на рисунках 1 и 2.



Табличный метод

№	1 разряд	2 разряда	3 разряда
0	7	27	297
1	3	53	233
2	0	30	120
3	6	96	186
4	3	33	573
5	6	76	976
6	9	59	959
7	7	37	217
8	2	62	692
9	5	75	345
Мера случайности	5.396934126983979	182.5006790118714	1837.9914931601525
Итог	Числа случайные	Числа случайные	Числа случайные

Рисунок 1 – Результаты расчетов для табличного метода



Алгоритмический метод

№	1 разряд	2 разряда	3 разряда
0	4	67	993
1	1	55	398
2	0	37	536
3	9	32	480
4	5	96	589
5	9	56	467
6	4	10	281
7	4	71	875
8	4	20	641
9	3	58	105
Мера случайности	14.461068253971462	220.44148550946556	1825.1223554384342
Итог	Числа случайные	Числа случайные	Числа случайные

Рисунок 2 – Результаты расчетов для алгоритмического метода

Затем критерий «случайности» тестируется на предельных значениях: когда последовательность монотонно возрастает или убывает, состоит из периодечски повторяющихся элементов. Результаты тестирования приведены на рисунке 3.

```
Тестирование на предельных значениях

Разрядность: 1
Последовательность: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Мера случайности: 50.55555555555556
Итог: Числа не случайные

Разрядность: 1
Последовательность: [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
Мера случайности: 50.55555555555556
Итог: Числа не случайные

Разрядность: 1
Последовательность: [1, 3, 1, 3, 1, 3, 1, 3, 1, 3]
Мера случайности: 57.5
Итог: Числа не случайные

Разрядность: 3
Последовательность: [100, 101, 102, 103, 104, 105, 106, 107, 108, 109]
Мера случайности: 405000.5005561735
Итог: Числа не случайные
```

Рисунок 3 – Результаты тестирования критерия

Затем пользователю предлагается ввести собственную последовательность чисел определенной разрядности для проверки ее случайности. Пример работы приведен на рисунке 4

```
Разрядность вводимой последовательности (1, 2 или 3).
1
Введите последовательность чисел через пробел
3 7 9 2 6 6 7 8 0 1 4 5 7 9 2 3 1 0
Коэффициент:  20.043209876543212
Числа подозрительные
```

Рисунок 4 – Пример обработки пользовательской последовательности

4 Код программы

Код основной программы, котрая инициирует генерацию последовательностей, рассчитывает критерии и выводит результаты, приведен в листинге 1 (используемый язык – Python).

Листинг 1 – Код основной программы

```
from prettytable import PrettyTable
from RandomClass import Random
from math import sqrt

N_RANDOMS = 100000
eps = 1e-6
N_OUTPUT = 10

numbers_limits = {1: [0, 10], 2: [10, 99], 3: [100, 999]}


def hi2_quantiles_for_v(v, ps=(1, 5, 10, 90, 95, 99)):
    return {xp: v + (sqrt(2 * v) * xp) + (2 * (xp ** 2) / 3) - (2 / 3) for
    xp in ps}


hi2_quantiles = {
    # 1 цифра -> возможные числа [0; 9] -> возможно 19 разниц [-9; 9] (10 чисел)
```

```
    1: {1: 2.088, 5: 3.325, 10: 4.158, 90: 14.684, 95: 16.919, 99:
    21.666},
    # 2 цифры возможные-> числа [10; 99] -> возможно 179 разниц [-89; 89] (90
    чисел)
    2: hi2_quantiles_for_v(89),
    # 3 цифры возможные-> числа [100; 999] -> возможно 1799 разниц [-899; 899]
    (900 чисел)
    3: hi2_quantiles_for_v(899),
}

def random_from_table():
    with open('random_ints.txt') as file:
        lines = file.readlines()

    numbers = list()
    for line in lines:
        numbers.extend(list(map(int, line.strip().split())))
    numbers = numbers[:N_RANDOMS]

    one = [number % 10 for number in numbers]
    two = [10 + number % 90 for number in numbers]
    three = [100 + number % 900 for number in numbers]

    return one, two, three


def random_from_alg():
    random = Random(2)
    one = [random.randint(*numbers_limits[1]) for _ in range(N_RANDOMS)]
    two = [random.randint(*numbers_limits[2]) for _ in range(N_RANDOMS)]
    three = [random.randint(*numbers_limits[3]) for _ in range(N_RANDOMS)]
    return one, two, three


def calc_coef(random_numbers, start_random, end_random):
    if start_random != 0:
        end_random += 1
    min_dif = start_random - end_random + 1
    max_dif = end_random - start_random - 1

```

```
    possible_differences = list(range(min_dif, max_dif + 1))
    probabilities_of_differences = {
        difference: (end_random - start_random - abs(difference)) / ((
    end_random - start_random) ** 2)
        for difference in possible_differences
    }

    n_differences = len(random_numbers) // 2
    differences = [random_numbers[2 * i + 1] - random_numbers[2 * i] for i
     in range(n_differences)]
    differences_counts = {difference: differences.count(difference) for
    difference in possible_differences}

    if sum(differences_counts.values()) != len(differences) or \
    abs(sum(probabilities_of_differences.values()) - 1) > eps:
        print(sum(differences_counts.values()))
        print(sum(probabilities_of_differences.values()))
        raise ValueError

    V = 0
    for difference in differences_counts.keys():
        V += (differences_counts[difference] ** 2) /
    probabilities_of_differences[difference]
    V = (V / n_differences) - n_differences
    return V


def analyze_coef(counted_coef, digits_amount):
    if counted_coef < hi2_quantiles[digits_amount][1] or counted_coef >
    hi2_quantiles[digits_amount][99]:
        return "Числа не случайные"
    elif hi2_quantiles[digits_amount][5] < counted_coef < hi2_quantiles[
    digits_amount][95]:
        return "Числа случайные"
    else:
        return "Числа подозрительные"


def check_limit_values():
    tests = [
```

```
    [1, list(range(10))],
    [1, list(range(9, -1, -1))],
    [1, [1, 3, 1, 3, 1, 3, 1, 3, 1, 3]],
    [3, list(range(100, 1000))],
    ]
    print('\n\n\Тестированиеn на предельных значениях')
    for digits, arr in tests:
        print()
        print(f'Разрядность: {digits}')
        print(f'Последовательность: {arr[:min(len(arr), N_OUTPUT)]}')
        coef = calc_coef(arr, *numbers_limits[digits])
        print(f'Мера случайности: {coef}')
        print(f'Итог: {analyze_coef(coef, digits)}')


def main():
    indexes = [i for i in range(N_OUTPUT)]

    for alg, alg_name in [[random_from_table, 'Табличный метод'], [random_from_alg, 'Алгоритмический метод']]:
        res_table = PrettyTable()
        one, two, three = alg()
        res_table.add_column("№", indexes + ['Мера случайности', 'Итог'])

        one_coef = calc_coef(one, *numbers_limits[1])
        two_coef = calc_coef(two, *numbers_limits[2])
        three_coef = calc_coef(three, *numbers_limits[3])
        res_table.add_column('1 разряд', one[:N_OUTPUT] + [one_coef, analyze_coef(one_coef, 1)])
        res_table.add_column('2 разряда', two[:N_OUTPUT] + [two_coef, analyze_coef(two_coef, 2)])
        res_table.add_column('3 разряда', three[:N_OUTPUT] + [three_coef, analyze_coef(three_coef, 3)])

        print(f"\t\t\t{alg_name}")
        print(res_table)

    check_limit_values()

    print("\n\n\n")
```

```
    digits = int(input("Разрядность вводимой последовательности (1, 2 или 3).\n")
    )
    print("Введите последовательность чисел через пробел")
    arr = list(map(int, input().split()))
    coef = calc_coef(arr, *numbers_limits[digits])
    print("Коэффициент: ", coef)
    print(analyze_coef(coef, digits))


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

Класс Random, реализующий алгоритм вихря Мерсенна, приведен в листинге 2.

Листинг 2 – Класс Random, реализующий алгоритм вихря Мерсенна

```

# MT19937, битный32- генератор MT
class Random():
    def __init__(self, c_seed=0):
        # Параметры n и r выбраны так, что характеристический многочлен примитивный
    или
        # (nw - r) число( клеток в регистре сдвига, пространство состояний) равна
    числу Мерсенна 19937
        self.w = 32  # размер слова -- 32 бит
        self.n = 624  # число элементов в регистре сдвига
        self.r = 31  # количество младших бит
        self.m = 397

        # Параметры закалки подобраны так, что мы можем получить хорошее равномерное
    распределение.
        self.l = 18
        self.s = 7
        self.t = 15
        self.u = 11
        self.a = 0x9908B0DF
        self.b = 0x9D2C5680
        self.c = 0xEFC60000

        self.f = 1812433253
```

```
21
22          # Массив для хранения состояний генератора
23          self.MT = [0 for _ in range(self.n)]
24          self.lower_mask = 0x7FFFFFFF  # битовая маска младших r бит,
25          self.upper_mask = 0x80000000  # битовая маска старших w-r бит
26
27          # Инициализация
28          self.index = self.n + 1
29          self.seed(c_seed)
30
31      def seed(self, num):
32          """
33          Начальное заполнение матрицы состояний
34          """
35          self.MT[0] = num
36          for i in range(1, self.n):
37              temp = self.f * (self.MT[i-1] ^ (self.MT[i-1] >> (self.w-2)))
    + i
38              self.MT[i] = temp & 0xffffffff
39
40      def twist(self):
41          """
42          Генерация следующих n значений из последовательности x_i
43          """
44          for i in range(self.n):
45              # Шаг 2. Вычисление (xiu | xi+1l)
46              x = (self.MT[i] & self.upper_mask) + \
47              (self.MT[(i+1) % self.n] & self.lower_mask)
48              # Шаг 3. Вычисление значения следующего элемента последовательности по
49              # рекуррентному выражению
50              xA = x >> 1
51              if (x % 2) != 0:
52                  xA = xA ^ self.a
53              self.MT[i] = self.MT[(i + self.m) % self.n] ^ xA
54          self.index = 0
55
56      def extract_number(self):
57          """
58          Закалка на( основе MT[index])
59          Необработанные последовательности, генерируемые рекурсией, обладают плохим
```

```
    равномерным распределением на больших
60          размерностях. Чтобы это исправить, используется метод закалки англ(.
    tempering), на выходе которого получается
61          итоговая псевдослучайная последовательность. Метод заключается в том, что
    каждое сгенерированное слово
62          умножается справа на специальную обратимую матрицу T размера w × w.
63          Каждые n чисел вызывается twist
64          """
65          if self.index >= self.n:
66              self.twist()
67          # Шаг 4. Вычисление x[i]T
68          y = self.MT[self.index]
69          y = y ^ ((y >> self.u) & 0xFFFFFFFF)
70          y = y ^ ((y << self.s) & self.b)
71          y = y ^ ((y << self.t) & self.c)
72          y = y ^ (y >> self.l)
73
74          self.index += 1
75          return y & 0xffffffff
76
77      def random(self):
78          """ Равномерное распределение [0,1) """
79          return self.extract_number() / 4294967296  # 2^w
80
81      def randint(self, a, b):
82          """ Случайное целое на отрезке [a,b) """
83          n = self.random()
84          return int(n / (1 / (b-a)) + a)
```