Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«САРАТОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИМЕНИ Н.Г.ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

**ТЕОРИЯ ПСЕВДОСЛУЧАЙНЫХ ГЕНЕРАТОРОВ**

**ОТЧЕТ ПО ПРАКТИЧЕСКОМУ ЗАДАНИЮ 2**

студентки 5 курса 531 группы

факультета компьютерных наук и информационных технологий

*Гендляра Сергея Максимовича*

фамилия, имя, отчество

Научный руководитель

Ст. преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_ И.И. Слеповичев

подпись, дата

Саратов 2024

## Преобразование ПСЧ к заданному распределению.

Описание задания:

Создать программу для преобразования последовательности ПСЧ в другую последовательность ПСЧ с заданным распределением:

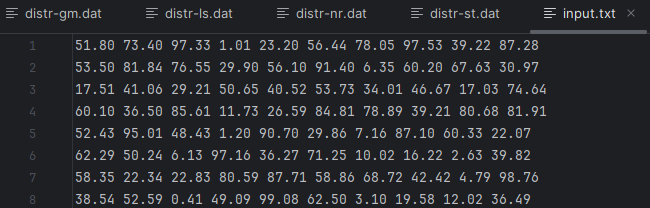
1. Стандартное равномерное с заданным интервалом;
2. Треугольное распределение;
3. Общее экспоненциальное распределение;
4. Нормальное распределение;
5. Гамма распределение (для параметра *c*=*k*);
6. Логнормальное распределение;
7. Логистическое распределение;
8. Биномиальное распределение.

Название программы: **rnc.exe**

### На входе

Текстовый файл с десятичными числами (разделитель – любой), интервал преобразуемых значений, параметры распределения.

Для управления приложением предлагается следующий формат параметров командной строки:

--f:<имя\_файла> - имя файла с входной последовательностью.  
В качестве параметра f будет использован файл input.txt  


Содержащий 10000 различных десятичных чисел

--d:<распределение> - код распределения для преобразования последовательности. Рекомендуется использовать следующие коды распределений:

* st – стандартное равномерное с заданным интервалом;
* tr – треугольное распределение;
* ex – общее экспоненциальное распределение;
* nr – нормальное распределение;
* gm – гамма распределение;
* ln – логнормальное распределение;
* ls – логистическое распределение;
* bi – биномиальное распределение.

--p1:<параметр1> - 1-й параметр, необходимый, для генерации ПСЧ заданного распределения.

--p2:<параметр2> - 2-й параметр, необходимый, для генерации ПСЧ заданного распределения.

--p3:<параметр3> - 3-й параметр, необходимый, для генерации ПСЧ гамма-распределением.

### На выходе

Текстовый файл distr-xx.dat с преобразованными числами, где <xx> – код распределения.

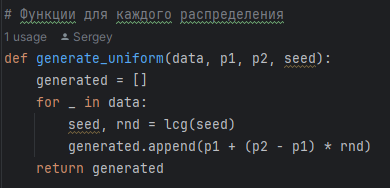
**Стандартное равномерное распределение с заданным интервалом.  
Описание алгоритма.**

Используется следующая формула:

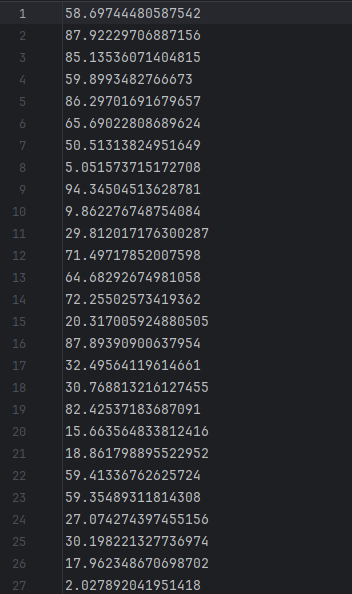
**Запуск программы с параметрами.**

rnc.exe --f input.txt --d st --p1 0 --p2 100

**Исходный код программы.**

****

**Пример работы программы.**

****

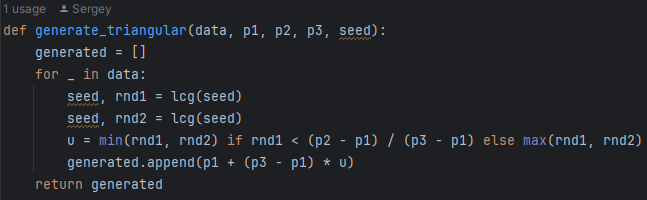
**Треугольное распределение.  
Описание алгоритма.**

Используется следующая формула: .

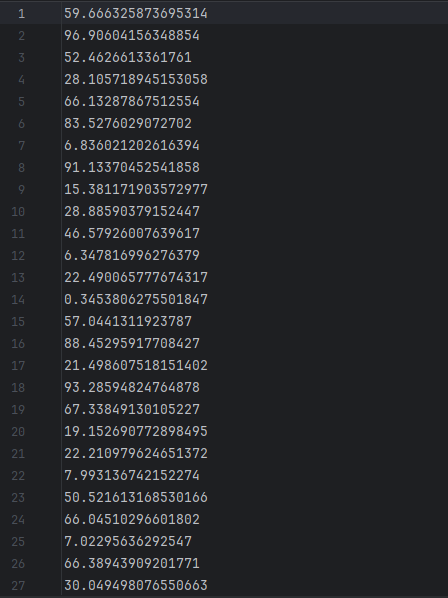
**Запуск программы с параметрами.**

rnc.exe --f input.txt --d tr --p1 0 --p2 50 --p3 100

**Исходный текст программы.**

****

**Пример работы программы.**

****

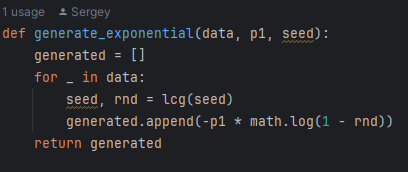
**Общее экспоненциальное распределение.  
 Описание алгоритма.**

Используется следующая формула:

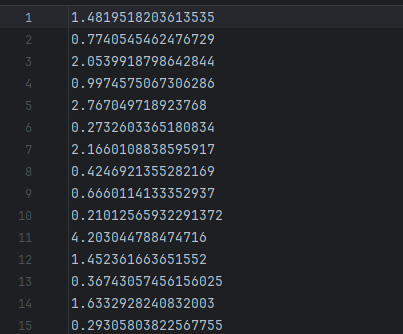
**Запуск программы с параметрами.**

rnc.exe --f input.txt --d ex --p1 1.5

**Исходный текст программы.**

****

**Пример работы программы.**



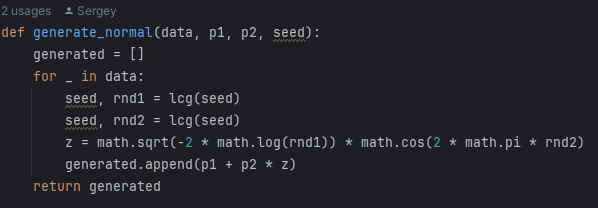
**Нормальное распределение.  
Описание алгоритма.**

Используется следующие формулы:

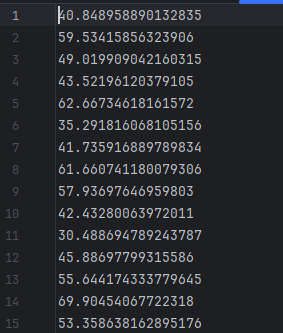
**Запуск программы с параметрами.**

rnc.exe --f input.txt --d nr --p1 50 --p2 10

**Исходный текст программы.**

****

**Пример работы программы.**

****

**Гамма распределение.  
Описание алгоритма.**

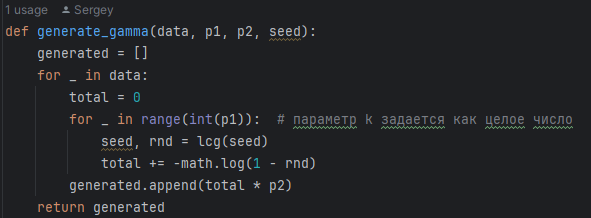
Используется следующая формула:

.

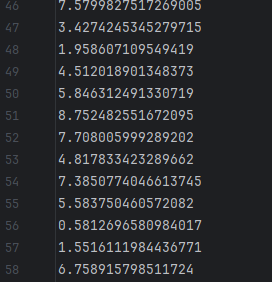
**Запуск программы с параметрами.**

rnc.exe --f input.txt --d gm --p1 2 --p2 2.0

**Исходный текст программы.**

****

**Пример работы программы.**

****

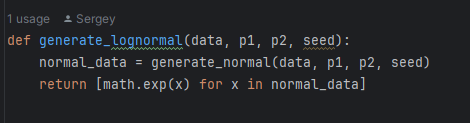
**Алгоритм 6. Логнормальное распределение.  
Описание алгоритма.**

Используется следующая формула:

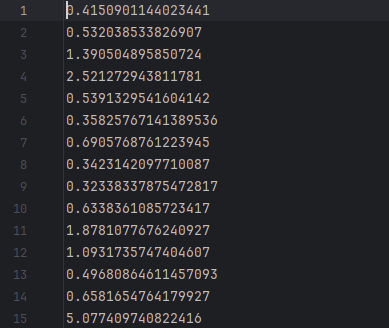
**Запуск программы с параметрами.**

rnc.exe --f input.txt --d ln --p1 0 --p2 1

**Исходный текст программы.**

****

**Пример работы программы.**



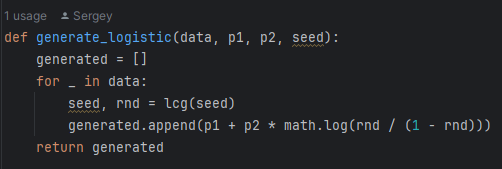
**Логистическое распределение.  
Описание алгоритма.**

Используется следующая формула:

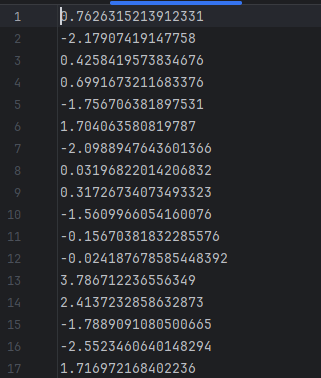
**Запуск программы с параметрами.**

rnc.exe --f input.txt --d ls --p1 0 --p2 1

**Исходный текст программы.**

****

**Пример работы программы.**

****

**Биномиальное распределение.  
Описание алгоритма.**

Используется следующая формула:

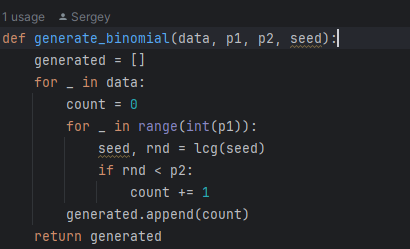
где .

Функция распределения вычисляется по следующей формуле:

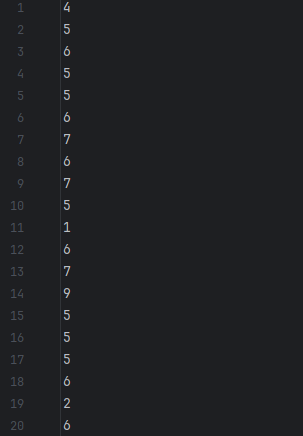
**Запуск программы с параметрами.**

rnc.exe --f input.txt --d bi --p1 10 --p2 0.5

**Исходный текст программы.**

****

**Пример работы программы.**

****

**Приложение А - листинг программы по генерации исходного файла generate\_input.py**

import time

# Линейный конгруэнтный генератор (LCG) для генерации случайных чисел без использования модуля random

def lcg(seed, a=1664525, c=1013904223, m=2 \*\* 32):

seed = (a \* seed + c) % m

return seed, seed / m

# Функция для генерации чисел и записи их в файл

def generate\_input\_file(filename, num\_values=10000, seed=None):

if seed is None:

seed = int(time.time()) # Установка начального значения seed на основе времени

with open(filename, 'w') as file:

for \_ in range(num\_values):

seed, rnd = lcg(seed)

file.write(f"{rnd \* 100:.2f} ") # Число с точностью до 2 знаков, разделитель — пробел

if \_ % 10 == 9: # Добавление новой строки после каждых 10 чисел

file.write("\n")

# Генерация файла input.txt

generate\_input\_file("input.txt")

**Приложение Б - листинг программы rnc.py**

import argparse

import math

import os

import time

# Генерация начального значения для seed на основе текущего времени

seed = int(time.time()) # Генерируем начальный seed один раз

# Линейный конгруэнтный генератор (LCG)

def lcg(seed, a=1664525, c=1013904223, m=2 \*\* 32):

"""Генерирует псевдослучайное число от 0 до 1 на основе линейного конгруэнтного метода"""

seed = (a \* seed + c) % m

return seed, seed / m

# Функции для каждого распределения

def generate\_uniform(data, p1, p2, seed):

generated = []

for \_ in data:

seed, rnd = lcg(seed)

generated.append(p1 + (p2 - p1) \* rnd)

return generated

def generate\_triangular(data, p1, p2, p3, seed):

generated = []

for \_ in data:

seed, rnd1 = lcg(seed)

seed, rnd2 = lcg(seed)

u = min(rnd1, rnd2) if rnd1 < (p2 - p1) / (p3 - p1) else max(rnd1, rnd2)

generated.append(p1 + (p3 - p1) \* u)

return generated

def generate\_exponential(data, p1, seed):

generated = []

for \_ in data:

seed, rnd = lcg(seed)

generated.append(-p1 \* math.log(1 - rnd))

return generated

def generate\_normal(data, p1, p2, seed):

generated = []

for \_ in data:

seed, rnd1 = lcg(seed)

seed, rnd2 = lcg(seed)

z = math.sqrt(-2 \* math.log(rnd1)) \* math.cos(2 \* math.pi \* rnd2)

generated.append(p1 + p2 \* z)

return generated

def generate\_gamma(data, p1, p2, seed):

generated = []

for \_ in data:

total = 0

for \_ in range(int(p1)): # параметр k задается как целое число

seed, rnd = lcg(seed)

total += -math.log(1 - rnd)

generated.append(total \* p2)

return generated

def generate\_lognormal(data, p1, p2, seed):

normal\_data = generate\_normal(data, p1, p2, seed)

return [math.exp(x) for x in normal\_data]

def generate\_logistic(data, p1, p2, seed):

generated = []

for \_ in data:

seed, rnd = lcg(seed)

generated.append(p1 + p2 \* math.log(rnd / (1 - rnd)))

return generated

def generate\_binomial(data, p1, p2, seed):

generated = []

for \_ in data:

count = 0

for \_ in range(int(p1)):

seed, rnd = lcg(seed)

if rnd < p2:

count += 1

generated.append(count)

return generated

# Обработка параметров командной строки

def parse\_arguments():

parser = argparse.ArgumentParser(description="Преобразование ПСЧ в другое распределение.")

parser.add\_argument("--f", dest="filename", required=True, help="Имя файла с входной последовательностью.")

parser.add\_argument("--d", dest="distribution", required=True,

choices=["st", "tr", "ex", "nr", "gm", "ln", "ls", "bi"], help="Код распределения.")

parser.add\_argument("--p1", dest="p1", type=float, required=True, help="Первый параметр.")

parser.add\_argument("--p2", dest="p2", type=float, required=False, help="Второй параметр.")

parser.add\_argument("--p3", dest="p3", type=float, required=False,

help="Третий параметр (для треугольного и гамма-распределений).")

return parser.parse\_args()

# Основная функция

def main():

args = parse\_arguments()

# Чтение входных данных

with open(args.filename, 'r') as file:

data = [float(num) for line in file for num in line.split()]

# Определение распределения и генерация значений

global seed # Используем глобально заданный начальный seed

if args.distribution == "st":

transformed\_data = generate\_uniform(data, args.p1, args.p2, seed)

elif args.distribution == "tr":

transformed\_data = generate\_triangular(data, args.p1, args.p2, args.p3, seed)

elif args.distribution == "ex":

transformed\_data = generate\_exponential(data, args.p1, seed)

elif args.distribution == "nr":

transformed\_data = generate\_normal(data, args.p1, args.p2, seed)

elif args.distribution == "gm":

transformed\_data = generate\_gamma(data, args.p1, args.p2, seed)

elif args.distribution == "ln":

transformed\_data = generate\_lognormal(data, args.p1, args.p2, seed)

elif args.distribution == "ls":

transformed\_data = generate\_logistic(data, args.p1, args.p2, seed)

elif args.distribution == "bi":

transformed\_data = generate\_binomial(data, int(args.p1), args.p2, seed)

else:

raise ValueError("Неподдерживаемое распределение")

# Сохранение результата

output\_filename = f"distr-{args.distribution}.dat"

with open(output\_filename, 'w') as file:

file.write("\n".join(map(str, transformed\_data)))

print(f"Результат сохранен в {output\_filename}")

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

main()