МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»**

Кафедра теоретических основ компьютерной безопасности и криптографии

**Лабораторная работа**

**Преобразование ПСЧ к заданному распределению**

ОТЧЕТ ПО ДИСЦИПЛИНЕ

«ТЕОРИЯ ПСЕВДОСЛУЧАЙНЫХ ГЕНЕРАТОРОВ»

студента 4 курса 431 группы

специальности 10.05.01 Компьютерная безопасность

факультета компьютерных наук и информационных технологий

Ухова Александра Андреевича

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Научный руководитель  Ст. преподаватель | \_\_\_\_\_\_\_\_\_\_ | И.И. Слеповичев |
|  | подпись, дата |  |

Саратов 2024

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_gjdgxs)

[2 Критерий хи-квадрат 5](#_30j0zll)

[3 Критерий серий 9](#_1fob9te)

[4 Критерий интервалов 11](#_3znysh7)

[5 Критерий разбиений 14](#_2et92p0)

[6 Критерий перестановок 16](#_tyjcwt)

[7 Критерий монотонности 18](#_3dy6vkm)

[8 Критерий конфликтов 21](#_1t3h5sf)

[9 Тестирование 23](#_4d34og8)

[ПРИЛОЖЕНИЕ А 25](#_2s8eyo1)

[ПРИЛОЖЕНИЕ B 32](#_17dp8vu)

# **Постановка задачи**

1. Сгенерировать псевдослучайную последовательность заданным методом.

2. Исследовать полученную псевдослучайную последовательность на случайность. Исходные данные Исходными данными для лабораторных занятий являются метод генерации псевдослучайных чисел, диапазон генерации случайных чисел, функция распределения, которой должны подчиняться случайные числа, количество генерируемых чисел.

**Задачи**

1) Сгенерировать последовательность из 10000 случайных чисел из диапазона [0,1]. Исходной программой для генерации ПСЧ может быть программа, созданная в рамках практической работы по данному курсу.

2) Протестировать статистические свойства последовательности псевдослучайных чисел:

a) Вычислить математическое ожидание последовательности;

b) Вычислить среднеквадратичное отклонение последовательности;

c) Сравните полученные оценки с заданными в пп. 1 параметрами. Постройте графики зависимостей оценок от объема выборки. Оцените относительные погрешности для какой-либо одной выборки. d) Вычислить значение и дать ответ на вопрос удовлетворяет ли ППСЧ

1. Критерию хи-квадрат;
2. Критерию серий;
3. Критерию интервалов;
4. Критерию разбиений;
5. Критерию перестановок;
6. Критерию монотонности;
7. Критерию конфликтов.

**На входе:** текстовый файл с ПСЧ, обозначения критерия.

**На выходе:** точечные оценки параметров ППСЧ, ответ о соответствии ППСЧ указанному критерию.

# **Критерий хи-квадрат**

Критерий хи-квадрат широко применяется в статистических приложениях для проверки гипотезы о том, что некоторая выборка чисел 𝑋 𝑛 = (𝑥1, 𝑥2, … , 𝑥𝑛 ) подчиняется некоторому закону распределения 𝐹(𝑥).

*Гипотеза H0* заключаете в том, что случайная величина 𝑋 подчиняется закону распределения 𝐹(𝑥).

Для проверки гипотезы рассмотрим выборку, состоящую из n независимых наблюдений над случайной величиной X:

По выборке построим эмпирическое распределение 𝐹 ∗ (𝑥) случайной величины X. Сравнение эмпирического 𝐹 ∗ (𝑥) и теоретического 𝐹(𝑥) (предполагаемого в гипотезе) производится с помощью специально подобранной функции — критерия согласия.

Статистика:

называется распределением хи-квадрат с *k*-1 степенью свободы.

Проверка гипотезы :

В зависимости от значения критерия  , гипотеза  может приниматься либо отвергаться:

, гипотеза выполняется.

– попадает в левый «хвост» распределения, означает что теоретические и практические значения очень близки, и гипотеза принимается.

– попадает в правый «хвост» распределения, - гипотеза отвергается.

Для точного определения соответствия вычисленной характеристики распределению хи-квадрат используют понятие квантилей распределения хи-квадрат.

*Квантиль хи-квадрат (* – это величина , при которой функция распределения хи-квадрат равна заданной вероятности :

где - функция распределения хи-квадрат с *n* степенями свободы, и

В итоге, проверка критерия для некоторой последовательности чисел (или наблюдений величины *Х*) будет состоять из следующих шагов:

1. Выполняем достаточное число независимых наблюдений.
2. Подсчитываем число наблюдений, попавших в каждый из интервалов
3. Подсчитываем статистику
4. Определяем, находится ли вычисленная в доверительном интервале (например, по таблице 5.1).

Для проверки принадлежности заданному распределению случайной величины критерием хи-квадрат, в качестве доверительного интервала можно использовать .

Если вычисленная величина находится в интервале [0; 0,1), – это выглядит неправдоподобно – распределение случайной величины подтверждается наблюдениями слишком точно, что в природе бывает редко (подкинуть 100 раз монету и получить ровно 50 на 50 не легче, чем 95 на 5). Если же величина находится в интервале – это означает, что гипотеза принадлежности значений наблюдаемой величины не подтвердилась.

**Исходный текст программы**

# 

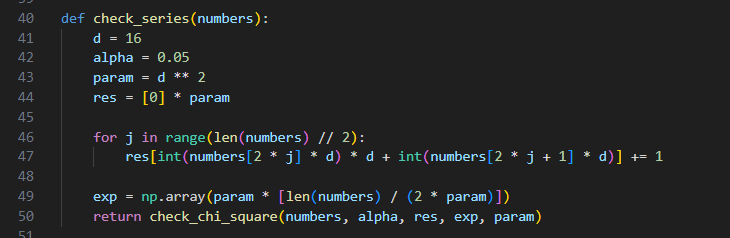
# **Критерий серий**

Критерий серий позволяет убедиться в том, что пары последовательных чисел равномерно распределены независимым образом. Проверка критерия проводится по аналогии с предыдущим случаем, однако, считать будем количество совпадений

Хи-квадрат критерий применяем к полученному набору с параметрами

Критерий можно обобщить на тройки, четверки и т.д. случайных величин. Однако ширина проверяемого диапазона увеличивается пропорционально степени, равной количеству величин в кортеже. Поэтому на практике, при рассмотрении больших серий чисел, используются менее точные критерии.

**Исходный текст программы**

****

# **Критерий интервалов**

Пусть и – два действительных числа таких, что . Рассмотрим длины подпоследовательностей , в которых . Такую последовательность будем называть *интервалом длины* r.

Сначала, нужно подсчитать число интервалов длиной .

**Шаги алгоритма подсчета числа интервалов:**

* 1. Инициализация. Присвоить , .
  2. . Если , то переход на шаг 5.
  3. . Переход к шагу 3.
  4. Если , то , иначе – .
  5. *.* Если то переход на шаг 2.

После этого мы можем применить хи-квадрат критерий для к значениям с параметрами

Здесь – вероятность того, что . Значения *n* и *t* выбираются так, чтобы ожидаемое значение было больше 5. Критерий часто применяют для . В этом случае на шаге 3 алгоритма можно обойтись без сравнения.

Частным случаем применения критерия интервалов является проверка с параметрами

и

Эти случаи называют проверкой «отклонения выше среднего» и проверкой «отклонения ниже среднего».

**Исходный текст программы**

# 

# **Критерий разбиений**

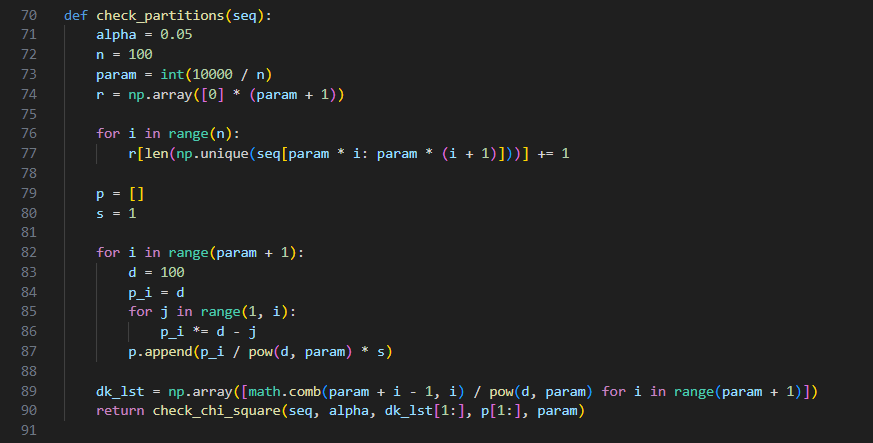
В общем случае критерия разбиений рассматриваются *n* групп *k* последовательных чисел, и подсчитывается число групп из *k* чисел с *r* различными числами. Затем применяется хи-квадрат критерий, в котором используются вероятности того, что в группе *r* различных чисел

Здесь – числа Стирлинга, задающие число способов разбиения множества из *n* элементов на *k* непересекающихся подмножеств, которые можно вычислить по формуле:

Так как вероятности очень малы, когда или 2, следует, перед применением критерия хи-квадрат, объединить несколько категорий, имеющих малые вероятности в одну.

Чтобы получить формулу для , следует подсчитать, сколько групп из *k* чисел, расположенных между 0 и , имеют точно r различных элементов, и разделить это число на .

**Исходный текст программы**

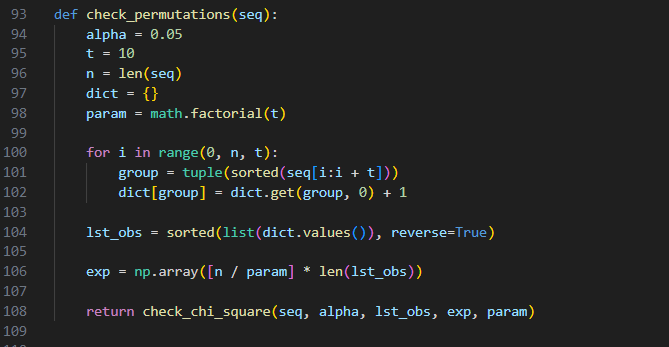
****

# **Критерий перестановок**

Последовательность разбивается на *n* групп по *t* элементов в каждой:

Элементы в каждой группе можно упорядочивать *t*! различными способами. Подсчитывается число групп с любым возможным порядком и применяется хи-квадрат критерий с возможными категориями и вероятностью 1/*t*! для каждой категории. Например, для , существует две категории: или . Для таких категорий будет уже шесть: или , , или , и т.д., …, или . В этом критерии предполагается, что не могут быть равны между собой.

**Исходный текст программы**

****

# **Критерий монотонности**

Последовательность можно проверить на предмет равномерности распределения монотонных серий чисел.

Суть метода в том, чтобы проверить длины всех восходящих (нисходящих) серий в последовательности и подсчитать для них следующую статистику

где *n* – длина последовательности, а матрицы коэффициентов следующие:

Далее, к полученной статистике применяем критерий хи-квадрат с шестью степенями свободы, когда *n* – большое (например, больше 4000).

В данном случае нельзя использовать к длинам серий критерий хи-квадрат, так как проявляется закономерность чередования длинных серий с короткими: вероятность того, что после длинной последовательности будет короткая, выше, чем вероятность длинной серии после длинной. А по условиям критерия хи-квадрат значения проверяемых величин должны быть независимы.

А также элементы матрицы А приведены приблизительно.

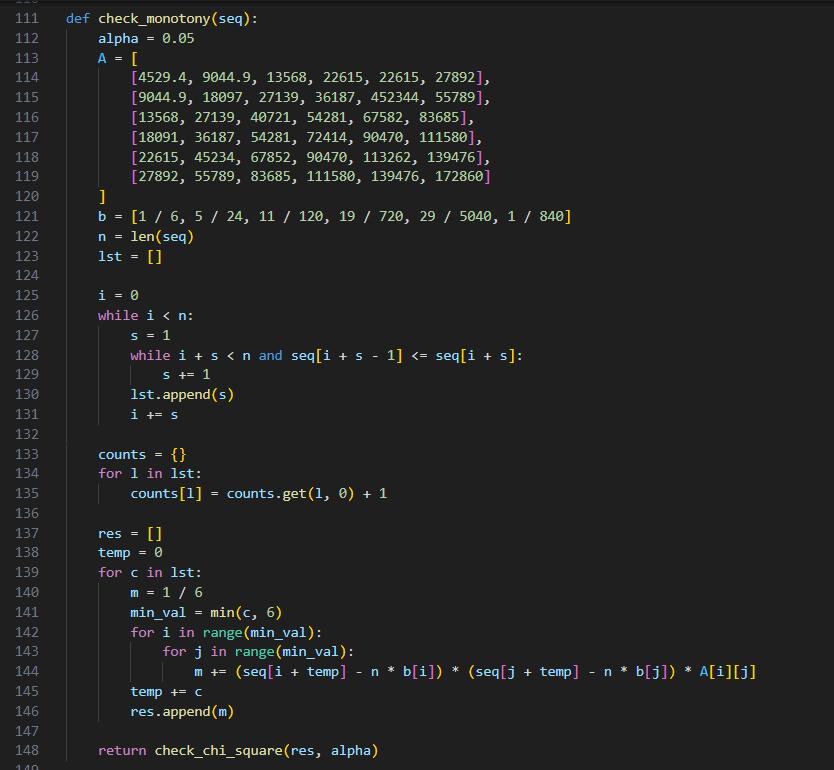
Для решения проблемы чередования длинных серий с короткими сериями можно сделать следующее.

1. «Выбрасываем» элемент последовательности, который следует непосредственно за серией.

2. Если 𝑥𝑗 больше 𝑥𝑗+1, то начнем следующую серию с 𝑥𝑗+2.

3. Мы получаем серии, длины которых независимы и, поэтому, можно использовать критерий хи-квадрат.

**Исходный текст программы**

****

# **Критерий конфликтов**

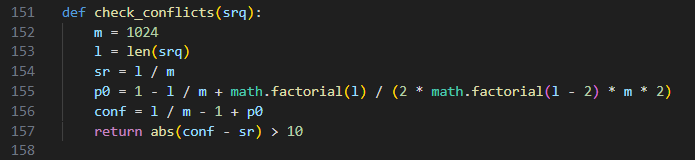
Предположим, что у нас *m* урн и *n* шаров, причем *m* значительно больше *n*. Если разместить шары в урнах наугад, то некоторые урны останутся пустыми, а в некоторых будет более одного шара. Когда в одну урну попадает больше одного шара, то говорят, что произошел «конфликт». Критерий конфликтов состоит в подсчете и оценке количества конфликтов.

Рассмотрим пример, когда , а . В среднем, число урн, приходящихся на один шар – 64. Вероятность того, что в конкретную урну попадет ровно *k* шаров, равна

отсюда, среднее число конфликтов в урне вычисляется по формуле

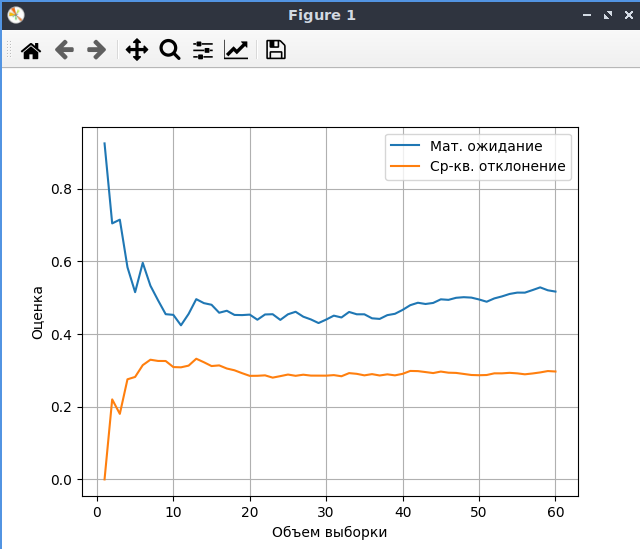
Так как – маленькое число, получим, что общее среднее число конфликтов во всех *m* урнах намного меньше

Этот критерий хорош, когда ГПСЧ выдает строки большой размерности.



1. **Тестирование**

Пример полученного графика для случайной последовательности, полученной с помощью генератора RC4.



**ПРИЛОЖЕНИЕ А  
Листинг программы**

import random

import click

import numpy as np

import math

from scipy.stats import chi2

import matplotlib.pyplot as plt

def gen\_random\_numbers(file\_path, n):

    f = open(file\_path, "w", encoding="utf-8")

    numbers = [random.randint(1, 10) % 2 for \_ in range(n)]

    f.write(" ".join(map(str, numbers)))

def transform\_lst\_nums(num\_lst):

    max\_elem = max(num\_lst) + 1

    return [num / max\_elem for num in num\_lst]

def get\_data\_from\_file(file\_path):

    with open(file\_path, "r") as file:

        data\_line = file.readline()

        num\_lst = list(map(int, data\_line.strip().split()))

    return transform\_lst\_nums(num\_lst)

def check\_chi\_square(lst\_num, alpha=0.05, lst\_=None, exp=None, param=None):

    if param is None:

        param = len(np.unique(lst\_num))

    if lst\_ is None:

        \_, lst\_ = np.unique(lst\_num, return\_counts=True)

    if exp is None:

        exp = np.array([len(lst\_num) / param] \* param)

    chi = np.sum((lst\_ - exp) \*\* 2 / exp)

    stat = chi2.ppf(1 - alpha, param - 1)

    return chi > stat

def check\_series(numbers):

    d = 16

    alpha = 0.05

    param = d \*\* 2

    res = [0] \* param

    for j in range(len(numbers) // 2):

        res[int(numbers[2 \* j] \* d) \* d + int(numbers[2 \* j + 1] \* d)] += 1

    exp = np.array(param \* [len(numbers) / (2 \* param)])

    return check\_chi\_square(numbers, alpha, res, exp, param)

def check\_interval(numbers, alpha=0.05, n=256, t=10, a=0.5, b=1):

    j, s, c\_r = -1, 0, np.array([0] \* (t + 1))

    while s < n:

        r = 0

        j += 1

        while j < len(numbers) and a <= numbers[j] <= b:

            r += 1

            j += 1

        c\_r[min(r, t)] += 1

        s += 1

    p = b - a

    arr\_1 = [n \* p \* pow(1.0 - p, r) for r in range(t)]

    arr\_2 = [n \* pow(1.0 - p, t)]

    exp = arr\_1 + arr\_2

    return check\_chi\_square(numbers, alpha, np.array(c\_r), np.array(exp), t + 1)

def check\_partitions(seq):

    alpha = 0.05

    n = 100

    param = int(10000 / n)

    r = np.array([0] \* (param + 1))

    for i in range(n):

        r[len(np.unique(seq[param \* i: param \* (i + 1)]))] += 1

    p = []

    s = 1

    for i in range(param + 1):

        d = 100

        p\_i = d

        for j in range(1, i):

            p\_i \*= d - j

        p.append(p\_i / pow(d, param) \* s)

    dk\_lst = np.array([math.comb(param + i - 1, i) / pow(d, param) for i in range(param + 1)])

    return check\_chi\_square(seq, alpha, dk\_lst[1:], p[1:], param)

def check\_permutations(seq):

    alpha = 0.05

    t = 10

    n = len(seq)

    dict = {}

    param = math.factorial(t)

    for i in range(0, n, t):

        group = tuple(sorted(seq[i:i + t]))

        dict[group] = dict.get(group, 0) + 1

    lst\_obs = sorted(list(dict.values()), reverse=True)

    exp = np.array([n / param] \* len(lst\_obs))

    return check\_chi\_square(seq, alpha, lst\_obs, exp, param)

def check\_monotony(seq):

    alpha = 0.05

    A = [

        [4529.4, 9044.9, 13568, 22615, 22615, 27892],

        [9044.9, 18097, 27139, 36187, 452344, 55789],

        [13568, 27139, 40721, 54281, 67582, 83685],

        [18091, 36187, 54281, 72414, 90470, 111580],

        [22615, 45234, 67852, 90470, 113262, 139476],

        [27892, 55789, 83685, 111580, 139476, 172860]

    ]

    b = [1 / 6, 5 / 24, 11 / 120, 19 / 720, 29 / 5040, 1 / 840]

    n = len(seq)

    lst = []

    i = 0

    while i < n:

        s = 1

        while i + s < n and seq[i + s - 1] <= seq[i + s]:

            s += 1

        lst.append(s)

        i += s

    counts = {}

    for l in lst:

        counts[l] = counts.get(l, 0) + 1

    res = []

    temp = 0

    for c in lst:

        m = 1 / 6

        min\_val = min(c, 6)

        for i in range(min\_val):

            for j in range(min\_val):

                m += (seq[i + temp] - n \* b[i]) \* (seq[j + temp] - n \* b[j]) \* A[i][j]

        temp += c

        res.append(m)

    return check\_chi\_square(res, alpha)

def check\_conflicts(srq):

    m = 1024

    l = len(srq)

    sr = l / m

    p0 = 1 - l / m + math.factorial(l) / (2 \* math.factorial(l - 2) \* m \* 2)

    conf = l / m - 1 + p0

    return abs(conf - sr) > 10

def math\_expectation\_and\_deviation(numbers):

    return numbers.mean(), numbers.std()

@click.command()

@click.option("/file", default=None, help="Имя файла с входной последовательностью")

def main(file):

    if file is None:

        gen\_random\_numbers("input\_file.txt", 10\_000)

        numbers = np.array(get\_data\_from\_file("input\_file.txt"))

    else:

        numbers = np.array(get\_data\_from\_file(file))

    math\_expectation, deviation = math\_expectation\_and\_deviation(numbers)

    try:

        print(f"Мат. ожидание: {math\_expectation}")

        print(f"Cр-кв. отклонение: {deviation}")

        print(f"Критерий хи-квадрат: {check\_chi\_square(numbers)}")

        print(f"Критерий серийт: {check\_series(numbers)}")

        print(f"Критерий интервалов {check\_interval(numbers)}")

        print(f"Критерий разбиений: {check\_partitions(numbers)}")

        print(f"Критерий перестановок: {check\_permutations(numbers)}")

        print(f"Критерий монотонности: {check\_monotony(numbers)}")

        print(f"Критерий конфликтов: {check\_conflicts(numbers)}")

    except Exception as err:

        print(f"В процессе произошла ошибка: {err}")

    size, mean, std = [], [], []

    for i in range(len(numbers)):

        size.append(i + 1)

        mean.append(numbers[:i + 1].mean())

        std.append(numbers[:i + 1].std())

    plt.xlabel("Объем выборки")

    plt.ylabel("Оценка")

    plt.plot(size, mean, label="Мат. ожидание")

    plt.plot(size, std, label="Cр-кв. отклонение")

    plt.legend()

    plt.grid(True)

    plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

    main()